CAMLEYE
A Functional Implementation of a Figure-Ground Discrimination

Giulio Manzonetto[a*], Giulio Marcon[a†], Anna Sperotto[a‡].

[a]{gmanzone, gmarcon, asperott}@dsi.unive.it

Abstract. In this document we describe the problems of Artificial Vision that are solved by our software Camleye and we discuss our implementation choices. We also provide a short user manual.

Keywords: Figure ground discrimination, edge detection, perceptual grouping, mean field annealing, simulated annealing.

Contents

*Giulio Manzonetto, matriculation number 794652
†Giulio Marcon, matriculation number 793939
‡Anna Sperotto, matriculation number 787327

1. Introduction

We now define the problem of perceptual grouping that we want to solve with our software 'Camleye'. Camleye is supposed to transform a raster image into an image containing only zero and one in those points that - in some way - identify the figures contained in the initial image. To complete this task Camleye performs a preliminary step: the *edge detection*.

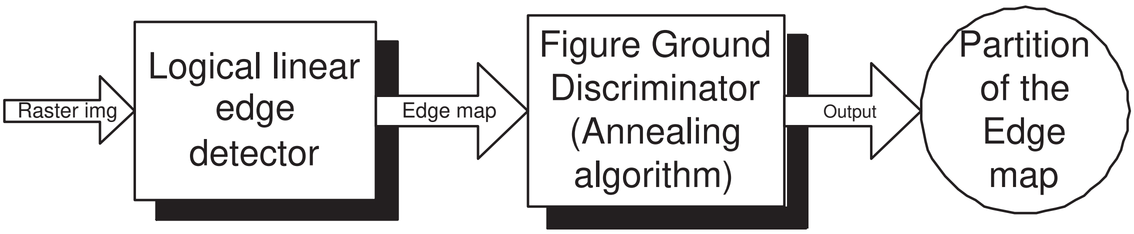


Figura 1. Description of the problem and presentation of the two steps.

The edge detection process consists on identifying the edges of the figure where the intensity of the image change brusquely. More precisely, our boundaries also have a *sense* and a *direction*.

After this first step, we determine a partition of the set of edges that have been found, classifying the edges either as *figure* or as *ground*.

To make our software more user-friendly we decided to create two different programs whose behaviours reflect the two operations described above.

The algorithm that we apply to perform the edge-detection is, essentially, the one available on-line at the web address [5], which is presented as a portable implementation of the method described in [4].

2. Linear Logic Operators for Edge Detection

The approach adopted by Iverson in [6] allows to define local operators responding only in those points of the image where there is a curve with a suitable geometry. As in the other classic algorithm for edge detection, and more generally for image processing, the phase of post-processing is avoided (except for a simple threshold). The operators are based on linear convolutions formalizing in an algebra the reasoning on the results of convolutions. This algebra (which is, in fact, a Boolean algebra having $\curlywedge$ as *meet* and

$\Downarrow$ as *join*) is based on logical operations, which are then approximated to obtain noise tolerance.

The operators which are defined to recognize the boundaries fetch the points of the image satisfying a normal and a tangential condition. The normal condition corresponds to a strong contrast with respect to a certain section, and the tangential condition to continuity. Every operator has a spatial support of oval shape having in its center the point of the image which is considered; in our case the dimension of the oval is approximately of 20 pixels along the tangential component and 10 pixels along the normal one.

The operator we used is the following:

$$\Phi = T \times N = (t^- \Uparrow t^+) \times (n'_c \Uparrow n''_l \Uparrow n''_r \Uparrow n^4_l \Uparrow n^4_r)$$

where:

- $t^-(x) = G_{2.5}(x)\sigma_{5.0}(-x) + 0.2G'_{2.5}(x)$
- $t^+(x) = G_{2.5}(x)\sigma_{5.0}(x) - 0.2G'_{2.5}(x)$
- $n'_c(x) = G'_{1.4}(x)$
- $n''_l(x) = G''_{1.4}(x + 0.8)$
- $n''_r(x) = -G''_{1.4}(x - 0.8)$
- $n^4_l(x) = -G''_{1.4}(x + 0.8)$
- $n^4_r(x) = G''_{1.4}(x - 0.8)$

$G_\sigma(x), G'_\sigma(x), \ldots$ denote respectively the Gaussian function and its derivative; $\sigma_\rho(x)$ is defined in [4].

The operator has just two tangential components since the qualitative results obtained with four tangential components, as in Iverson's implementation, are not good enough to justify the additional computational cost.

The reimplementation was fairly difficult, even if we had the original Iverson's code. In particular, this implementation differs from the one described in article and, rather, uses some definitions and technics introduced in Iverson's PhD thesis. Moreover, the parameters which are used and the different ways in which functions are approximated suggest that it has been used a *trial and error* approach instead of an attempt of optimisation: computing precisely those functions we obtained manifestly nasty results.

Particularly significant is the presence in the original code of an operator $\Uparrow$ approximated for five arguments, whose behaviour is completely different from that we should have if we apply it more times, thus:

$$\Uparrow_5(x_1, x_2, x_3, x_4, x_5) \neq \Uparrow(x_1, \Uparrow(x_2, \Uparrow(x_3, \Uparrow(x_4, x_5))))$$

The framework of linear/logic operators has been completely reimplemented and the versatility of the programming language allowed us to write 'just' 350 lines of code against the 5000 of the original.

Since we use two functions for the tangential component and five for the normal component, we apply, for each pixel of the image, ten convolution operators having a medium dimension of 20×10 pixel. Moreover, is important to underline that this operation individuates all edges having a determined angle: so we need to repeat these operations for each angle for which we want to find edges (the number of parts in which we divide the angle is a parameter of the program). Since we have a vast amount of computations, we did some optimizations. The most relevant are:

- *edges in opposite sense*: to find all edges with the same direction but opposite sense we can use the property of complementation of the Boolean algebra performing half computations; convolutions are applied just one time, $\underset{+}{\triangle}$ and $\overset{+}{\triangledown}$ are applied applied directly on a certain sense and reversed on the opposite sense;
- *convolution*: the masks we used work with integers values rather than with floating points as in the original version; masks are appropriately normalized to obtain proper answers.
- *loop unrolling*: the loops have been translate in recursive functions and have been adapted to minimize the number of basic operations; other optimizations of the processor (inlining) contributed to make the program efficient.

3. Figure Ground Discrimination as an Optimization Problem

The *Annealing* algorithm was born as a method of simulation of solids temper. Annealing is a heat treatment wherein a material is altered, causing changes in its properties such as strength and hardness. It is a process that produces conditions by heating to above the re-crystallization temperature and maintaining a suitable temperature, and then gradually reducing the temperature.

The algorithm starts at high initial temperature. At this temperature, nearly 100% of the tested transitions are accepted. The procedure is applied at constant temperature in order to reach a near thermal equilibrium state at this temperature. Then the temperature is slightly decreased and the procedure is applied again. This process is repeated until the system is frozen, i.e., when the energy cannot significantly be reduced anymore.

The two annealing algorithms that we will describe depend on suitable parameters which must be estimated by the user to obtain good results.

3.1. From edge to edgels

A first problem, which is common to all annealing algorithms we will consider, is to transform the map of edges which is taken in input into a map of *edgels*.

An edge can be viewed, intuitively, as a segment which is completely contained into a unitary cell. This cell is positioned at some point within the image.

An edge e is then characterized by its position (x_e, y_e), its angular coefficient d_e and its intensity i_e.

An edgel is something very similar: indeed it is characterized by the same quantities and can be thought like a straight line of arbitrary length.

As described in [1], if in the edge-map there is a set of adjacent edges constituting a line, then they can be substituted by a single edgel having as position their medium point, as angular coefficient the angular coefficient of the edges and as intensity the average of their intensities.

For each input edge we have to compute the position of an edge that could be its *extension*, to check whether such an edge exists, to check if it has the same angular coefficient and if there are other edges that can be grouped to form a single edgel. To perform this operation efficiently, we use *tail recursive* functions and we collect the edges in a list which is ordered w.r.t. the lexicographical order on the triples $(angle, x, y)$.

Then, we save the correspondence between every edgel and the set of edges that generated it. This correspondence will be used at the end to come back from edgels to edges. Indeed, the expected output should be a partition of the edges, not of the edgels.

A feature that we implemented in our program is the repression of those edgels which are composed by less than a certain number of edges in the phase preceding the optimization process.

3.2. The interaction matrix

Another step which is common to both algorithms is the creation of a matrix containing the interaction coefficients among edges. This matrix is symmetric by construction. For this reason we developed a library to handle efficiently this kind of data structures. To compute the values of the interaction matrix we first need to compute the coefficients of:

- cocircularity: it constrains the shapes to be as circular as possible or, as a special case, as linear as possible;
- smoothness: it enforces shapes with low curvature;
- proximity: it restricts the interaction to occur in between nearby edgels;
- contrast: it increases the interaction between edgels with similar contrast.

In the case of Meanfield annealing we need to create also the vector of external fields δ.

3.3. Mean field annealing algorithm

The *mean field annealing* algorithm is quite simple.

1. a solution vector $\mu^{(0)}$ is randomly generated;
2. at every iteration i the vector $\mu^{(i)}$ is updated in a synchronous or synchronous way, and the temperature is decremented;

- every 6 iterations the exit test is performed: if the average of the components of the vector $\mu^{(i)}$ is near to 1 or -1 the algorithm terminates, otherwise another iteration is done,
- when the convergence is reached the components $\mu_i^{(r)}$ of the solution vector are set either to 1 (if $\mu_i^{(r)} > 0$) or to -1 (if $\mu_i^{(r)} < 0$).

The difference between the synchronous and asynchronous update of the solution vector consists on the fact that, in the synchronous case we have to update all its components μ_i at every iteration, whilst in the asynchronous case we randomly update a single component.

3.4. Simulated annealing algorithm

Simulated annealing is a Monte Carlo algorithm. At high temperature the algorithm behaves like a *random search*. At low temperature the Simulated Annealing becomes similar to *steepest descent* methods. The optimal solutions are found in the zone of the domain which is more promising.

The Simulated Annealing algorithm for a general optimization problem can be described as follows:

- Generate an initial configuration x_0 with value of the energy function E_0. Select an initial value t_0 for the temperature,
- For each value of the temperature, perform the following steps:
  - generate an admissible configuration through a small random change of the initial configuration (one component of the solution vector is switched from -1 to 1 or vice versa). Evaluate the energy difference ΔE between the two configurations.
  - If $\Delta E \leq 0$, then the configuration has a value of the objective function which is smaller than the previous one. The solution is accepted and the old one is discarded,
  - If $\Delta E > 0$, then the configuration has a value of the objective function which is bigger than the previous one. The configuration is accepted with probability $exp\left(-\frac{\Delta E}{T}\right)$.
- This process is iterated until the thermic equilibrium is reached, then the temperature is decremented.

The process terminates when the percentage of state transitions accepted is less then 1.

4. A Parallel Implementation of MFA

Experimental results show that the algorithms we implemented are very expensive in terms of time. For this reason we ha decided to implement a parallel version of synchronous mean field annealing.

We suppose to have k processors numbered from 0 to $k-1$. The number of the i-th machine is called the *rank* of that machine.

The process running in the machine of rank 0 (which is the machine where the software has been launched) extracts from the input file the list of edges, converts it in a list of edgels and creates an interaction matrix $J_{n \times n}$. This matrix is then divided in k sub matrixes with n columns and $\frac{n}{k}$ lines[4]. When n is not perfectly divisible for the number of machines the last matrix which is created will have $\frac{n}{k} + (n \; mod \; \frac{n}{K})$ rows. We show an example of such a situation in Figure 2.

[4]We use Euclidean division to compute the number of lines.

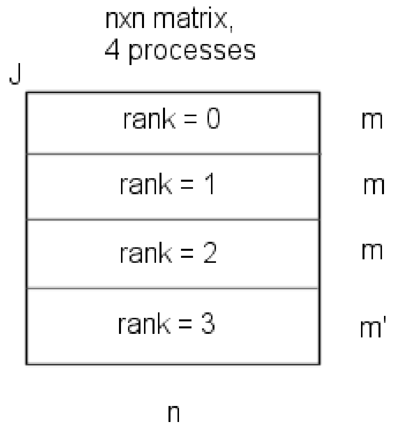


Figura 2. Example of domain decomposition. In this figure the $n \times n$ coefficient matrix is divided among four processors. Thus we have $m = \frac{n}{4}$ and $m' = m + n \ mod \ m$.

This operation is called *domain decomposition*. The sub-matrixes which are built in this step are subsequently distributed to all the processes which are involved.

At this point, every process has a part of the initial matrix J. The lines of the matrix J which are assigned to the process of rank i are those going from index $\left(i * \frac{n}{K}\right)$ to index $\left((i+1) * \frac{n}{K} - 1\right)$.

Thus every process can compute a fragment of the solution vector as explained in Subsection 3.3. At every iteration the processes exchange the computed information and each updates the solution vector (relative to that iteration) merging the parts received from the other processes and its own part.

Subsequently, every process continues the execution of the algorithm on that vector (that at this point is the same for everybody). Notice that the exchanges of information between the processes will be frequent, but the quantity of data which is passed is quite small. As before, the processes execute the test on the exit condition every 6 iterations. Notice that the test will give the same answer on all machines because it is executed on a copy of the same vector. Thus, all processes terminates at the same time.

When the exit condition becomes true, the process of rank 0 creates the output file.

5. Results

In this section we presents the results we have obtained while trying to find the correct parameters for the algorithms.

It has been difficult to obtain good performances (in terms of time). After three months of full-time work we finally obtained a version of the software whose execution time is on the order of minutes instead of hours.

This is the result of a typical execution on a 1Ghz Duron equipped with 512mb of RAM, that required about 10 minutes:

```
$ time ./loglin.opt.exe inout/Paolina.pgm
reading image...
Logical Linear Detection...: 100%
real    3m7.990s
user    0m0.010s
sys     0m0.010s

$ time ./camleye.opt.exe -n 2 inout/Paolina-edges.ps
reading edges... converting 23191 edges into edgels...
done, 8069 edgels created
removing edgels with less than 2 edge... done, 5184 remaining edgels
creating interaction matrix... done
Mean Field Annealing...
iteration: 26976  Annealing...: 100%
done
getting figure and writing out... done, figure has 333 edgels
real    7m0.435s
user    0m0.020s
sys     0m0.010s
```

In order to test our software we mainly used two images. The first one, taken from the archive of test images of the Caltech Vision Group, was also used by Iverson in [6]. The original image of Paolina can be found at the following address `http://www.vision.caltech.edu/html-files/archive.html`. The second one is a synthetic image created to test the figure ground discrimination. The image contains a line and two circles with some noise added pointwise with probability 0.01.

Figura 3. Antonio Canova, *Paolina Borgese*, 1805-1808

(a) (b)

Figura 4. (a) Synthetic image for tests (b) Synthetic image with added noise

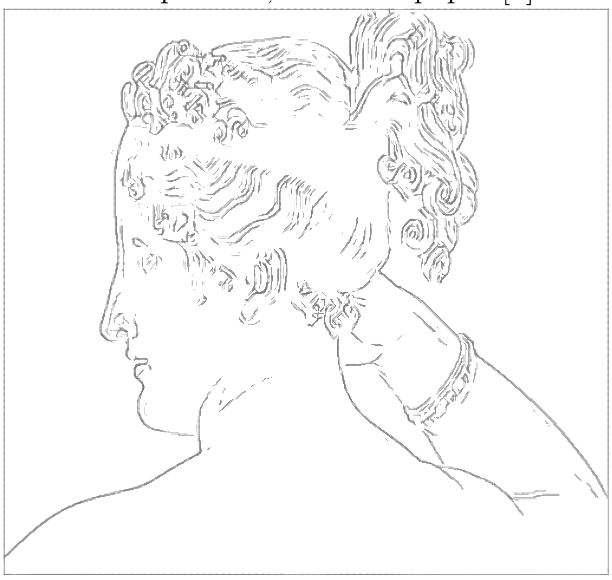
5.1. Boundaries recognition

We now show that tangential operators with two components are suitable for our algorithm of edge detection. These images show the results we have obtained on Paolina with a number of components going from one to eight. We can notice that using a single component we obtain an image with too much noise, whilst four components lead to an image where significant boundaries are missed. Using two components, all significant boundaries are found and the quantity of noise is tolerable.

1 component

2 components, as in the paper [4]

4 components, as in the implementation [5]



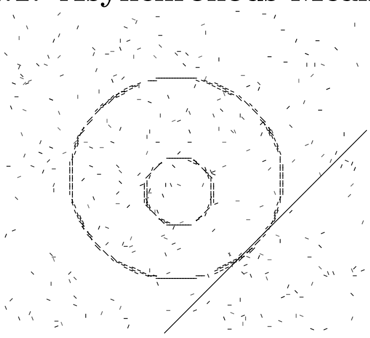
8 components

5.2. Figure-ground discrimination

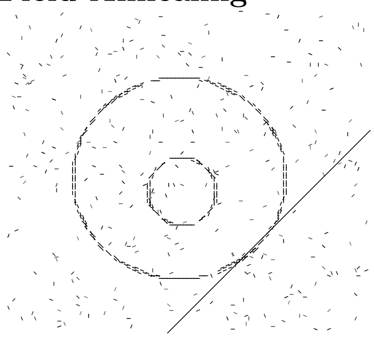
The synthetic image that we have used for the test has 740 edges: 417 edges are already present in the original image and 323 edges constitute the added noise. The pre-processor obtains 495 edgels.

The experimental results show the values that we have used for parameters, the number of iterations done until convergence, the final edgels and the corresponding edges.

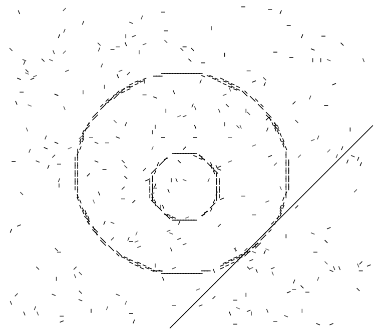
5.2.1. Asynchronous Mean Field Annealing



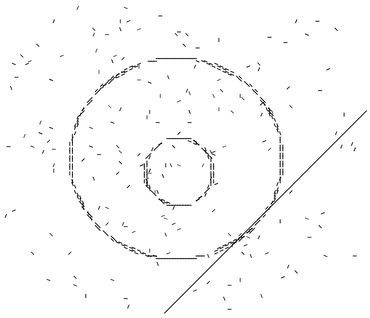
$k = 1.5, \alpha = 0.0055$
$i = 12282$



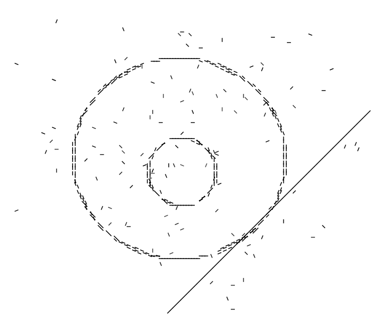
$k = 1.5, \alpha = 0.001$
$i = 12132$



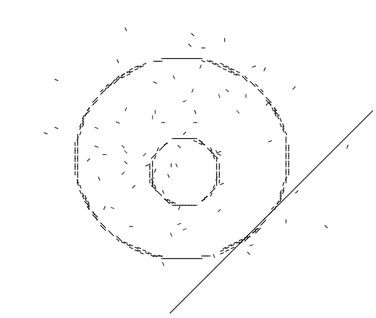
$k = 1.5, \alpha = 0.01$
$i = 11460$



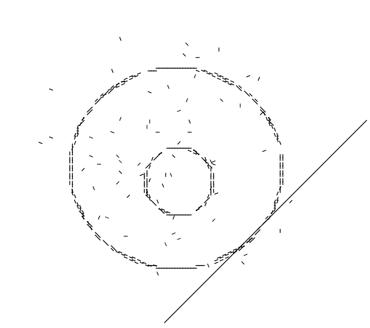
$k = 1.5, \alpha = 0.03$
$i = 11526$



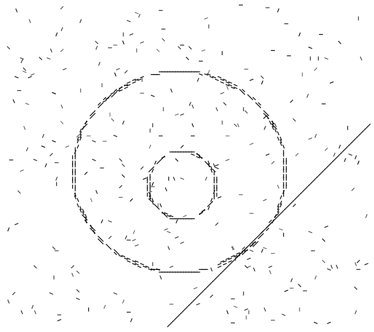
$k = 1.5, \alpha = 0.05$
$i = 13062$



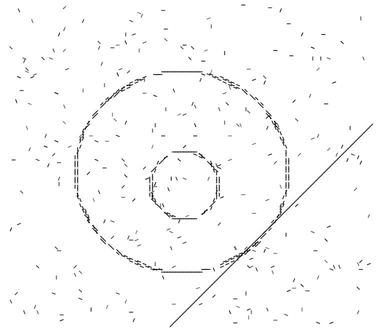
$k = 1.5, \alpha = 0.07$
$i = 11112$



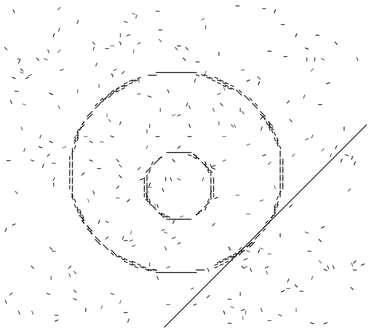
$k = 1.5, \alpha = 0.075$
$i = 12270$



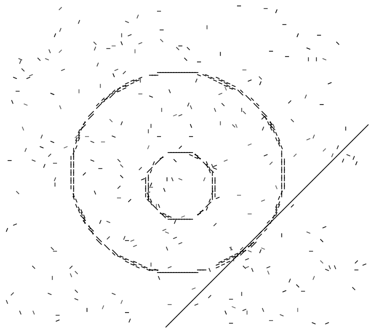
$k = 1, \alpha = 0.0055$
$i = 12504$



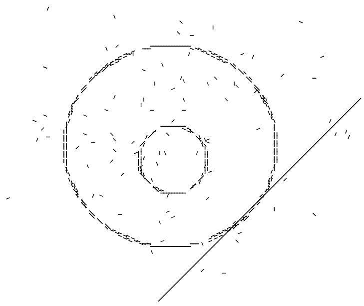
$k = 0.75, \alpha = 0.0055$
$i = 13092$



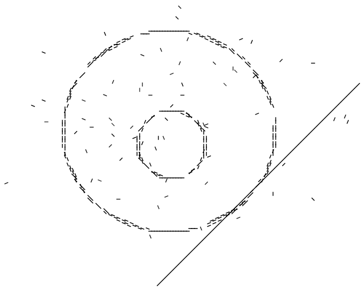
$k = 0.5, \alpha = 0.0055$
$i = 12600$



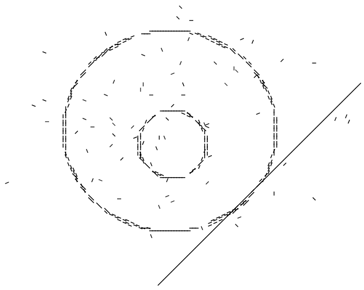
$k = 0.25, \alpha = 0.0055$
$i = 13014$



$k = 0.5, \alpha = 0.04$
$i = 13188$



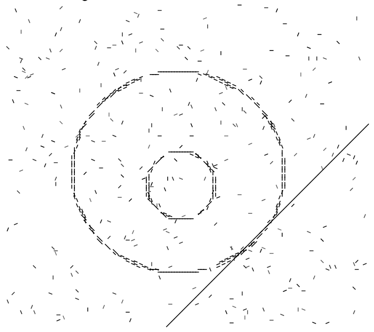
$k = 0.5, \alpha = 0.05$
$i = 12978$



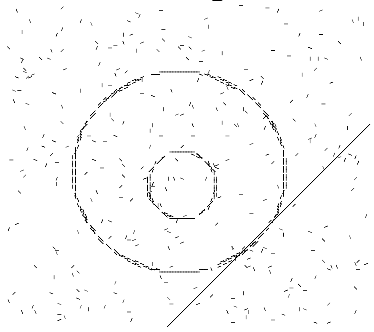
$k = 0.6, \alpha = 0.055$
$i = 12438$

iterations	k	α	final edgels	final edges
12282	1,5	0,0055	465	710
12132	1,5	0,001	487	732
11460	1,5	0,01	441	686
11526	1,5	0,03	349	594
13062	1,5	0,05	288	532
11112	1,5	0,07	242	486
12270	1,5	0,075	235	478
12504	1	0,0055	458	703
13092	0,75	0,0055	454	699
12600	0,5	0,0055	447	692
13014	0,25	0,0055	430	675
13188	0,5	0,04	266	510
12978	0,5	0,05	236	475
12438	0,6	0,055	232	473

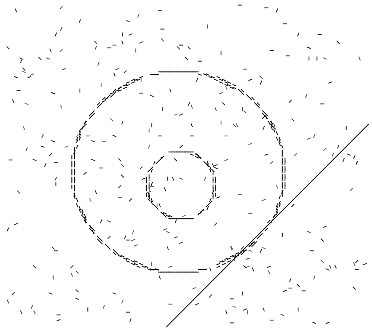
5.2.2. Synchronous Mean Field Annealing



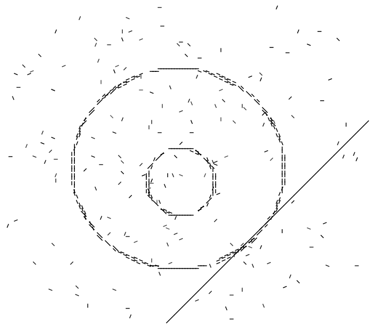
$k = 1.5, \alpha = 0.0055$
$i = 11910$



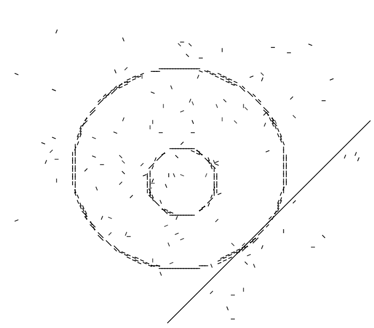
$k = 1.5, \alpha = 0.001$
$i = 11856$



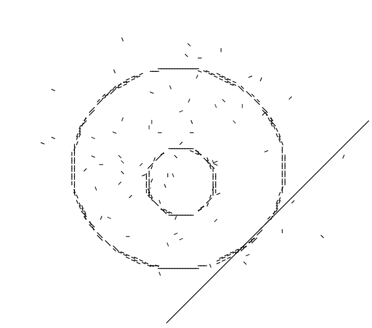
$k = 1.5, \alpha = 0.01$
$i = 10890$



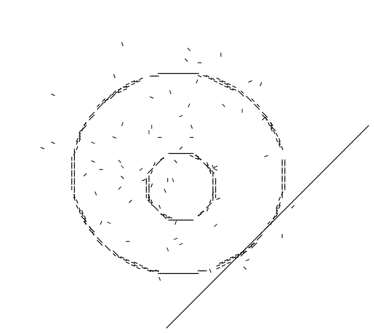
$k = 1.5, \alpha = 0.03$
$i = 10950$



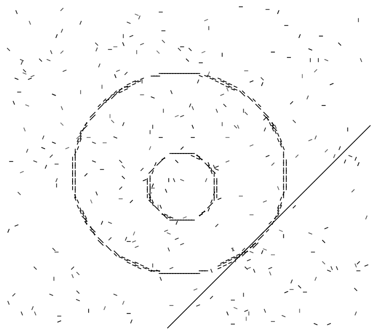
$k = 1.5, \alpha = 0.05$
$i = 12366$



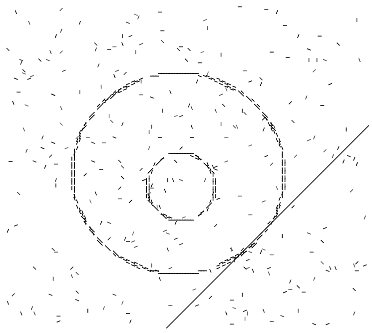
$k = 1.5, \alpha = 0.07$
$i = 10584$



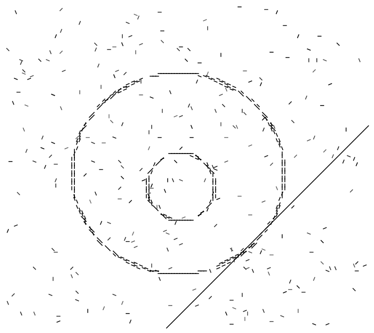
$k = 1.5, \alpha = 0.075$
$i = 11868$



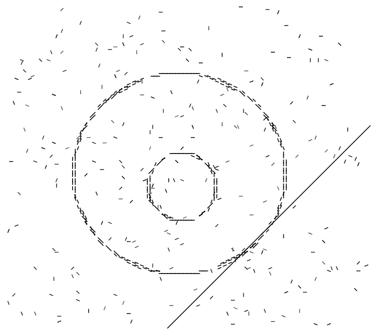
$k = 1, \alpha = 0.0055$
$i = 11994$



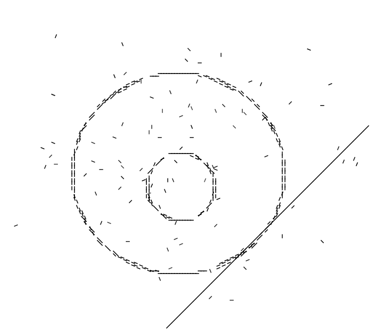
$k = 0.75, \alpha = 0.0055$
$i = 12666$



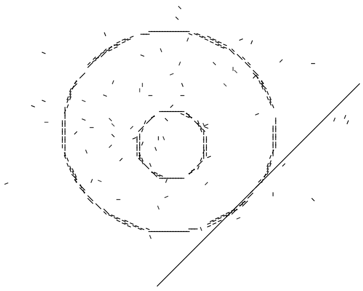
$k = 0.5, \alpha = 0.0055$
$i = 12084$



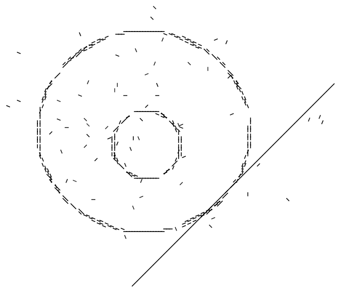
$k = 0.25, \alpha = 0.0055$
$i = 12378$



$k = 0.5, \alpha = 0.04$
$i = 12588$



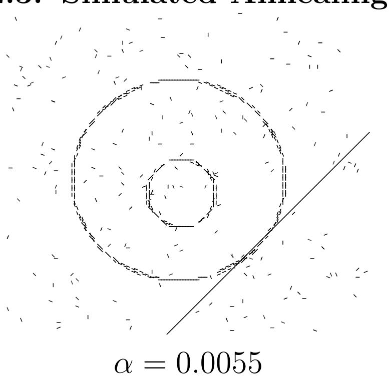
$k = 0.5, \alpha = 0.05$
$i = 12120$



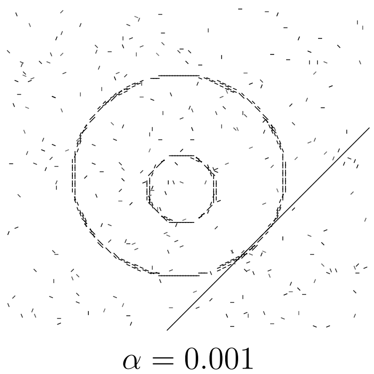
$k = 0.6, \alpha = 0.055$
$i = 11934$

iterations	k	α	edgels	edges
11856	1,5	0,001	487	732
10890	1,5	0,01	441	686
10950	1,5	0,03	349	594
12366	1,5	0,05	288	532
10584	1,5	0,07	242	486
11868	1,5	0,075	235	478
11994	1	0,0055	458	703
12666	0,75	0,0055	454	699
12084	0,5	0,0055	447	692
12378	0,25	0,0055	430	675
12588	0,5	0,04	266	510
12120	0,5	0,05	236	475
11934	0,6	0,055	231	470

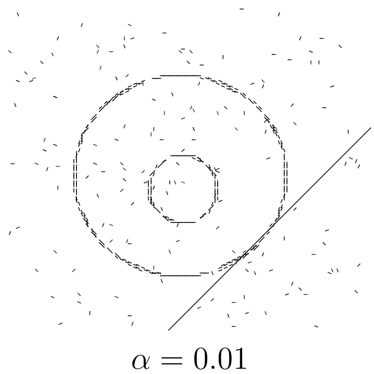
5.2.3. Simulated Annealing



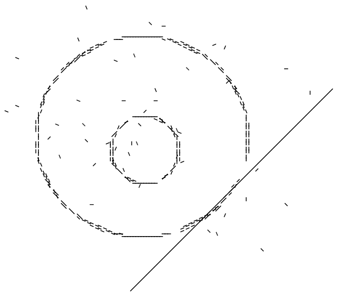
$\alpha = 0.0055$
$i = 509953$



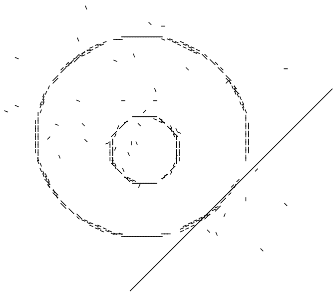
$\alpha = 0.001$
$i = 623826$



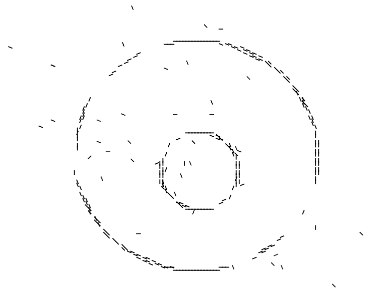
$\alpha = 0.01$
$i = 455492$



$\alpha = 0.03$
$i = 381227$



$\alpha = 0.033$
$i = 396080$



$\alpha = 0.04$
$i = 376276$

iterations	α	final edgels	final edges
509953	0,0055	375	620
623826	0,001	439	684
455492	0,01	335	580
381227	0,03	212	451
396080	0,033	203	441
376276	0.04	164	300

5.2.4. Treating real images with asynchronous MFA

Comparing these results with the preceding ones it is clear that the correct value for the parameter α depends on the number of edgels of the input image. The reason is that in the normalization of the matrix of interaction coefficients we use the parameter α multiplied for the number of edgels.

original image

synchronous MFA, $k = 1.5$, $\alpha = 0.0055$

asynchronous MFA, $k = 1.5$, $\alpha = 0.0055$

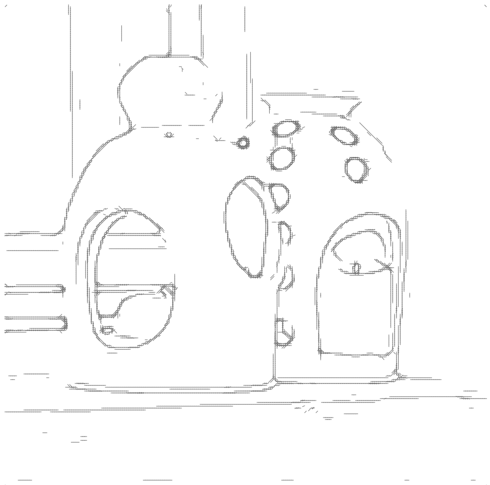
asynchronous MFA, $k = 1.5$, $\alpha = 0.002$

asynchronous MFA, $k = 1.5$, $\alpha = 0.001$

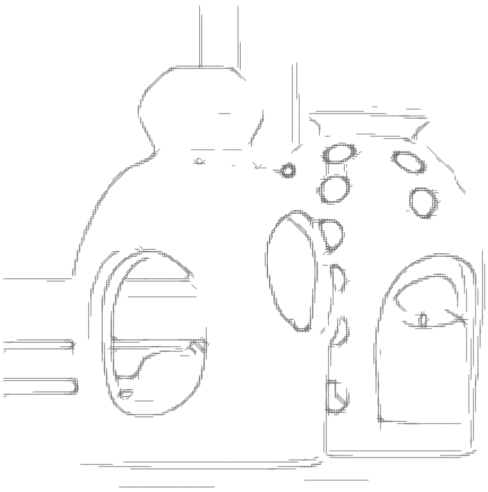
5.2.5. Other real cases

These results have been obtained with the asynchronous Mean Field Annealing by modifying only the parameter α.

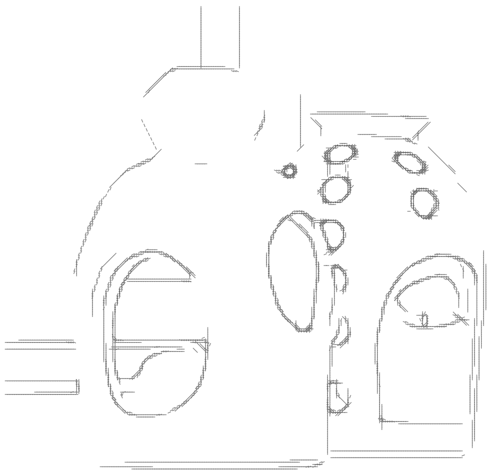
Original image



Boundaries (6727 edges, 1836 edgels)



$\alpha = 0.0005$



$\alpha = 0.0010$

The following result has been obtained with a threshold of edgels composed by at least two edges (the original image counts 4953 edgels):

Original image

Boundaries (12628 edges, 2895 edgels)

$\alpha = 0.0001$

$\alpha = 0.0010$

6. User manual

6.1. Loglin: functional implementation of an edge detector

Loglin is the program performing edge detection. If the program is executed without parameters, or with wrong parameters, the following message is provided:

```
Linear Logic Edge Detector
Usage: loglin.exe [options] file
  file           (pgm/pbm/ppm) file containing ascii or binary image

Options:
  -help          display this help message
  -ndirs         number of directions (only even values) (default = 16)
```

```
  -t              threshold value (default = 0.01)
```

Indeed, the program needs at least one parameter, i.e., the name of the input image. The image formats which are supported are *pgm, pbm, ppm.*

Before applying the edge detection to *ppm* images, we first convert them in grey scale images using the following formula (standard ITU YCbCr):

$$grey = \begin{pmatrix} 0.2125 & 0.7154 & 0.0721 \end{pmatrix} \begin{pmatrix} red \\ green \\ blue \end{pmatrix}$$

The program *loglin* can be also executed with optional parameters:

- the parameter *-help* shows an help message for the user,
- the parameter *-ndirs* specifies the number of directions in which the angle of edges is divided,
- the parameter *-t* followed by a floating point is used as threshold.

6.2. Camleye: a functional implementation of a figure-ground discriminator

The usage of *camleye*, which is the figure ground discriminator, is more complex. If the program is executed without parameters, or with wrong parameters, the following message is provided:

```
Caml's eye
Usage: camleye.exe [options] file
  file            ps file containing edges

Options:
  -help           display this help message
  -k              k value (default = 1.5)
  -alpha          alpha value (default = 0.0055)
  -n              minimum number of edge in an edgel (default = 1)
  -s              save edge map
  -syn            MFA's synchronous version
  -sa             Simulated Annealing algorithm
```

The program needs the name of the postscript file containing an edge map created with the loglin application. If the file does not exists or it is not a valid postscript an error message is given.

The program *camleye* can be also executed with optional parameters:

- the parameter *-help* shows an help message for the user,
- the parameter *-k*, followed by a floating point, sets the value of the variable k specified in the paper [1],

- the parameter *-alpha*, followed by a floating point, sets the value of the variable α specified in the paper [1]. In the case of Simulated Annealing, this parameter correspond to the variable λ,

- the parameter *-n*, followed by an integer, is used to get rid of all edgels which are composed by less than n edges (this is a form of threshold),

- the parameter *-s* it is used to save the edge map resulting from the threshold operation described above,

- the parameter *-syn* launch the synchronous version of MFA,

- the parameter *-sa* launch the Simulated Annealing algorithm.

A. Appendix: Technical choices

This software has been developed in Ocaml (Objective Caml ver. 3.0.7). To implement the parallel version of synchronous mean field annealing we have used:

- MPI (version 1.2.5): a library which is used for parallel programming in MIMD style.

- OCamlMPI: Caml bindings for a large subset of MPI functions (written by Xavier Leroy).

REFERENCES

1. L. Hrault, R. Horaud, *Figure Ground Discrimination: A Combinatorial Optimization Approach*, IEEE PAMI, Vol 15, No 9, pp. 899-914, September 1993.
2. A. Amir , M. Lindenbaum, *Ground from Figure Discrimination*, Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, p.521, June 23-25, 1998
3. P. Parent and S.W.Zucker. *Trace inference, Curvature Consistency and Curve Detection.* IEEE Transactions on Pattern Analysis and Machine Intelligence, 11(8):823-839, August 1989.
4. Lee A. Iverson, Steven W. Zucker: *Logical/Linear Operators for Image Curves.* IEEE Transactions on Pattern Analysis and Machine Intelligence 17(10): 982-996 (1995)
5. L.A. Iverson, *Logical/Linear Operators for Image Curves implementation.* `ftp://ftp.cim.mcgill.ca/pub/people/leei/loglin.tar.gz`
6. L.A. Iverson, *Toward Discrete Geometric Models for Early Vision*, Ph.D. Dissertation, McGill University, Montreal, 1993.