Published in Image Processing On Line on YYYY–MM–DD.
Submitted on YYYY–MM–DD, accepted on YYYY–MM–DD.
ISSN 2105–1232 © YYYY IPOL & the authors CC–BY–NC–SA
This article is available online with supplementary materials,
software, datasets and online demo at
http://dx.doi.org/10.5201/ipol

Piecewise Affine Image Segmentation Based on Mumford-Shah Functional

Gabriele Facciolo[1]

[1] CMLA, ENS Cachan, France (facciolo@cmla.ens-cachan.fr)

PREPRINT January 5, 2014

Abstract

This document describes and illustrates how to use the IPOL LaTeX class, created to produce a uniform layout for IPOL articles. The restrictions imposed on articles to be published in IPOL are briefly discussed.

Keywords: segmentation

1 Introduction

Segmenting an image means finding its homogeneous regions (homogeneous characteristics: color, texture, etc...) and its borders. Hopefully these regions will correspond to real world objects, while the edges will correspond to separation between them. Finding homogeneous regions, which fit a certain model, and the borders of the objects can be considered as two different problems, but one of the most studied mathematical models in image processing addresses both simultaneously. The Mumford-Shah functional (Eq. (1)) resumes elegantly the requisites of a segmentation penalizing the excessive length of borders, reducing the model error and imposing smoothness on the result. In the book by Morel and Solimini [4], this functional is studied in deep mathematical rigorosity. They propose a variational implementation strategy that will be used as guide for the present work.

There are many other approaches to segmentation: in some cases we could be just interested on determining one single region while in other cases we want to obtain a full partition of the image. Some segmentation techniques allow for determining overlapping regions (like the circles depicted in figure 8), while others enforce the fact that the regions are disjoint. But in [4] is shown that the majority of methods, in some way, are minimizing a variational energy, giving more or less importance to different terms of the minimization, i.e. edge length, region regularity.

Most of the recent work is centered on unifying frameworks of different techniques [7] and defining more accurate models for the segmented regions. For instance, [6] uses: 1. Stochastic processes to model the regions, and 2. The level set framework to minimize the functional.

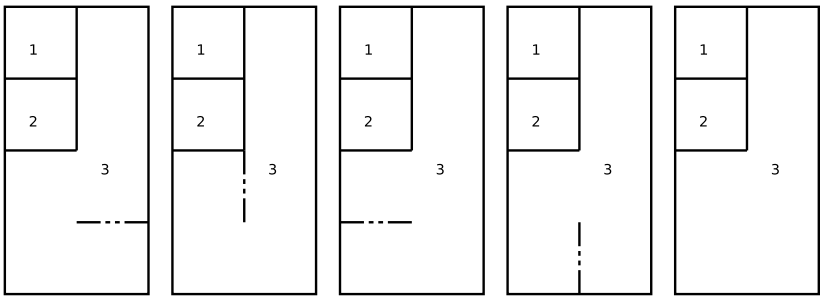

Figure 1: Different edge configurations that correspond to the same partition.

1.1 How big is the segmentation search space?

Before presenting the strategy we will spend a little time discussing the problem. We know that the Mumford-Shah conjecture is not fully proved yet [4], then we cannot assure the uniqueness of a solution. Instead of trying to answer this question, let's just try to figure out the size of the space of all possible segmentations, just to appreciate the complexity of the problem of finding *one* segmentation.

Clearly in a digital image of $n \times n$ pixels there is a finite amount nP of possible partitions, but the exact number of possible partitions is not known. The best we can do is try to bound this quantity:

- On one side we can consider the partition as all possible configurations of boundaries between the pixels. In an image of $n \times n$ pixels there are $2 \times n \times (n-1)$ edges between the pixels. A configuration with all the boundaries set as "active" corresponds to one region per pixel, while and 0 boundaries corresponds to 1 complete region. The total amount is :

  $$nP < 2^{2n(n-1)}$$

  Which for 10×10 pixels image corresponds to: 1.53×10^{54} possible configurations. We can observe that this bound considers a lot of configurations that correspond to the same partition as shown in figure 1.

- Another possibility is to consider all possible ways of labelling the n^2 pixels in k regions $C_k^{n^2}$, then sum for all the possible regions:

  $$nP < \sum_{k=1}^{n^2} C_k^{(n^2)} = 2^{n^2}$$

  This bound is tighter than the previous, for an image of 10×10 pixels leads to: 1.26×10^{30}, but is also considering useless configurations with disconnected regions, because it models the pixels as independent.

Considering the size of the search space we must accept that all the algorithms have to be heuristic or based on weakened restrictions. For instance, the algorithmic approach we are going to use was proposed in [4] and has been demonstrated to find the unique minimum in the subset *2-normal segmentations*. The 2-normal segmentations are "minimal" in the sense that, any further merging of their regions will increase energy function, instead of decreasing.

1.2 Mumford-Shah Image Segmentation

As mentioned before, minimization of the Mumford-Shah functional (Eq. (1)) produces a boundary set B and a piecewise smooth approximation f of the image u.

$$E(B,f) = \lambda_1 \int_{(x,y)\in\Omega} [f(x,y)-u]^2 dA + \lambda_2 \int_{\Omega\setminus B} |\nabla f|^2 dA + \lambda_3 \int_B ds \tag{1}$$

The first term minimizes the error between $f(x,y)$ and u, the second term imposes smoothness of the approximation f over all image Ω, except on the boundaries B, and the third term minimizes the length of the set B of all the boundaries. We will refer to from Morel and Solimini's [4] work for the variational framework, where a simplified segmentation model without smoothing term is considered (Eq. (2)). Additionally, we will impose the piecewise affine model $f_a(x,y) = ax + by + c$.

$$E(B,f) = \int_{(x,y)\in\Omega} [f_a(x,y)-u]^2 dA + \lambda \int_B ds \tag{2}$$

If the value of λ is too small, the boundaries have no penalization an the resulting the segmentation is trivial (one region per pixel). With a higher value of λ, a couple of regions O_1 and O_2 (with common boundary denoted as $\partial(O_1,O_2)$ and its length as $\ell(\partial(O_1,O_2))$) will be passible of merging if: $E(B\setminus\partial(O_1,O_2), f') < E(B,f)$, replacing in Eq. (2), and observing that the modification in f' are local and that the difference in boundary length is just the length of the common boundary, we obtain:

$$\int_{O_1\cup O_2} (f_a-u)^2 dA - \lambda \int_{\partial(O_1,O_2)} ds < \int_{O_1} (f_a-u)^2 dA + \int_{O_2} (f_a-u)^2 dA$$

$$err(O_1\cup O_2) - \lambda\ell(\partial(O_1,O_2)) < err(O_1) + err(O_2)$$

$$\lambda > \frac{err(O_1\cup O_2) - err(O_1) - err(O_2)}{\ell(\partial(O_1,O_2))} \tag{3}$$

Finally, all the boundaries that met the *merging criterion* of equation (3) are removed until no more boundaries can be removed. The resulting segmentation will be *2-normal* (according to the previous definition). In [4] has been proven that this strategy leads to a unique segmentation since the λ value acts as a scale parameter: increasing its value increases the tension of the borders, reducing the boundaries that met the criterion and producing a coarser segmentation.

2 Description of the algorithm

In [3] it is proposed that any region merging algorithm can be defined by establishing tree concepts: Region Model, Merging Criterion and Merging Order.

It is possible to describe the algorithm that minimizes the simplified Mumfors-Shah functional, described in the previous section, in terms of these concepts: The *Region Model* is the piecewise affine model and affects the values of the *err* function (Eq. (3)). The *Merging Criterion* is given by the equation (3): all the boundaries that met the criterion are removed. And the *Merging Order* is given by the usage of the scale parameter λ which allows for merging small regions first, and then increase their size.

In other words, the method will choose an initial (small) λ and will merge all the regions that have edges that don't met the criterion. Then the λ is increased and the previous process is repeated until reaching some stop condition.

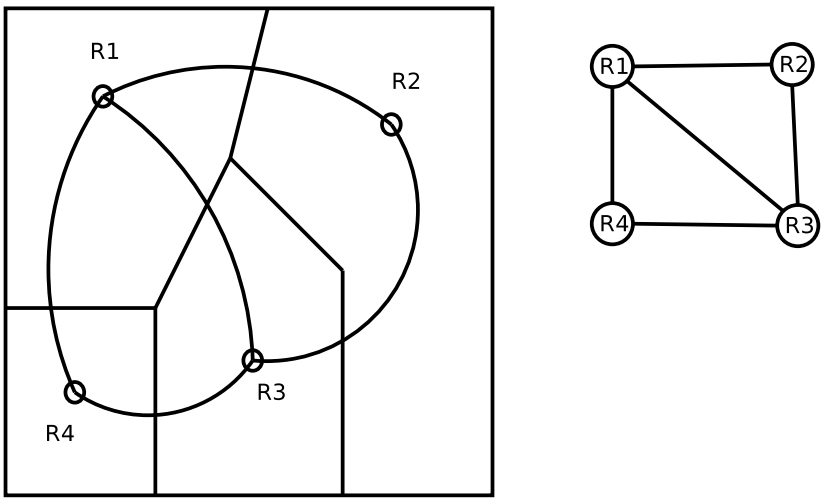

Figure 2: Region-Edge duality and Region Adjacency Graph.

This definition of the algorithm finds a partition from a bottom-up perspective, which corresponds to merging smaller regions into bigger ones. But there are methods that use a top-down region splitting. In any case, when a decision of merging or splitting is made, this decision is never revisited, hence the algorithm is greedy.

As we are going to use a region merging strategy, the data structure should allow to perform merging operations between the regions easily. This structure is the *Region Adjacency Graph*.

In this graph, nodes correspond to regions and edges between nodes represent the neighborhood relationship. Two nodes (regions) would be connected by an edge if they are neighbors. The merging algorithm is then a procedure to remove edges from the graph and merge the corresponding nodes. Usually the process of removing edges is performed in an iterative fashion. At each step, only two neighboring regions are evaluated and eventually merged. In other cases it is possible to consider more than two regions for each step [1].

2.1 Affine region model

Our segmentation algorithm uses an affine region mode. This means that the pixels contained in a region are modeled by a linear equation: $f_i(x, y) = ax + by + c$.

For every region of the resulting segmentation, it will provide three parameters of the model. There are many ways to fit information to a model by estimating its parameters. Probably the most known technique is *least squares fitting* [5].

Provided the data $u(x, y)$ of inside the region T to fit to a model $ax + by + c$, the least squares method minimizes the quadratic error between the model and the points:

$$\min_{a,b,c} err(T) = \min_{a,b,c} \sum_{(x_i,y_i)\in T} \left[(ax_i + by_i + c) - u(x_i, y_i)\right]^2$$

This minimization problem can be seen as an over-determined system of linear equations, then it is possible to rewrite it in matrix form:

$$\min_x (Ax - d)^2 \tag{4}$$

where:

$$A = \begin{pmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \\ \vdots & \vdots & \vdots \\ x_i & y_i & 1 \end{pmatrix} \quad d = \begin{pmatrix} u(x_1, y_1) \\ u(x_2, y_2) \\ u(x_3, y_3) \\ \vdots \\ u(x_i, y_i) \end{pmatrix} \quad x = \begin{pmatrix} a \\ b \\ c \end{pmatrix}$$

The minimum value of equation (4) is reached when: $A^t(Ax - d) = 0$, which leads to the solution:

$$x = (A^t A)^{-1} A^t d$$

The computations needed to obtain the value of the parameters x are linearly related to the amount of the data points. But observing the solution we can use some interesting properties that allow to improve computation of the parameters.

$$M_T = A^t A = \begin{pmatrix} \sum x_i^2 & \sum x_i y_i & \sum x_i \\ \sum x_i y_i & \sum y_i^2 & \sum y_i \\ \sum x_i & \sum y_i & \sum 1 \end{pmatrix} \quad N_T = A^t d = \begin{pmatrix} \sum x_i d(x_i, y_i) \\ \sum y_i d(x_i, y_i) \\ \sum d(x_i, y_i) \end{pmatrix}$$

We can observe that all the elements of the matrixes are accumulators of some kind, only related to the data. Additionally, the inversion of a matrix and the product of two fixed size matrixes takes a constant time.

Applying this to our case, these observations allow to speed-up the computation of the parameters for the merged region to a constant time. Simply by: 1. Reusing (and adding) the accumulators from the component regions to obtain the accumulators corresponding to the merged regions ($M_{T_i \cup T_j} = M_{T_i} + M_{T_j}$ and $N_{T_i \cup T_j} = N_{T_i} + N_{T_j}$), 2. A constant job to compute the inverse of $A^t A$, and 3. The parameters for the new region without traversing all the pixels of the merged region.

This observation leads to a speed improvement on determining the model during the merging procedure and reducing the cost from linear (in the amount of pixels) to constant. But we also need the residual errors of the model. For the moment, the only way to obtain these values is by traversing all the pixels and computing the error function:

$$err(T) = \sum_{(x_i, y_i) \in T} [(ax_i + by_i + c) - u(x_i, y_i)]^2 \tag{5}$$

2.2 Data structures

To model the Region Adjacency Graph we can choose between different data structures: 1. The *adjacency matrix*, which is too expensive ($(200 \times 200)^2$ slots for representing the edges of a squared 200 pixel image), 2. The *edge list representation*, is quite inefficient for removing the region (because it requires searching for the specific edge in a single list), or 3. The *region adjacency list* stores a list of edges for each region which can be easily concatenated on a merging event.

Figure 3 displays a sketch of the graph structure using an adjacency list. Notice that each edge is shared among two lists and that it contains references to the nodes. This facilitates the removal of the edges and merging of the neighboring regions.

Another fundamental data structure will be the priority queue for storing the edges: all the edges are candidates to removal but is only removed the one that have least λ value in each step. The sorting of the queue is just partial as we only need the smallest element. This allows to obtain a

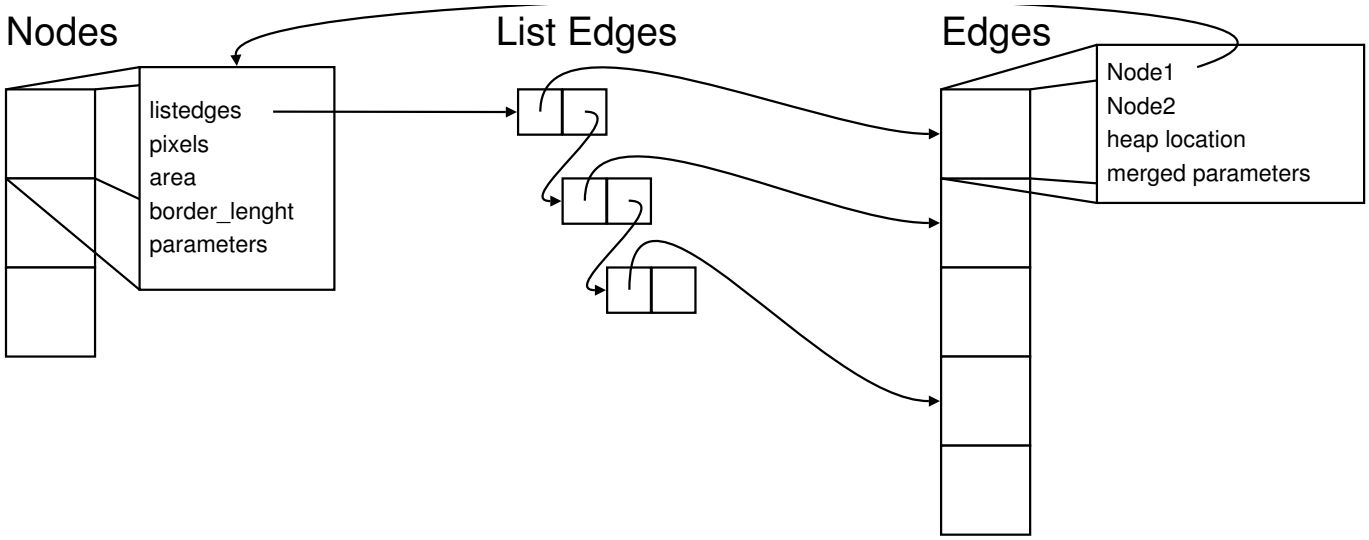

Figure 3: Representation of the graph as adjacency list (List Edges is a chained list). Each Node has a reference to the list of references of its edges and each Edge has a reference to the two connected Nodes.

faster implementation that will determine the final speed of the algorithm, because (as we'll see later) most of the time spent by the algorithm will be extracting the maximum value from the priority queue and updating the value of the remaining nodes in it.

The priority queue is implemented with a *binary heap*, where the most common operations are *find_max* with $O(\lg(n))$, and updating a value on the heap which is also $O(\lg(n))$.

All the data structures were specifically built for the usage in the region merging algorithm. This decision is due to the fact that edges of the graph structure are linked with the heap elements and the tested library (*boost.org*) did not allow to perform this kind of linking.

Further characteristics of data structures will be described next:

- **pixels**: All pixels of the image are represented by a structure containing the position of the pixel in the image (x-pos, y-pos), its gray-level information and a reference to the next pixel. The most common operations over the pixels are:

  1. **full traversal of the list**: $O(n)$ (for the computation of the model).
  2. **the merging operation**: Merging of two pixel lists (for the merging of the regions).

  In order to accelerate to $O(1)$ a *reference to the last element* of the list, is quite useful.

- **regions**: The regions will reference the first pixel of a null terminated chained list, which will contain all the pixels of the region. At the beginning of the execution, every pixel will be considered as a region.

- **nodes**: The nodes of the graph are the regions of the image to be segmented. Each node has a reference to a list of edges. The nodes also contain all the information referent to the region, for instance: a chained list of all its pixels, a boundary length counter, an area counter and the `representation parameters`.

  The most common operation over the nodes is merging. A merging triggers multiple operations: Fusion of the edge lists of the two regions, update of the reference to the actual region for each

neighboring region, computation of a new **fitness** for each fused edge, and update of the heap of edges.

Another common operation over the nodes is the traversal of all the nodes. This is possible because the nodes are stored in a vector.

- **edge**: An edge contains references to the nodes it connects and the `representation parameters` of the merged region in case of removing it. Also, for simplicity, it contains a reference to its location in the priority queue of edges as most of the time the edge weight is being updated.

2.3 Pseudo-code of the Region Merging Algorithm

The following pseudo-code describes with more detail the steps of the algorithm:

1. Initialize the graph structure: one region per pixel and one edge for each boundary.
2. For each edge e of the structure compute the maximal λ value of eq. (1): for which the neighboring regions will be merged, store the λ and the edge in a priority queue.
3. Extract the maximum value ($max\lambda$) of the λs from the priority queue
4. For each edge e that has $\lambda \leq max\lambda$ do:
   (a) Remove e from the priority queue
   (b) Merge the nodes connected by to the edge e: merge the pixel lists, update the area, the length of the boundaries, determine and remove the edges that are redundant after the merging.
   (c) Remove all the merged edges from the priority queue
   (d) For all the edges of the new region: recompute merging value λ with its neighbors,and update the priority queue.
5. If the stop condition (number of nodes or $max\lambda$ value) is not met return to step 3.

2.4 Time Complexity Analysis

In order to study time complexity of the current implementation, we are going to deduce a recurrence relation. It is possible to consider the algorithm as recursive since, in every iteration, it performs the same task only decreasing by one the number of regions.

The same average case analysis will be useful for the piecewise affine segmentation and for the constant affine (which is simpler), but we'll find that for the first (affine) the average case analysis is not sufficient to describe the algorithm.

Let's first define some variables that will make the recurrence relation more readable[1]:

- N: is the total amount of regions currently represented in the graph.
- N_0: the initial amount of regions. This constant number corresponds to the area of the image measured in pixels .

[1]Notice that all the variables are defined relative to N.

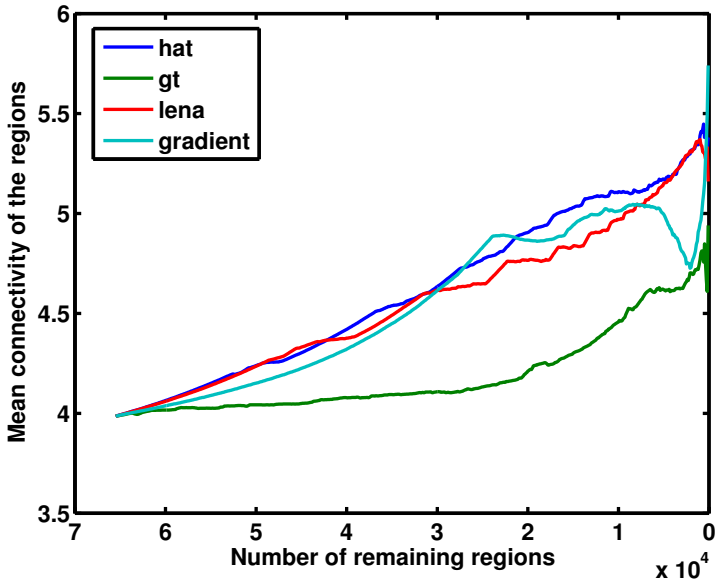

Figure 4: Mean number of neighbors per region, during the execution of the region merging algorithm with different test images. Notice that the mean value never exceeds the 6 neighbors.

- E: the total amount of edges between regions. Notice that the initial number of edges in an image with $m \times n$ pixels is $2m(n-1)$ but for simplicity we take $E \sim 2N$.
- V: is the mean number of neighbors for a region. This value can be computed by taking $V = 2 * E/N$, but experimental verification (as can be seen in Figure 4) has shown that it's $V \sim 5$.
- A: the mean area of a region (measured in pixels), we can compute it by taking $A = N_0/N$, but we'll see that not always the algorithm behaves like expected.

If during the loop only two regions are merged at a time, the recurrence relation of the region merging loop is simply:

$$T[N] = T[N-1] + Merge\_Two\_Regions[N]$$

and developing it leads to:

$$T[N] = \sum_{n=1}^{N} Merge\_Two\_Regions[n]$$

To determine the exact cost of $Merge\_Two\_Regions[n]$ let's study the complexity of each operation involved in the the region merging, resumed for simplicity in the table 1.

2.4.1 Constant piecewise segmentation case

For the constant piecewise segmentation case, the model for a merged region, as well as the error, can be computed in constant time. This is accomplished by considering the area of the merged regions since the means can be updated easily [4] (steps 5.1 & 5.2 of table 1). Resulting in the following cost:

$$\begin{aligned} Merge\_Two\_Regions[n] &= 11 + 2\lg(2n) + 10\,[3 + \lg(2n)] \\ &= 41 + 12\lg(2n) \end{aligned}$$

Step	Algorithm step description	Complexity
1	Obtain the max of the $e = edges\_heap$	const.
2	Remove the top of $edges\_heap$	$O(\lg E)$
3	Merge the regions (of edge e) $R_i = R_i \cup R_j$	
3.1	- concatenate the adjacency lists	$O(V) \sim 5$
3.2	- remove duplicated edges from the list	$O(V) \sim 5$
4	Delete the duplicated edges in R_i from $egdes\_heap$	$O(\lg E)$
5	Update the merging cost $\forall$ edge e of R_i	$\times 2V \sim \times 10$
5.1	- compute the merged model if removing e	$O(2A)$ or const.
5.2	- compute the $err = model - data$ if removing e	$O(2A)$ or const.
5.3	- compute Mumford-Shah functional	const.
5.4	- update location of e in $edges\_heap$	$O(\lg E)$

Table 1: Time complexity detail corresponding to the stages of the region merging inner loop.

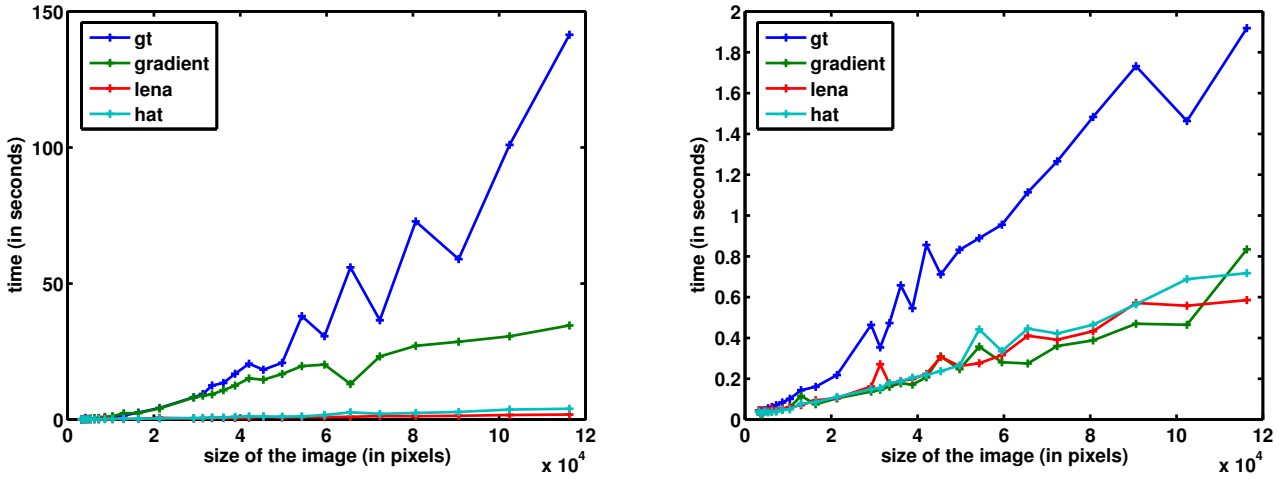

Figure 5: Time simulations with the affine piecewise (left) and constant piecewise (right) implementations, using the images: 'gt', 'hat', 'lena' and 'gradient'. The images have been resized (horizontal axis) to in order to test the time spent by the algorithm.

Then replacing it into the recurrence we obtain:

$$\begin{aligned} T[N_0] &= \sum_{n=1}^{N_0} (41 + 12\lg(2n)) \\ &= 41N_0 + 12 \underbrace{\sum_{n=1}^{N_0} \lg(2n)}_{<N_0 \lg(2N_0)} \end{aligned}$$

The last sum governs the time complexity of this algorithm being $O(n \lg(2n))$ in the average case, which is coherent with the results of the simulations shown in figure 5.

2.4.2 Piecewise affine segmentation case

In the case of piecewise affine regions, computation of the merged model[2], as well as the computation of the error, requires to visit all the pixels $O(2A) = O(2N_0/N)$ increasing the merging time consumption: $Merge\_Two\_Regions[n] = 21 + 12\lg(2n) + 40N_0/n$, then the total time becomes:

$$T[N_0] \;=\; 21N_0 + 12\underbrace{\sum_{n=1}^{N_0}\lg(2n)}_{<N_0\lg(2N_0)} + 40\underbrace{\sum_{n=1}^{N_0} N_0/n}_{N_0\sum_{n=1}^{N_0}\frac{1}{n}}$$

Clearly, the total time complexity of the algorithm will be governed by the third term which is formed by a partial sum of an *harmonic series*. This series can be bounded by $H_n = \sum_{k=1}^{n}\frac{1}{k} < \lg(n) + \frac{1}{2n} + \gamma$ where $\gamma \cong 0.577$ is the Euler-Mascheroni constant[2]. Therefore, average time complexity results in $O\left(n\lg(2n)\right)$.

From the previous analysis, it seems that both algorithms (the piecewise constant and the piecewise affine) have the same order of complexity. But observing the results of the affine segmentation (figure 5), this algorithm is taking considerably more time for some images. This observation led us to study the worst case.

The average time of the constant piecewise algorithm also coincides with the worst case, but with the affine piecewise we've made the assumption that all the regions have "mostly" the same amount of pixel size at every stage of the execution. This fact is reflected in the definition $A = N_0/N$. For the worst case, either all the regions have one pixel or there is only one region that is grown one pixel at a time. For this worst case, computation of the error and the model (steps 5.1 & 5.2 in the table 1) will take much more time: $N_0 - N$ evaluation when there are N regions. This change will affect the total time complexity in the following manner:

$$T[N_0] \;=\; 21N_0 + 12\underbrace{\sum_{n=1}^{N_0}\lg(2n)}_{<N_0\lg(2N_0)} + 40\underbrace{\sum_{n=1}^{N_0} N_0 - n}_{\frac{N_0(N_0+1)}{2}}$$

The resulting order of $O(n^2)$ explains better the results of the simulations (figure 5) with the images 'gt' and 'gradient'. The kind of images in which the worst case is more likely to appear, will be further discussed in the following section.

3 Results and Conclusions

The results of the simulations have been separated in two cases: Natural and synthetic images.

3.1 Synthetic images

The objective of the algorithm was to obtain better segmentations of images that have gradients. In particular, the elevation models like the one shown in figure 7. For this case, it is clear that roofs of buildings are better segmented by the affine algorithm because it "sees" the discontinuities in the

[2]We've seen in section 2.1 that the computation of the least squares error can be reduced to a constant time, but we consider it as linear to be more generic (we just loose a constant in this case).

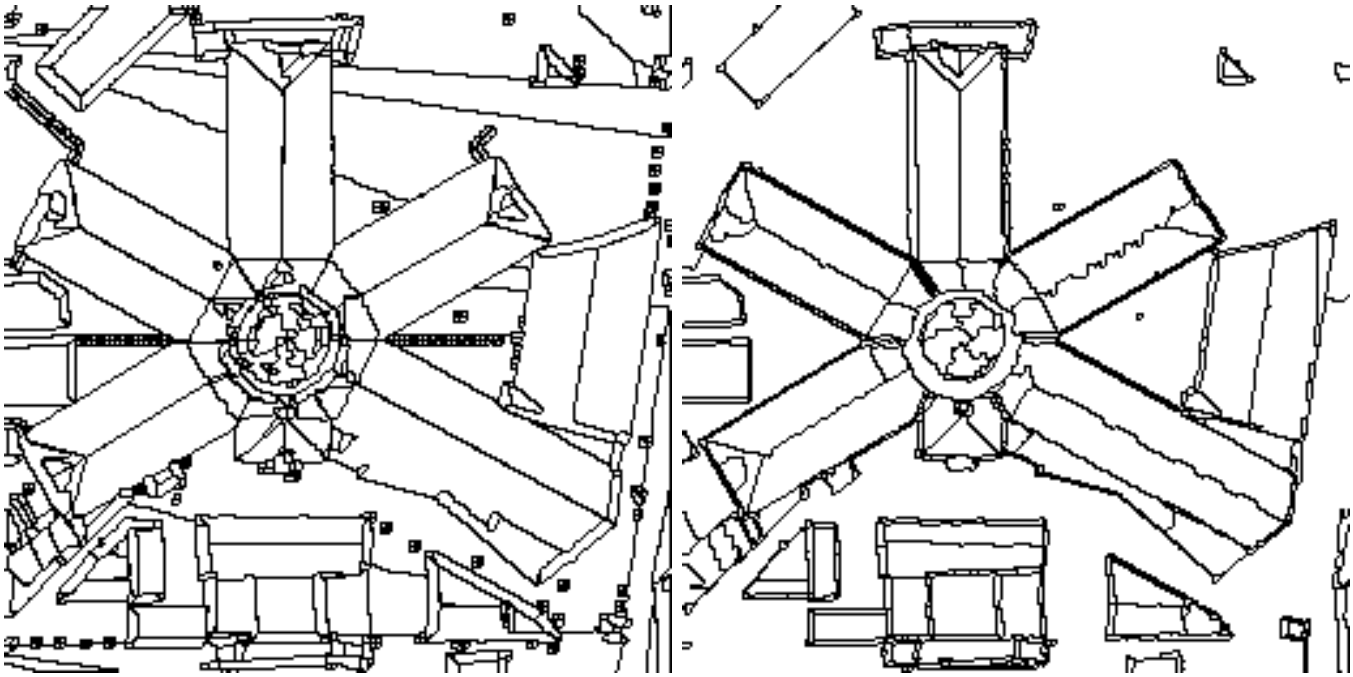

Figure 6: (left) Multichannel piecewise constant segmentation using the channels of the horizontal and vertical gradients of the image. (right) Piecewise affine segmentation. The result that uses gradients has straighter borders, but in some places there are false borders product of the filter used to compute the derivative.

gradient. Nonetheless, the boundaries of some regions are not so regular. One possible improvement is to impose some extra regularity constraint (besides length) over the boundaries, i.e. curvature. Another successful example is presented in figure 8, where the constant segmentation is incapable of determining 4 regions, while the affine finds exactly what we see.

As the affine regions have constant gradients (vertical and horizontal) it is possible to segment the image of derivatives with a constant piecewise algorithm (which is faster). The result of this experiment is compared in figure 6 with the piecewise affine version. Notice that boundaries found by the constant algorithm are more regular, possibly due to the filtering introduced by gradients computation. Also, due to the filtering is the dilatation of the structures and the appearance of some false borders around some buildings (in upper left corner).

3.2 Natural images

In reality, images with constant gradients are not very common in nature and when they appear, they are not linear gradients like the ones we are finding. Anyway we analyzed two real images to test performance. Results with real images (figure 9 and 10) do not have perfect segmentation of the objects because gradients are not uniform. Nevertheless, reconstructions are interesting as they have better visual impact than the constant piecewise reconstruction.

In terms of times, as mentioned before, some images may have a bigger time complexity using the affine piecewise algorithm. In general, natural images are best featured to reach the average case performance because of the many little details or textures common in the natural images. During the execution of the merging algorithm these structures are merged in small groups, then the groups are merged with each other and so on. Then no region grows out of control, escaping the worst case.

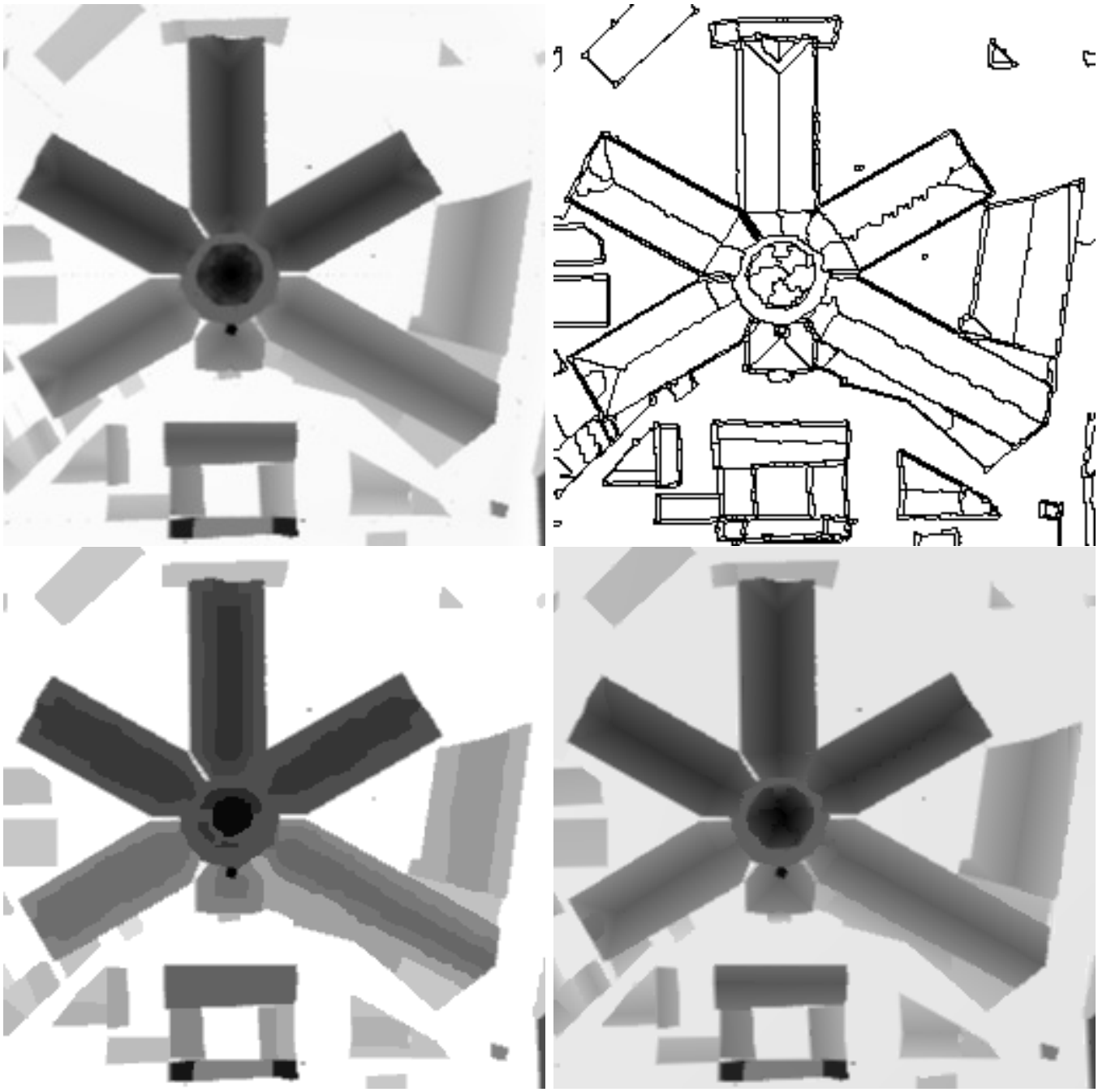

Figure 7: Segmentation of a syntectic image 'gt' in 1000 regions. Comparison between the original image (upper left), borders of the affine segmentation(upper right), result of the constant segmentation (lower left) and reconstruction of the affine segmentation (lower right).

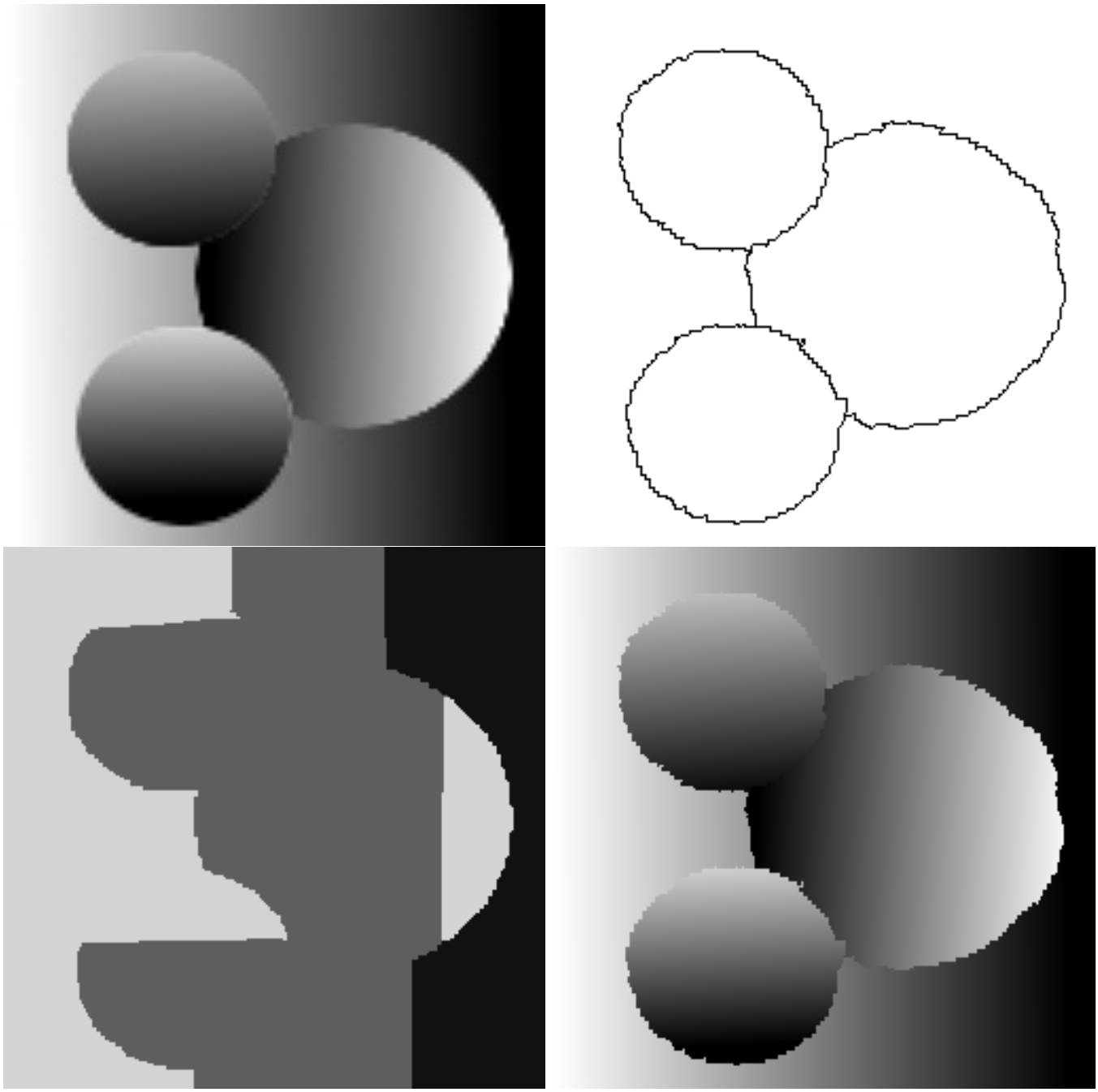

Figure 8: Segmentation of a syntectic image 'gradient' in 4 regions. Comparison between the original image (upper left), borders of the affine segmentation(upper right), result of the constant segmentation (lower left) and reconstruction of the affine segmentation (lower right).

On the other hand, synthetic images have uniform regions (remember that uniform in the affine case includes uniform gradients) that are grown since the first iteration of the algorithm and many small unassigned pixels fall in the worst case.

3.3 Conclusions

We can conclude that the resulting segmentation algorithm meets the objective of segmenting correctly affine regions (roofs) present in digital elevation models. Additionally, the resulting implementation of the algorithm can now be used as an extensible framework to experiment with other Mumford-Shah functionals using different models.

Image Credits

© IPOL (there's no need to credit this image, here is used as an example.)

References

[1] L. Demaret, N. Dyn, M. Floater, and A. Iske. Adaptive thinning for terrain modelling and image compression, 2004.

[2] Eric W. Weisstein et al. Harmonic number. from mathworld - a wolfram web resource.

[3] L. Garrido and P. Salembier. Region based analysis of video sequences with a general merging algorithm. In IX European Signal Processing Conference, EUSIPCO'98, 1998.

[4] J.M. Morel and S. Solimini. *Variational Methods in Image Segmentation*. Birkhauser, 1995.

[5] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical Recipes in C*. Cambridge University Press, second edition, 1995.

[6] A. Tsai, Yezzi, A, Jr, and A. S. Willsky. Curve evolution implementation of the Mumford-Shah functional for image segmentation, denoising, interpolation, and magnification. *IEEE Tr. Im. Proc.*, 10(8):1169–1186, August 2001.

[7] Song Chun Zhu, Tai Sing Lee, and Alan L. Yuille. Region competition: Unifying snakes, region growing, energy/bayes/MDL for multi-band image segmentation. In *ICCV*, pages 416–, 1995.

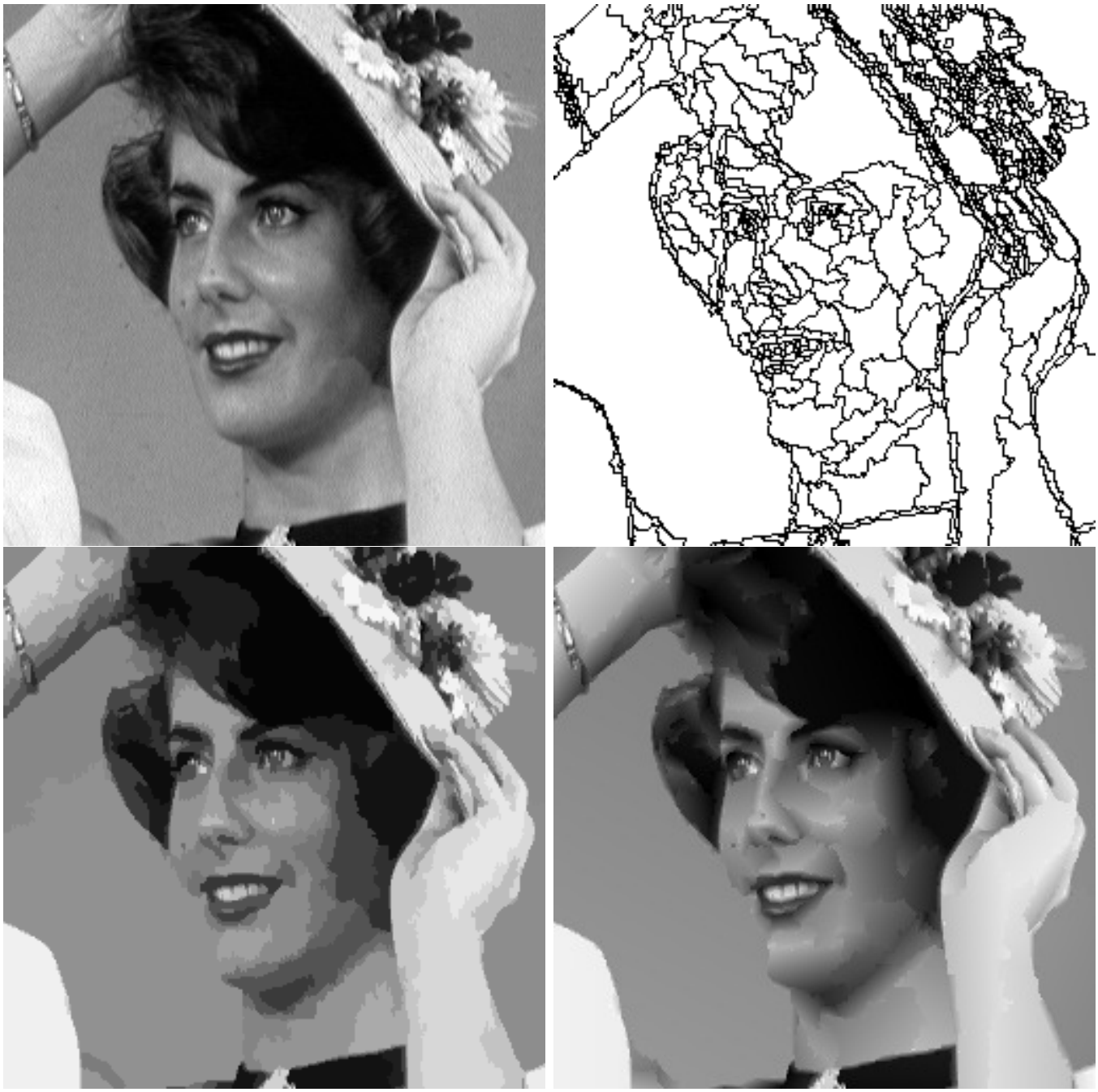

Figure 9: Segmentation of the natural image 'hat' in 1000 regions. Comparison between the original image (upper left), borders of the affine segmentation(upper right), result of the constant segmentation (lower left) and reconstruction of the affine segmentation (lower right).

Figure 10: Segmentation of the natural image 'lena' in 1000 regions. Comparison between the original image (upper left), borders of the affine segmentation(upper right), result of the constant segmentation (lower left) and reconstruction of the affine segmentation (lower right).