Multiple Contour Finding and Perceptual Grouping using Minimal Paths

Laurent D. COHEN
CEREMADE, UMR 7534, Université Paris-Dauphine
75775 Paris cedex 16, France; Email: cohen@ceremade.dauphine.fr

Abstract

We present a new approach for finding a set of contour curves in an image. We consider the problem of perceptual grouping and contour completion, where the data is a set of points in the image. A new method to find complete curves from a set of contours or edge points is presented. Our approach is based on a previous work on finding contours as minimal paths between two end points using the fast marching algorithm [1]. Given a set of key points, we find the pairs of points that have to be linked. The paths that join them complete the initial set of contours and allow to close them. In a second part, we propose a scheme that does not need key points for initialization. Key points are automatically selected from a larger set of admissible points. We illustrate the capability of our approach to close contours with synthetic examples.

1 Introduction

We are interested in perceptual grouping and finding a set of curves in an image with the use of energy minimizing curves. Since their introduction, active contours [2] have been extensively used to find the contour of an object in an image through the minimization of an energy. In order to get a set of contours of different objects, we need many active contours to be initialized on the image. The level sets paradigm [3, 4] allowed changes in topology. It enables to get multiple contours by starting with a single one. However, these do not give satisfying results when there are gaps in the data since the contour may propagate into a hole and then split to many curves where only one contour is desired. This is the problem encountered with perceptual grouping where a set of incomplete contours is given. For example, in a binary image like the ones in Fig. 1 with a drawing of a shape with holes and spurious edge points, human vision can easily fill in the missing boundaries, remove the spurious ones and form complete curves. Perceptual grouping is an old problem in computer vision. It has been approached more recently with energy methods [5, 6, 7]. These methods find a criteria for saliency of a curve

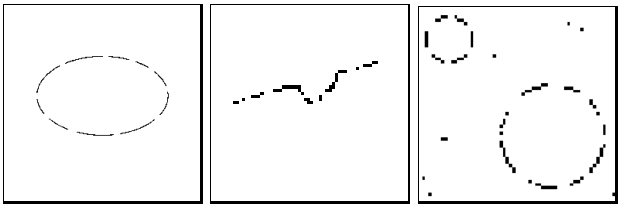

Figure 1: Examples of incomplete contours

component or for each point of the image. In these methods, the definition of saliency measure is based indirectly on a second order regularization snake-like energy ([2]) of a path containing the point. However, the final curves are obtained generally in a second step as ridge lines of the saliency criteria after thresholding. In [8] a similarity between snakes and stochastic completion field is reported. Motivated by this relation between energy minimizing curves like snakes and completion contours, we are interested in finding a set of completion contours on an image as a set of energy minimizing curves.

In order to solve global minimization for snakes, the authors of [1] used the minimal paths, as introduced in [9, 10]. The goal was to avoid local minima without demanding too much on user initialization, which is a main drawback of classic snakes [11]. Only two end points were needed. The numerical method has the advantage of being consistent (see [1]) and efficient using the Fast Marching algorithm introduced in [12]. In this paper we propose a way to use this minimal path approach to find a set of curves drawn between points in the image. As a first step, a set of end points is assumed to be given. We also introduce a technique that automatically finds the end points. This can be also viewed as an extension of the minimal path approach by finding automatically, based on construction of a minimal energy global map, a set of *key* end points. In order to find a set of most salient contour curves in the image, we draw the minimal paths between pairs of *linked neighbors* selected among the *key* end points.

In our examples, the potential P to be minimized

along the curves is usually an image of edge points that represent simple incomplete shapes. These edge points are represented as a binary image with small potential values along the edges and high values at the background. Such a potential can be obtained from real images by edge detection (see [13]). The potential could also be defined as edges weighted by the value of the gradient or as a function of an estimate of the gradient of the image itself, $P = g(\|\nabla I\|)$, like in classic snakes. In these cases the chosen function has to be such that the potential is positive everywhere, and it has to be decreasing in order to have edge points as minima of the potential. The potential could also be a grey level image as in [1].

The problems we solve in this paper are presented as follows:

- Minimal path between two points: The solution proposed in [1] is reviewed in Section 2.
- Minimal paths between a given set of pairs of points is a simple application of the previous one.
- Minimal paths for a given set of unstructured points: we propose a way to find pairs of linked neighbors and paths between them (Section 3).
- Minimal paths between an unknown set of point: Our main contribution concerns the automatic finding of key points and the drawing of minimal paths that leads to completed curves (Section 4).

A detailed version of this work is presented in [14].

2 Minimal Paths. Weighted distance

2.1 Global minimum for Active Contours

We present in this section the basic ideas of the method introduced in [1] to find the global minimum of the active contour energy using minimal paths. The energy to minimize is similar to classical deformable models (see [2]) where it combines smoothing terms and image features attraction term (Potential P):

$$E(C)=\int_\Omega \Big\{ w_1 \|C'(s)\|^2 + w_2 \|C''(s)\|^2 + P(C(s)) \Big\} ds \quad (1)$$

where $C(s)$ represents a curve drawn on a 2D image and Ω is its domain of definition. The authors of [1] have related this problem with the recently introduced paradigm of the level-set formulation. In particular, its Euler equation is equivalent to the geodesic active contours [4]. The method introduced in [1] improves energy minimization because the problem is transformed in a way to find the global minimum.

2.2 Problem formulation

As explained in [1], we are lead to minimize

$$E(C) = \int_{\Omega=[0,L]} \{w + P(C(s))\} ds, \quad (2)$$

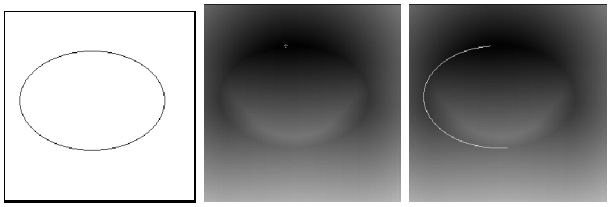

Figure 2: Finding a minimal path between two points. On the left, the potential is minimal on the ellipse. In the middle, the minimal action or weighted distance to the marked point. On the right, minimal path using backpropagation from the second point.

where s is the arclength parameter ($\|C'(s)\| = 1$). The regularization of this model is now achieved by the constant $w > 0$ (see [1, 15] for details). Given a potential $P \geq 0$, the energy is like a distance weighted by $\tilde{P} = P + w$. The minimal action $\mathcal{U}$ is defined as the minimal energy integrated along a path between starting point p_0 and any point p:

$$\mathcal{U}(p) = \inf_{\mathcal{A}_{p_0,p}} E(C) = \inf_{\mathcal{A}_{p_0,p}} \left\{ \int_\Omega \tilde{P}(C(s)) ds \right\} \quad (3)$$

where $\mathcal{A}_{p_0,p}$ is the set of all paths between p_0 and p. The minimal path between p_0 and any point p_1 in the image can be easily deduced from this action map by a simple back-propagation (gradient descent on $\mathcal{U}$) starting from p_1 until p_0 is reached.

2.3 Fast Marching Resolution

In order to compute this map $\mathcal{U}$, a front-propagation equation related to Eqn. (3) is solved: $\frac{\partial C}{\partial t} = \frac{1}{P}\vec{n}$. It evolves a front starting from an infinitesimal circle shape around p_0 until each point inside the image domain is assigned a value for $\mathcal{U}$. The value of $\mathcal{U}(p)$ is the time t at which the front passes over p.

The *Fast Marching* technique, introduced in [12], was used in [1] noticing that the map $\mathcal{U}$ satisfies the Eikonal equation $\|\nabla \mathcal{U}\| = \tilde{P}$ and $\mathcal{U}(p_0) = 0$. Since classic finite difference schemes for this equation are unstable, an up-wind scheme was proposed by [12]:

$$\begin{aligned} (\max\{u - \mathcal{U}_{i-1,j}, u - \mathcal{U}_{i+1,j}, 0\})^2 \quad &+ \\ (\max\{u - \mathcal{U}_{i,j-1}, u - \mathcal{U}_{i,j+1}, 0\})^2 \quad &= \quad \tilde{P}_{i,j}^2. \end{aligned} \quad (4)$$

The improvement made by the *Fast Marching* is to introduce order in the selection of the grid points. This order is based on the fact that information is propagating *outward*, because the action can only grow due to the quadratic Eqn. (4). The main idea is similar to the construction of minimum length paths in a graph between two given nodes introduced in [16] (see discussion in [1]). Complexity of *Fast Marching* on a grid

Algorithm for 2D Fast Marching

- Definitions:
  - *Alive* set: grid points at which the action value $\mathcal{U}$ has been reached and will not be changed;
  - *Trial* set: next grid points (4-connexity neighbors) to be examined. An estimate U of $\mathcal{U}$ has been computed using Eqn. (4) from alive points only (i.e. from $\mathcal{U}$);
  - *Far* set: all other grid points, there is not yet an estimate for U;
- Initialization:
  - *Alive* set: start point $p_0, U(p_0) = \mathcal{U}(p_0) = 0$;
  - *Trial* set: reduced to the four neighbors p of p_0 with initial value $U(p) = \tilde{P}(p)$ ($\mathcal{U}(p) = \infty$);
  - *Far* set: all other grid points, $\mathcal{U} = U = \infty$;
- Loop:
  - Let $p = (i_{min}, j_{min})$ be the *Trial* point with the smallest action U;
  - Move it from the *Trial* to the *Alive* set;
  - For each neighbor (i, j) of (i_{min}, j_{min}):
    - If (i, j) is *Far*, add it to the *Trial* set;
    - If (i, j) is *Trial*, update $U_{i,j}$ with Eqn. (4).

Table 1: *Fast Marching* algorithm

with P nodes is bounded by $O(P \log_2 P)$ for the *Fast Marching* on a grid with P nodes. The algorithm is detailed in Table 1. An example is shown in Fig. 2. Solving Eqn. (4) is detailed in appendix.

3 Finding multiple contours from a set of key points p_k

The method of [1], detailed in the previous section allows to find a minimal path between two endpoints. We are now interested in finding many or all contours in an image. A first step for multiple contours finding in an image is to assume we have a set of points p_k given on the image and then find contours passing through these points. We will discuss later how to define these points, in particular in Section 4. For the moment we assume the points are either given by a preprocessing or by the user. We propose to find the contours as a set of minimal paths that link pairs of points among the p_k's. If we also know which pairs of points have to be linked together, finding the whole set of contours is a trivial application of the previous section. This would be similar to the method in [17] which used a dynamic programming approach to find the paths between successive points given by the user. The problem we are interested in here is also to find out which pairs of points have to be connected by a contour. Since the set of points p_k's is assumed to be given unstructured, we do not know in advance how the points connect. This is the key problem that is solved here using a minimal action map.

3.1 Main ideas of the approach

Our approach is similar to computing the distance map to a set of points and their Voronoi diagram. However, we use here a weighted distance defined through the potential P. This distance is obtained as the minimal action with respect to P with zero value at all points p_k. Instead of computing a minimal action map for each pair of points, as in Section 2, we only need to compute one minimal action map in order to find all paths. At the same time the action map is computed we determine the pairs of points that have to be linked together. This is based on finding meeting points of the propagation fronts. These are *saddle points* of the minimal action $\mathcal{U}$. In Section 2, we said that calculation of the minimal action can be seen as the propagation of a front through $\frac{\partial C}{\partial t} = \frac{1}{P}\vec{n}$. Although the minimal action is computed using fast marching, the level sets of $\mathcal{U}$ give the evolution of the front. During the fast marching algorithm, the boundary of the set of alive points also gives the position of the front. In the previous section, we had only one front evolving from the starting point p_0. Since all points p_k are set with $\mathcal{U}(p_k) = 0$, we now have one front evolving from each of the starting points p_k. In what follows when we talk about front meeting, we mean either the geometric point where the two fronts coming from different p_k's meet, or in the discrete algorithm the first alive point which connects two components from different p_k's (see Figs. 3 and 4).

Our problem is related to the approach presented at the end of [1] in order to find a closed contour. Given only one end point, the second end point was found as a saddle point. This point is where the two fronts propagating both ways meet. Here we use the fact that given two end points p_1 and p_2, the saddle point S where the two fronts starting from each point meet can be used to find the minimal path between p_1 and p_2. Indeed, the minimal path between the two points has to pass by the meeting point S. This point is the point half way (in energy) on the minimal path between p_1 and p_2. Backpropagating from S to p_1 and then from S to p_2 gives the two halves of the path.

3.2 Some definitions

These definitions will be used in what follows.

- For a point p in the image, we note $\mathcal{U}_p$ the minimal action obtained by Fast Marching with potential $\tilde{P}$ and starting point p.
- X being a set of points in the image, $\mathcal{U}_X$ is the minimal action obtained by Fast Marching

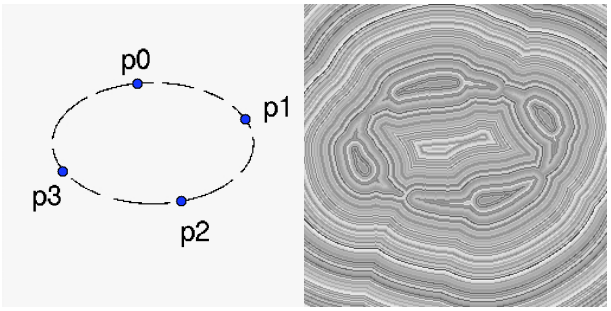

Figure 3: Ellipse example with four points. On the left the incomplete ellipse as potential and four given points; on the right the minimal action map (random LUT to show the level sets) from these points.

with potential $\tilde{P}$ and starting points $\{p, p \in X\}$. This means that all points of X are initialized as alive points with value 0 and all their 4-connexity neighbors are *trial* points. This is easy to see that $\mathcal{U}_X = \min_{p \in X} \mathcal{U}_p$.

- The *region* R_k associated with a point p_k is the set of points p of the image closer in energy to p_k than to other points p_j. This means that minimal action $\mathcal{U}_{p_k} \leq \mathcal{U}_{p_j}, \forall j \neq k$. Thus, if $X = \{p_j, 0 \leq j \leq N\}$, we have $\mathcal{U}_X = \mathcal{U}_{p_k}$ on R_k and the computation of $\mathcal{U}_X$ is the same as the simultaneous computation of each $\mathcal{U}_{p_k}$ on each region R_k. These are the simultaneous fronts starting from each p_k.
- The *region index* r is $r(p) = k, \forall p \in R_k$. (Voronoi Diagram for weighted distance).
- A *saddle point* $S(p_i, p_j)$ between p_i and p_j is the first point where the front starting from p_i to compute $\mathcal{U}_{p_i}$ meets the front starting from p_j to compute $\mathcal{U}_{p_j}$; At this point, $\mathcal{U}_{p_i}$ and $\mathcal{U}_{p_j}$ are equal and this is the smallest value for which they are equal.
- Two points among the p_k's will be called *linked neighbors* if they are selected to be linked together. The way we choose to link two points is to select some *saddle points*. Thus points p_i and p_j are *linked neighbors* if their *saddle point* is among the selected ones.

3.3 Saddle points and Reconstruction of the set of curves

The main goal of our method is to obtain all significant paths joining the given points. However, each point should not be connected to all other points, but only to those that are closer to them in the energy sense. In order to form closed curves, each point p_k should not have more than two *linked neighbors*. The criteria for two points p_i and p_j to be connected is

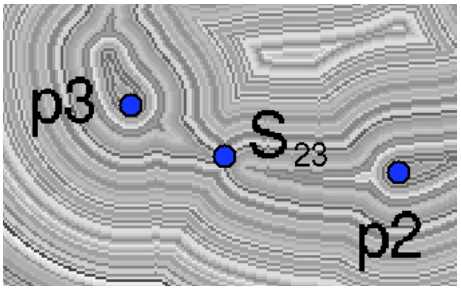

Figure 4: Zoom on a *saddle point*.

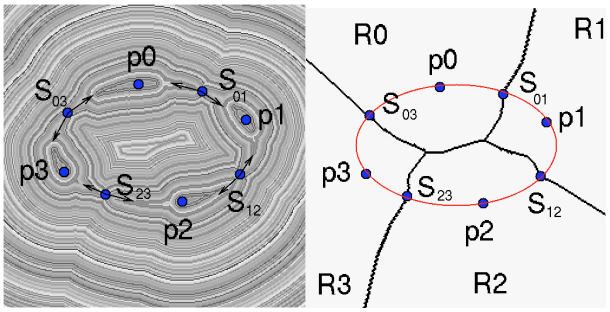

Figure 5: Ellipse example with four points. On the left the *saddle points* are found, and backpropagation is made from them to each of the two points from where the front comes; on the right, the minimal paths and the Voronoi diagram obtained.

that their fronts meet before other fronts. It means that their *saddle point* $S(p_i, p_j)$ has lower action $\mathcal{U}$ than the *saddle points* between these points and other points p_k. The fact that we limit each p_k to have no more than two connections makes it possible that some points will have only one or no connection. This helps removing some isolated spurious points or getting different closed curves not being connected together. We illustrate this in the example of Fig. 7 where one of the p_k is not linked to any other point since all the other points already have two linked neighbors. In case we also need to have T-junctions, the algorithm can be used with a higher number of linked neighbors allowed for each endpoint. A non symmetric relation may also be used to link each point to the closest or the two closest ones, regardless of whether these have already two or more neighbors. In the exemple of Fig. 7, such an approach would link the spurious points with the circles and postprocessing would be needed to remove undesired links.

Once a *saddle point* $S(p_i, p_j)$ is found and selected, backpropagation relatively to final energy $\mathcal{U}$ should be done both ways to p_i and to p_j to find the two halves

of the path between them. We see in Fig. 5 this backpropagation at each of the four *saddle points*. At a saddle point, the gradient is zero, but the direction of descent towards each point are opposite. For each backpropagation, the direction of descent is the one relative to each region. This means that in order to estimate the gradient direction toward p_i, all points in a region different from R_i have their energy put artificially to ∞. This allows finding the good direction for the gradient descent towards p_i. However, as mentioned earlier, these backpropagations have to be done only for selected *saddle points*. In the fast marching algorithm we have a simple way to find *saddle points* and update the *linked neighbors*.

As defined above, the set of *regions* R_k covers the whole image, and forms the Voronoi diagram of the image (see Fig. 5). All *saddle points* are at a boundary between two *regions*. For a point p on the boundary between R_j and R_k, we have $\mathcal{U}_{p_k}(p) = \mathcal{U}_{p_j}(p)$. The *saddle point* $S(p_k, p_j)$ is a point on this boundary with minimal value of $\mathcal{U}_{p_k} = \mathcal{U}_{p_j}$. This gives a rule to find the *saddle points* during the fast marching algorithm.

Each time two fronts coming from p_k and p_j meet for the first time, we define the meeting point as $S(p_k, p_j)$. This means that we need to know for each point of the image from where it comes. This is easy to keep track of its origin by generating an index map updated at each time a point is set as alive in the algorithm. Each point p_k starts with index k. Each time a point is set as alive, it gets the same index as the points it was computed from in formula (4). In that formula, the computation of $U_{i,j}$ depends only on at most two of the four pixels involved. Following notations of appendix, this means the neighbor points A_1 and B_1. These two pixels have to be from the same *region*, except if (i, j) is on the boundary between two regions. If A_1 and B_1 are both alive and with different indexes i and j, this means that regions R_i and R_j meet there. If this happens for the first time, the current point is set as the *saddle point* $S(p_i, p_j)$ between these regions. A point on the boundary between R_i and R_j is given the index of the neighbor point with smaller action A_1. At the boundary between two regions there can be a slight error on indexing. This error of at most one pixel is not important in our context and could be refined if necessary.

3.4 Algorithm

The algorithm for this section is described in Table 2 and illustrated in Figs. 3 and 5. When there is a large number of p_k's, this does not change much the computation time of the minimal action map, but this makes more complex dealing with the list of linked neighbors and *saddle points*. This may generate more conflicting neighbor points, and due to the constraint of having at most two linked neighbors, some gaps may remain between contours. The method can be applied to a whole set of edge points or points obtained through a preprocessing. This was actually our first step in this work. However, choosing few key points simplifies the computation of *saddle points* and *linked neighbors* and the geometry of the paths. When there are few key points, they are not too close to each other. Finding all paths from a given set of points is interesting in the case of a binary potential defined, like in Fig. 3, for perceptual grouping. It can be used as well when a special preprocessing is possible, either on the image itself to extract characteristic points or on the geometry of the initial set of points to choose more relevant points. In what follows we give a way to find automatically a set of key points.

Algorithm with previously defined p_k

- Initialization:
  - p_k's are given
  - $\forall k, V(p_k) = 0; R(p_k) = k$; p_k *alive*.
  - $\forall p \notin \{p_k\}, V(p_k) = \infty; R(p) = -1$; p is *far* except 4-connexity neighbors of p_k's that are *trial* with estimate U using Eqn. 4.
- Loop for computing $V = \mathcal{U}_{\{p_k, 0 \leq k \leq N\}}$:
  - Let $p = (i_{min}, j_{min})$ be the *Trial* point with the smallest action U;
  - Move it from the *Trial* to the *Alive* set with V(p) = U(p);
  - Update $R(p)$ with the same index as point A_1 in formula (5) (see appendix). If $R(A_1) \neq R(B_1)$ and we are in case 1 of appendix where both points are used and if this is the first time regions $R(A_1)$ and $R(B_1)$ meet, $S(p_{R(A_1)}, p_{R(B_1)}) = p$ is set as a *saddle point* between $p_{R(A_1)}$ and $p_{R(B_1)}$. If these points have not yet two *linked neighbors*, they are put as *linked neighbors* and $S(p_{R(A_1)}, p_{R(B_1)}) = p$ is selected,
    For each neighbor (i, j) of (i_{min}, j_{min}):
    - If (i, j) is *Far*, add it to the *Trial* set;
    - If (i, j) is *Trial*, update action $U_{i,j}$.
- Obtain all paths between selected *linked neighbors* by backpropagation each way from their *saddle point* (see Section 3.3).

Table 2: Algorithm of Section 3

4 Finding a set of key points p_k

As motivated in Section 3.4, the problem is now, given a potential, finding automatically a set of points

p_k that can be used as start and end points for the minimal path approach. This way a set of most representative curves would be found in the image. The way endpoints are linked together is similar to the previous section, except we determine the set of endpoints during the minimal action computation. We will see below that the method we propose here has two advantages. First, it avoids computing the energy map to a point when it is not useful. This permits to have much lower computation time for the final energy map ($P \log_2 P$ multiplied by an order less than $\log N$, with N the number of key points). Second, we need to store only one energy map, which means each point has only one value of the energy kept. In order to make "classical" backpropagation between all pairs of points, we would have to store and manage with the whole set of energy maps for all points p_k. We propose below a variation of the algorithm of section 3, which dynamically adds key points and updates the minimal action map. Once the set of key points is found, the final result is the same as in Section 3, but only one computation is needed. We do *not* need a second step running algorithm of Section 3 with the found $p_k's$.

4.1 Algorithm

The main idea is to find iteratively new points on the image and say that two points have to be linked by a minimal path if the fronts starting from these points meet before they meet any other front. As before, in order to get closed curves, we look for two linked neighbors for each point. This means that each key point is linked by a minimal path to at most two key points. In order to find the next key point, we look for point with highest action among a subset of admissible points. This point is the most far in energy from the previously obtained key points. The main algorithm is described in Table 3 and detailed in the next sections.

Algorithm with automatic selection of p_k

- The *admissible points* set $\mathcal{A}$ is defined in section 4.2;
- Initialization:
  - p_0 chosen among *admissible points* (see 4.2)
  - $V_0 = \mathcal{U}_{p_0}$
- Loop: $p_k, V_k, 0 \leq k \leq n$ being known:
  - Let p_{n+1} be the *admissible point* with the highest value of action V_n;
  - Compute $V_{n+1} = \mathcal{U}_{\{p_k, 0 \leq k \leq n+1\}}$. From this definition, computation is made easier since $V_{n+1} = \min(V_n, \mathcal{U}_{p_{n+1}})$. Fast Marching is limited to points where $V_{n+1} \leq V_n$ (see 4.3).
  - Update *saddle points* (see 4.4).
  - Stopping criteria: If $\sup_{\mathcal{A}} V_{n+1} \leq T_{\mathcal{U}}$ or if $n \geq N_{max}$, where $T_{\mathcal{U}}$ and N_{max} are given.
- Select the *saddle points*.
- Obtain all paths between selected *linked neighbors* by backpropagation from their *saddle point* according to the final energy map V_N (see Section 4.4).

Table 3: Main algorithm

4.2 Admissible points

The set $\mathcal{A}$ of *admissible points* should contain all points that are likely to be on the curves we are looking for. These are defined as local minima of the potential P in the general case. For a binary potential defining a set of contour points, as we usually have for perceptual grouping, $\mathcal{A}$ is included in the set of contour points. In order to limit the number of admissible points, we add the condition on a smoothed version of the gradient of the potential to be large enough. This is to impose two kinds of properties. First, if the set of points contains thick curves, this keeps only points that are on the boundary. Second, this removes spurious isolated edge points. In order to start the algorithm, a first admissible point p_0 has to be chosen. This can be done either by the user, or at random, or taking the first of the list. In case we do not want the user to give the initial point, we can use a random point p_0 only in order to define the next point p_1 obtained by the algorithm. And then we start again removing the previous p_0 and replacing it by p_1. This avoids to get a point in the middle of an open curve. This gives preference to points that are at ends of a curve. Another possible interaction with the user could be to give a region of interest in the image, where the admissible points will be constrained to be. Thus the user has only to circle roughly an object in order to get its contours. A priori information on the grey level of the object or the background (for example vessels in medical applications or roads in aerial images) can also be used as a way to define admissible points.

4.3 Fast Marching and partial map computation

For the first point p_0, the fast marching described in section 2.3 is used to compute $V_0 = \mathcal{U}_{p_0}$. For the following points p_k, the same fast marching could be used to obtain $V_{n+1} = \mathcal{U}_{\{p_k, 0 \leq k \leq n+1\}}$ with $p_k, 0 \leq k \leq n+1$ as initial alive points with value 0, as in Section 3. However, it is not necessary to compute the whole map again. In order to estimate V_{n+1}, we need to compute $\mathcal{U}_{p_{n+1}}$ only for those points that have a value smaller than the previously obtained energy map V_n. In the fast marching algorithm, each time a point p has to be put as alive with a value $U(p)$, it is compared to the previous map V_n. If $V_n(p) > U(p)$, the point is put as alive with value $V_{n+1}(p) = U(p) =$

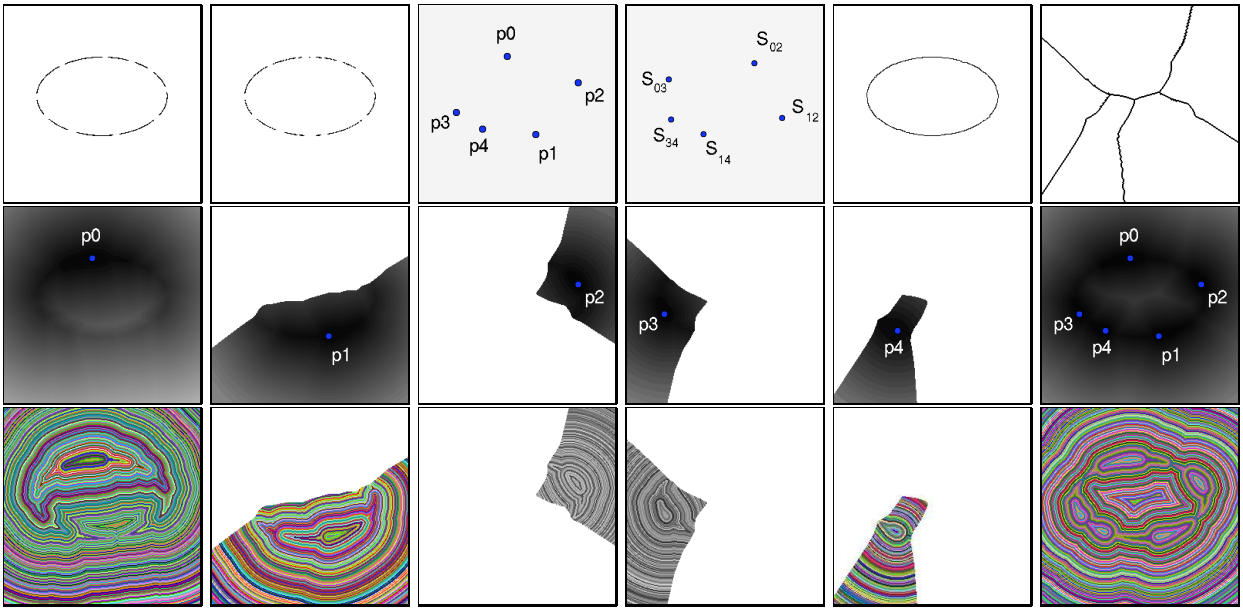

Figure 6: Ellipse example: successive partial map computation for five points. From left to right, line 1: potential, admissible points, found key points, *saddle points*, final paths and voronoi diagram; line 2: successive partial maps for the 5 key points and final map; line 3: the same with random color map to visualize level sets.

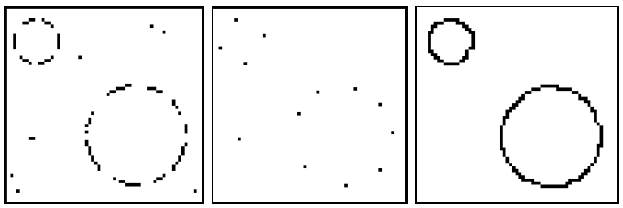

Figure 7: Circles: potential, key points and paths.

$\mathcal{U}_{p_{n+1}}(p)$, and its neighbors are updated as usual in Table 1. In case $V_n(p) \leq U(p)$, the point is put as alive with values $V_{n+1}(p) = V_n(p)$, and $U(p) = \infty$ and no update is done on its neighbors. This is a way to stop propagation around this point. This makes the whole propagation stop as soon as we passed over all points that are closer in energy to p_{n+1} than to the other previous p_k.

Therefore, the computation of the whole map does not cost much more than computation of the fast marching a few times over the image (a rough estimation is $\log N$ times, with N the number of p_k's instead of N times in case we would recompute the map at each step). Thus the computation time of this step is not too much dependent on the number of key points. We see in Fig. 6 an example of running this algorithm on the ellipse image. Notice the order in which the points p_k were chosen. The first p_0 is on the top of the ellipse. In consequence the second point p_1 is on the bottom. Then p_2 and p_3 are on right and left. On the second and third rows of the figure, we show the partial map computation, that is the set of pixels for which a new value of minimal action was computed. For such a simple example, we see in the energy map to the first point p_0 that the second key point is in fact the *saddle point* between p_0 and itself.

4.4 Finding the Saddle points

In the fast marching algorithm, as we modified it in the previous section, we have a simple way to find and update the linked neighbors and *saddle points*. The definition and criteria for finding a *saddle point* is the same as in the algorithm of Section 3. However, since we add key points at each step, some *saddle points* detected earlier are not *saddle points* anymore. So we have to check each time a *saddle point* is set as alive in a new *region*. It is then removed from the set of *saddle points* (see Fig. 8). This comes from the fact that this point is no more on the boundary of the previously obtained regions. Often, the new key point added was itself a *saddle point*, and it is also removed from the set of *saddle points*. Since the *saddle point* between two *key points* may change during the algorithm, it is easier to define the *selected saddle points* only at the end, once all *key points* are known.

We see in Fig. 6 results on an ellipse for the determination of key points and their selected *saddle*

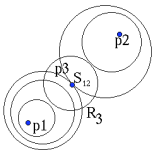

Figure 8: The *saddle point* between p_1 and p_2 is not a *saddle point* anymore when included in region R_3.

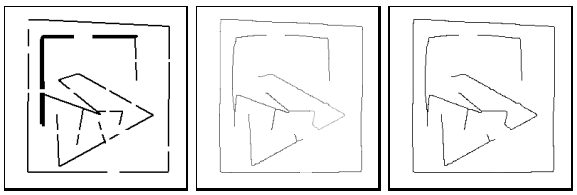

Figure 9: Complex exemple: From left to right:potential, final paths with energy as grey level and final paths with 30 key points.

points. The paths that are obtained correspond to the completed curve that have filled in the holes. Fig. 7 illustrates the capacity of our method to deal with a contour image including spurious points and more than one curve. In the example of Fig. 9, more complex data is taken and we show the results with 30 key points. The result gives a simplified and completed set of contour curves. We see in this example that limiting the number of linked neighbors to at most two linking paths can change the way the contours are completed. We show in this figure the energy of the found paths. Each time we compute a path between two points p_k and p_j, we know the *saddle point* $S(p_k, p_j)$ and its energy V_N. This energy is in fact equal to the cost of the path which links $S(p_k, p_j)$ to p_k and to p_j. Therefore the energy of the path between p_k and p_j is equal to $2V_N(S(p_k, p_j))$. The smaller this energy is, the more reliable the path can be considered. It could be a criteria to choose best curves if necessary in more complex images as in [5].

5 Conclusion

We presented a new method that finds a set of contour curves in an image. It was applied to perceptual grouping to get complete curves from a set of noisy contours or edge points with gaps. In a first method, we assume given a set of key points, and we found the pairs of key points that had to be linked with minimal paths. In a second method, the set of key points is automatically extracted from a set of admissible points, which can be the whole set of edge points. The whole set of minimal paths completes the initial set of contours and allows to close these contours.

References

[1] Laurent D. Cohen and R. Kimmel. Global minimum for active contour models: A minimal path approach. *IJCV*, 24(1):57–78, August 1997.

[2] M. Kass, A. Witkin and D. Terzopoulos. Snakes: Active contour models. *IJCV*, 1(4):321–331, Jan. 1988.

[3] R. Malladi, J. A. Sethian, and B. C. Vemuri. Shape modeling with front propagation: A level set approach. *IEEE PAMI*, 17(2):158–175, february 1995.

[4] V. Caselles, R. Kimmel, and G. Sapiro. Geodesic active contours. *IJCV*, 22(1):61–79, 1997.

[5] A. Shaashua and S. Ullman. Structural saliency: The detection of globally salient structures using a locally connected network. In *Proc. ICCV'88*, Dec. 1988.

[6] G. Guy and G. Medioni. Inferring global perceptual contours from local features. *IJCV*, 20(1/2) Oct. 1996.

[7] L. R. Williams and D. W. Jacobs. stochastic completion fields: a neural model of illusory contour shape and salience. In *Proc. ICCV'95*, June 1995.

[8] L. R. Williams and D. W. Jacobs. Local parallel computation of stochastic completion field. In *Proc. IEEE CVPR'96*, San Francisco, USA, June 1996.

[9] R. Kimmel, N. Kiryati, and A. M. Bruckstein. Distance maps and weighted distance transforms. *JMIV*, 6:223–233, May 1996.

[10] R. Kimmel, A. Amir, and A. Bruckstein. Finding shortest paths on surfaces using level sets propagation. *IEEE* PAMI-17(6):635–640, June 1995.

[11] Laurent D. Cohen. On active contour models and balloons. *CVGIP:IU*, 53(2):211–218, March 1991.

[12] J. A. Sethian. *Level Set Methods: Evolving Interfaces in Geometry, Fluid Mechanics, Computer Vision and Materials Sciences*. Cambridge Univ. Press, 1996.

[13] Laurent D. Cohen and Isaac Cohen. Finite element methods for active contour models and balloons for 2-D and 3-D images. *IEEE* PAMI-15(11), Nov. 1993.

[14] Laurent D. Cohen. Multiple contour finding and perceptual grouping using minimal paths. TR 0101, CEREMADE, January 2001. To appear in JMIV.

[15] T. Deschamps and Laurent D. Cohen. Minimal paths in 3D images and application to virtual endoscopy. In *Proc. ECCV'00*, Dublin, Ireland, July 2000.

[16] E. W. Dijkstra. A note on two problems in connection with graphs. *Numerische Math.*, 1:269–271, 1959.

[17] D. Geiger, A. Gupta, L. Costa, J. Vlontzos. Dynamic programming for detecting, tracking, and matching deformable contours. *IEEE PAMI*, 17(3), Mar. 1995.

Appendix : 2D Up-Wind Scheme

Notice that for solving Eqn. (4), only alive points are considered. Considering the neighbors of grid point (i, j) in 4-connexity, we note $\{A_1, A_2\}$ and $\{B_1, B_2\}$ the two couples of opposite neighbors such that we get the ordering $\mathcal{U}(A_1) \leq \mathcal{U}(A_2)$, $\mathcal{U}(B_1) \leq \mathcal{U}(B_2)$, and $\mathcal{U}(A_1) \leq \mathcal{U}(B_1)$. Considering that we have $u \geq \mathcal{U}(B_1) \geq \mathcal{U}(A_1)$, the equation derived is

$$(u - \mathcal{U}(A_1))^2 + (u - \mathcal{U}(B_1))^2 = \tilde{P}_{i,j}^2 \qquad (5)$$

Based on testing the discriminant Δ of Eqn. (5), one or two neighbors are used to solve it:

1. If $\tilde{P}_{i,j} > \mathcal{U}(B_1) - \mathcal{U}(A_1)$, solution of Eqn. (5) is $u = \frac{\mathcal{U}(B_1)+\mathcal{U}(A_1)+\sqrt{2\tilde{P}_{i,j}^2 - (\mathcal{U}(B_1)-\mathcal{U}(A_1))^2}}{2}$.
2. else $u = \mathcal{U}(A_1) + \tilde{P}_{i,j}$.