Multiple Contour Finding and Perceptual Grouping using Minimal Paths

Laurent D. Cohen
CEREMADE UMR CNRS 7534, Université Paris IX Dauphine,
Place du Marechal de Lattre de Tassigny, 75775 Paris Cedex 16, France
Tel : 33-1-44 05 46 78 Fax : 33-1-44 05 45 99 Email : cohen@ceremade.dauphine.fr

January 23, 2001

To appear in Journal of Mathematical Imaging and Vision, 2001.

Abstract

We address the problem of 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 [5]. Given a set of key points, we find the pairs of points that have to be linked and the paths that join them. We use the *saddle points* of the minimal action map. The paths are obtained by backpropagation from the *saddle points* to both points of each pair.

In a second part, we propose a scheme that does not need key points for initialization. A set of key points is automatically selected from a larger set of admissible points. At the same time, saddle points between pairs of key points are extracted. Next, paths are drawn on the image and give the minimal paths between selected pairs of points. The set of minimal paths completes the initial set of contours and allows to close them. We illustrate the capability of our approach to close contours with examples on various images of sets of edge points of shapes with missing contours.

Keywords: Perceptual grouping, salient curve detection, active contours, minimal paths, fast marching, level sets, weighted distance, reconstruction, energy minimization.

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 [10] 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 [13, 1] 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.

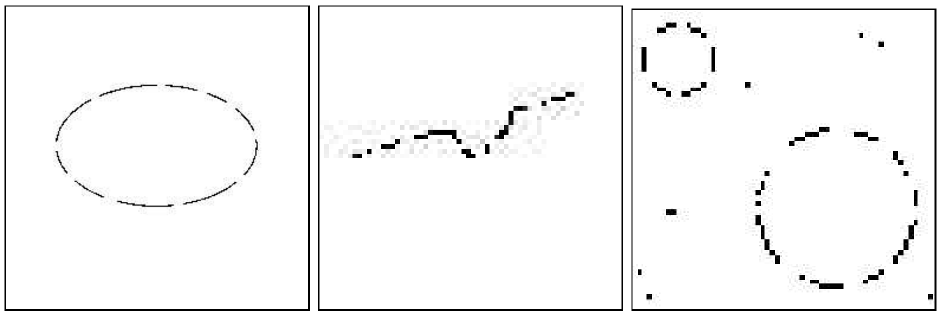

Figure 1: Examples of incomplete contours

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 figure 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 [16, 9, 17]. These methods find a criteria for saliency of a curve 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 ([10]) 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 [18] 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 [5] used the minimal paths, as introduced in [12, 11]. The goal was to avoid local minima without demanding too much on user initialization, which is a main drawback of classic snakes [3]. Only two end points were needed. The numerical method has the advantage of being consistent (see [5]) and efficient using the Fast Marching algorithm introduced in [15]. 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 [4]). 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 [5].

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

- Minimal path between two points: The solution proposed in [5] 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 between a given set of unstructured points: We propose a way to find the pairs of linked neighbors and the paths between them in 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 as presented in Section 4.

Finally, in Section 5, we conclude with possible extensions.

2 Minimal Paths and weighted distance

2.1 Global minimum for Active Contours

We present in this section the basic ideas of the method introduced in [5] to find the global minimum of the active contour energy using minimal paths. The energy to minimize is similar to classical deformable models (see [10]) where it combines smoothing terms and image features attraction term (Potential P):

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

where $C(s)$ represents a curve drawn on a 2D image and Ω is its domain of definition. For classic active contours, s may be any parameterization of the curve while with a geometric model, s is the arclength,

L is the length of the curve and $\Omega = [0, L]$. The authors of [5] 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 [1]. The method introduced in [5] improves energy minimization because the problem is transformed in a way to find the global minimum. It avoids the solution being sticked in local minima. It reduces the user initialization to giving the two end points of the contour C. Let us explain each step of this method.

2.2 Problem formulation

Most of the classical deformable contours have no constraint on the parameterization s, thus allowing the parameterization itself to be part of the minimization. In [5], contrary to the classical snake model (but similarly to geodesic active contours), s represents the arc-length parameter, which means that $\|C'(s)\| = 1$, leading to a geometric energy form. Considering a simplified energy model without the second derivative term leads to the expression $E(C) = \int \{w\|C'\|^2 + P(C)\}ds$. Assuming that $\|C'(s)\| = 1$ leads to the formulation

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

The regularization of this model is now achieved by the constant $w > 0$. This term integrates as $\int_\Omega w ds = w \times L$ and allows to control the smoothness of the contour (see [5] for details).
We now have an expression in which the internal energy can be included in the external potential. Given a potential $P \geq 0$ that takes lower values near desired features, we are looking for paths along which the integral of $\tilde{P} = P + w$ is minimal. The surface of minimal action $\mathcal{U}$ is defined as the minimal energy integrated along a path between a starting point p_0 and any point p:

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

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. Assuming that potential $P \neq 0$ (this is always the case for $\tilde{P}$), the action map has only one local minimum which is the starting point p_0. The minimal path is found by a simple back-propagation, that is a gradient descent on the minimal action map $\mathcal{U}$ starting from p_1 until p_0 is reached. Thus, contour initialization is reduced to the selection of the two extremities of the path. We explain in the next section how to compute efficiently the action map $\mathcal{U}$.

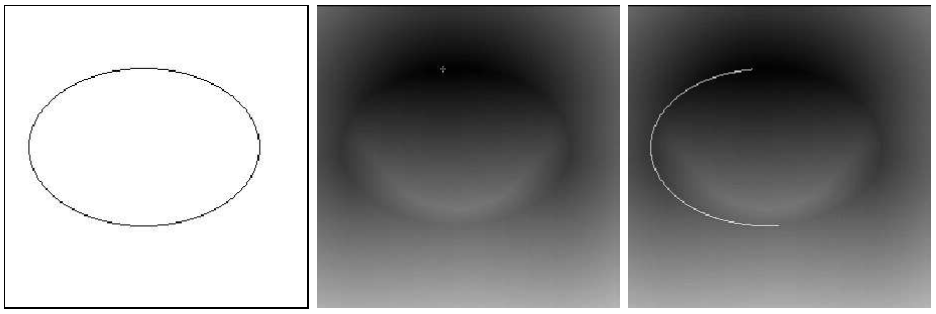

Figure 2: Finding a minimal path between two points. On the left, the potential is minimal on the ellipse. In the middle, the minimal action or weighted distance to the marked point. On the right, minimal path using backpropagation from the second point.

2.3 Fast Marching Resolution

In order to compute this map $\mathcal{U}$, a front-propagation equation related to Equation (3) is solved:

$$\frac{\partial C}{\partial t} = \frac{1}{\tilde{P}}\vec{n}. \tag{4}$$

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 the point p.

The *Fast Marching* technique, introduced in [15], was used in [5] noticing that the map $\mathcal{U}$ satisfies the Eikonal equation:

$$\|\nabla \mathcal{U}\| = \tilde{P} \quad \text{and} \quad \mathcal{U}(p_0) = 0. \tag{5}$$

Classic finite difference schemes for this equation tend to overshoot and are unstable. An up-wind scheme was proposed by [15]. It relies on a one-sided derivative that looks in the up-wind direction of the moving front, and thereby avoids the over-shooting associated with finite differences:

$$\begin{aligned}(\max\{u - \mathcal{U}_{i-1,j}, u - \mathcal{U}_{i+1,j}, 0\})^2 &\;+\\ (\max\{u - \mathcal{U}_{i,j-1}, u - \mathcal{U}_{i,j+1}, 0\})^2 &\;=\; \tilde{P}^2_{i,j},\end{aligned} \tag{6}$$

giving the correct viscosity-solution u for $\mathcal{U}_{i,j}$. 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 Equation (6).

This technique of considering at each step only the necessary set of grid points was originally introduced for the construction of minimum length paths in a graph between two given nodes in [7].

Algorithm for 2D Fast Marching

- Definitions:
  - *Alive* set: all 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 Equation (6) 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: reduced to the starting point p_0, with $U(p_0) = \mathcal{U}(p_0) = 0$;
  - *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, with $\mathcal{U} = U = \infty$;
- 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 (i.e. $\mathcal{U}(p) = U_{i_{min}, j_{min}}$ is frozen);
  - For each neighbor (i, j) (4-connexity in 2D) of (i_{min}, j_{min}):
    - If (i, j) is *Far*, add it to the *Trial* set and compute $U_{i,j}$ using Eqn. 6;
    - If (i, j) is *Trial*, update the action $U_{i,j}$ using Eqn. 6.

Table 1: *Fast Marching* algorithm

The algorithm is detailed in Table 1. An example is shown in Figure 2. The *Fast Marching* technique selects at each iteration the *Trial* point with minimum action value. In order to compute this value, we have to solve Equation (6) for each trial point, as detailed in Table 5.

Thus it needs only one pass over the image. To perform efficiently these operations in minimum time, the *Trial* points are stored in a min-heap data structure (see details in [15]). Since the complexity of the operation of changing the value of one element of the heap is bounded by a worst-case bottom-to-top proceeding of the tree in $O(\log_2 P)$, the total work is bounded $O(P \log_2 P)$ for the *Fast Marching* on a grid with P nodes.

3 Finding multiple contours from a set of key points p_k

The method of [5], 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 [8] 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 equation 4. 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 Figures 3 and 4).
Our problem is related to the approach presented at the end of [5] 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. This is in fact an approximation, due to some discretization error in finding the meeting point S. If high precision is needed, a subpixel location of *saddle points* can be made based on the final energy map. In order to get the precise minimal path between the two points, we could also backpropagate from the second point to the first as in Section 2, but computation time would be then much increased.

3.2 Some definitions

Here are some definitions that 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 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.

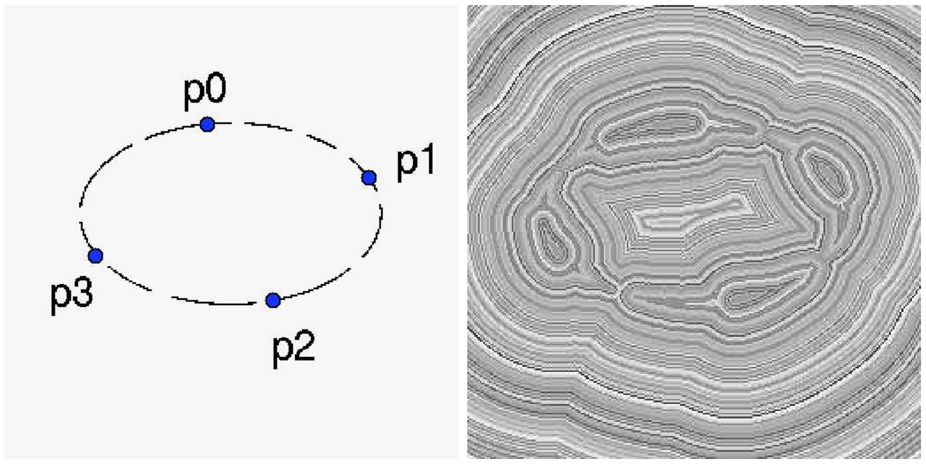

Figure 3: Ellipse example with four points. On the left the incomplete ellipse as potential and four given points; on the right the minimal action map (random LUT to show the level sets) from these points.

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 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 Figure 8 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 Figure 8, such an approach would link the spurious points with the circles. Postprocessing would be needed to remove undesired links, based on high energy for example.

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 Figure

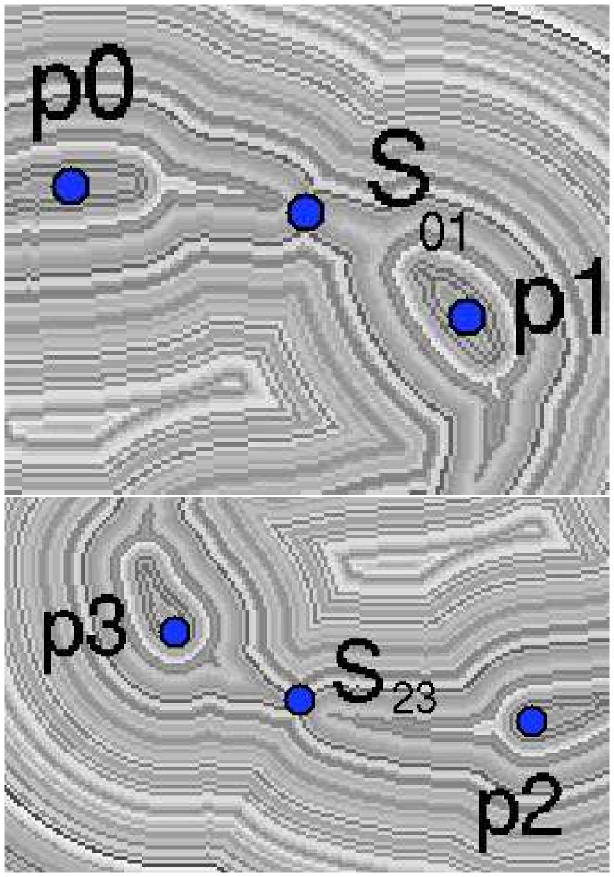

Figure 4: Zoom on *saddle points* between two key points.

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 *region* R_k associated with a point p_k is the set of points p of the image such that minimal energy $\mathcal{U}_{p_k}(p)$ to p_k is smaller than all the $\mathcal{U}_{p_j}(p)$ to other points p_j. The set of such *regions* R_k covers the whole image, and forms the Voronoi diagram of the image (see figure 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 $\mathcal{U}_{p_k}(p) = \mathcal{U}_{p_j}(p)$.

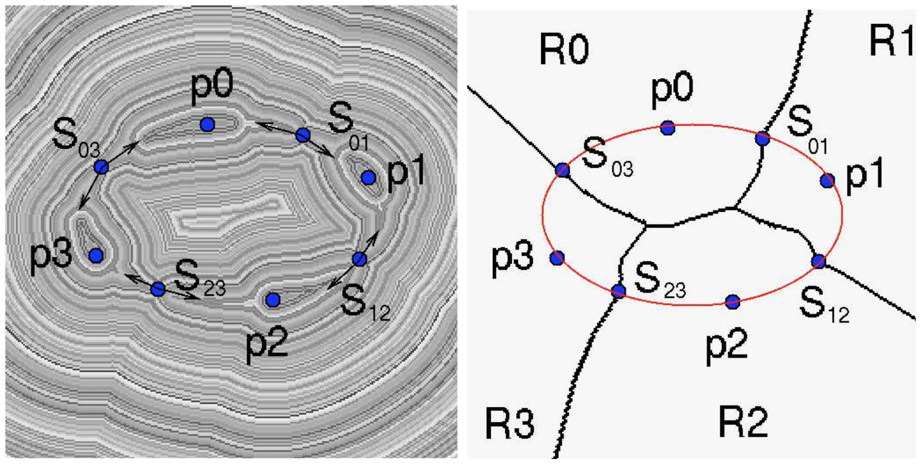

Figure 5: Ellipse example with four points. On the left the *saddle points* are found, and backpropagation is made from them to each of the two points from where the front comes; on the right, the minimal paths and the Voronoi diagram obtained.

The *saddle point* $S(p_k, p_j)$ is a point on this boundary with minimal value of $\mathcal{U}_{p_k}(p) = \mathcal{U}_{p_j}(p)$. This gives us 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 (6). In that formula, the computation of $U_{i,j}$ depends only on at most two of the four pixels involved. Following notations of Table 5, 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 figures 3 and 5. When there is a large number of p_k's, this does not

```
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. 6.
  • 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 (7) (see ap-
        pendix). If $R(A_1) \neq R(B_1)$ and we are in case 1 of table 5 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 $(i, j)$ (4-connexity) of $(i_{min}, j_{min})$:
          * If $(i, j)$ is Far, add it to the Trial set and compute $U$ using Eqn. 6;
          * If $(i, j)$ is Trial, recompute the action $U_{i,j}$, and update it.
  • 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

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 ([2]). 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 Figure 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.

4 Finding a set of key points p_k

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, and 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 the highest energy point 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.

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

Algorithm with automatic selection of p_k

- The set $\mathcal{A}$ of *admissible points* is defined in section 4.2;
- Initialization:
  - p_0 is chosen among the *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 the points where $V_{n+1} \leq V_n$ (see Section 4.3).
  - Update the set of *saddle points* (see Section 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 thresholds.
- 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

the gradient of the potential to be large enough. This is to impose two kinds of properties:

- If the set of points contains thick curves, this keeps only points that are on the boundary.
- 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 the set of admissible points.

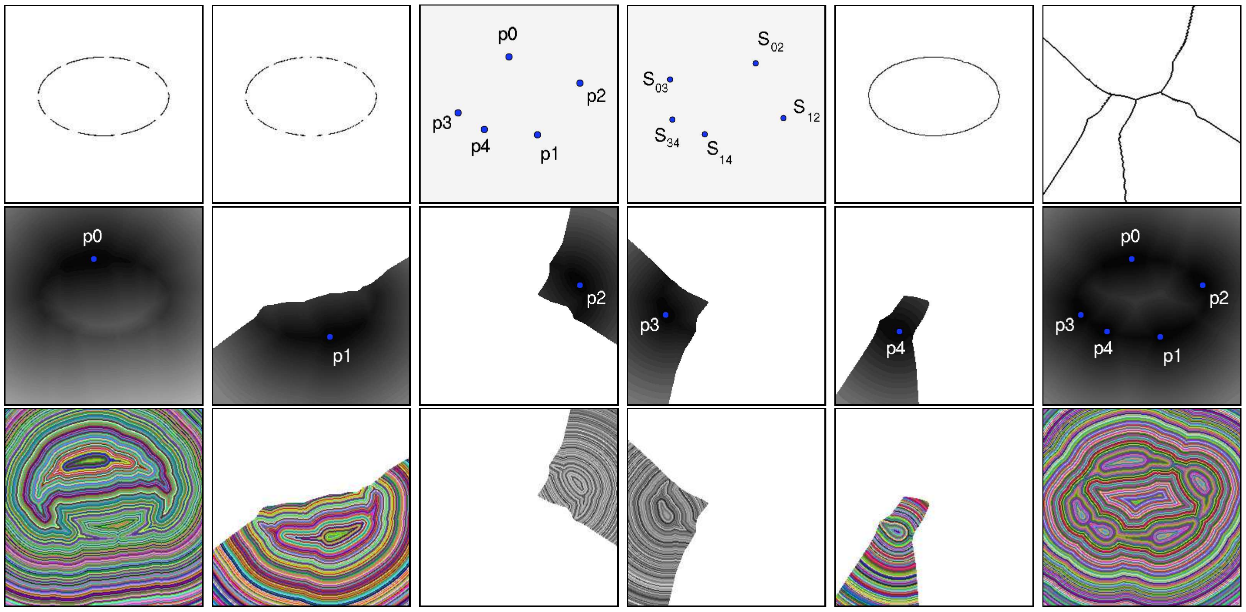

Figure 6: Ellipse example: successive partial map computation for five points. From left to right, line 1: potential, admissible points, found key points, *saddle points*, final paths and voronoi diagram; line 2: successive partial maps for the 5 key points and final map; line 3: the same with random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

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 $V_{n+1}(p) = U(p) = \mathcal{U}_{p_{n+1}}(p)$, 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

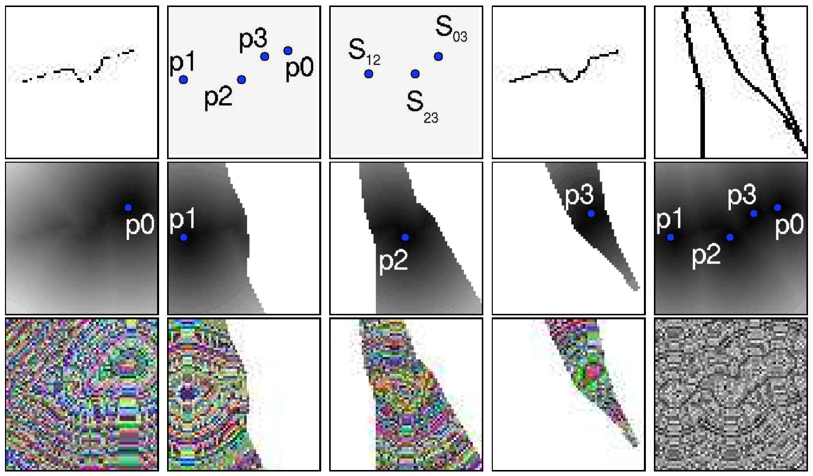

Figure 7: Curve example: same as in Figure 6 with successive partial map computation for four points.

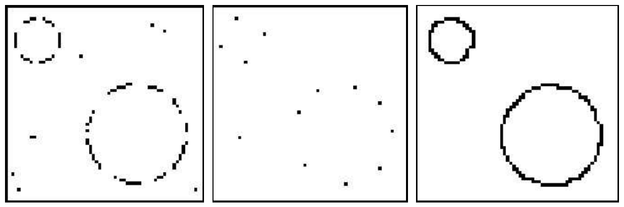

Figure 8: Two circles; From left to right: potential, key points and final paths.

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 Figure 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 pixel 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. Notice that this *saddle point* would be enough to find the complete ellipse through backpropagation both ways to p_0 as we did for finding a closed curve in [5]. In the second example, on Figure 7, the two extreme points on right and left are found first and then the two in the middle.

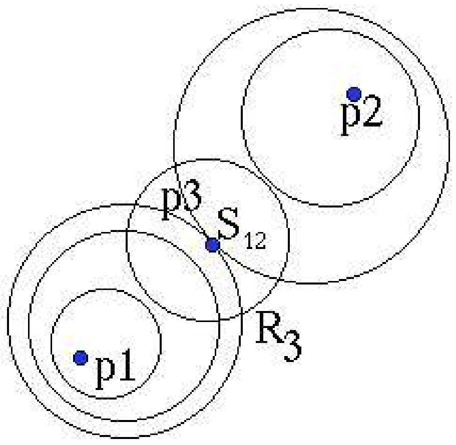

Figure 9: The *saddle point* between p_1 and p_2 is not a *saddle point* anymore when it becomes included in region R_3.

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 Figure 9). 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 Figures 6 and 7 results on simple curves for the determination of key points and their selected *saddle points*. In both cases, the paths that are obtained correspond to the completed curve that have filled in the holes. Figure 8 illustrates the capacity of our method to deal with a contour image including spurious points and more than one curve. In the example of Figure 10, more complex data is taken and we show the results with 30 and 40 key points. We see that the main contours are the same, the completed large square and a set of other curves. 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 con-

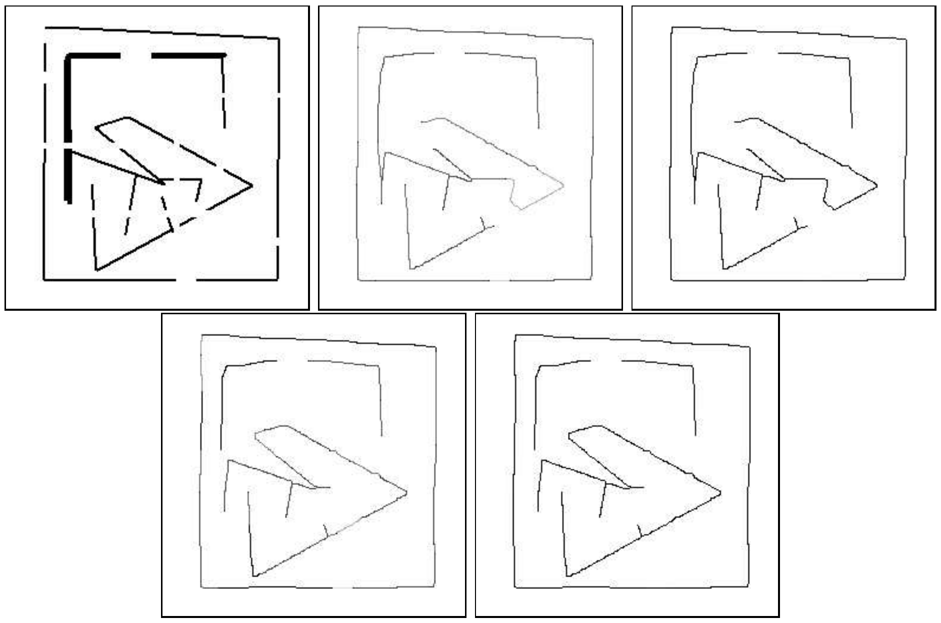

Figure 10: Complex exemple: From left to right, line 1: potential, final paths with energy as grey level and final paths with 30 key points; line2: same with 40 key points.

tours 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 the best curves if necessary in more complex images as in [16]. Notice in Figure 10 that you can only compare the energy of different paths in the same image, but the two images are not represented with the same color map.
We show in figure 11 an application of our approach combined with the saliency map of [9]. In such an example, the given dots are too few to enable finding the circle as a minimal path. Indeed, taking two opposite points on the ellipse, the minimal path between them will not be along the ellipse but rather along a straight line. By passing through low potential points (in black) along the circle, the path will also pass through more high potential points (background in white). Thus applying the method of [9] gives a saliency map that is much more dense than the original image. Taking the saliency map as potential, our approach allows finding the whole circle as a set of minimal paths between points determined automatically. The set of admissible points here can

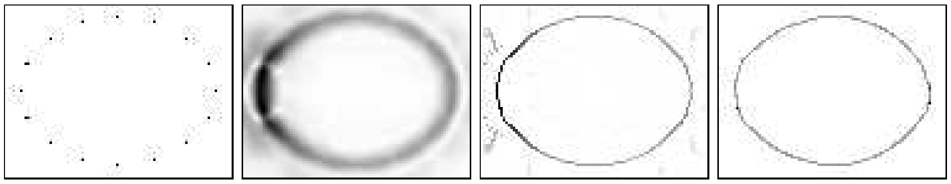

Figure 11: Finding a set of minimal path using a saliency map as potential. From left to right, original data, saliency map, ridge lines and minimal paths.

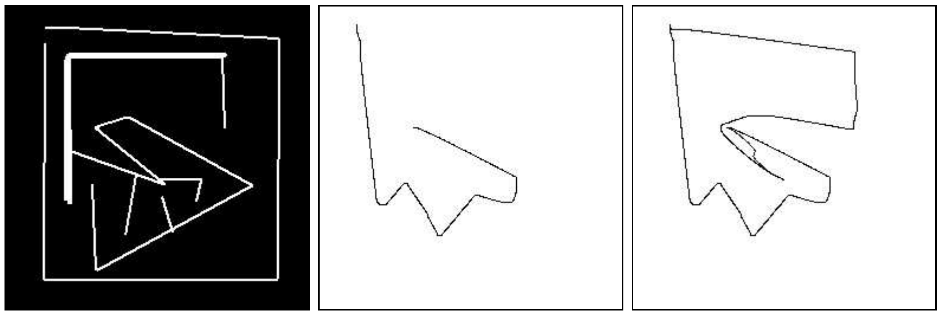

Figure 12: Our method permits to find a path inside the labyrinth. On the left the potential is obtained by reverse video from the contour potential of Figure 10. In the middle, we require two p_k's and on the right three.

be either the initial set of points of the original image or the set of points obtained by a threshold on the saliency map. We can also find the ellipse by looking for ridge curves on the saliency map but there are many spurious ridge curves obtained.

We summarize the ideas of sections 4.3 and 4.4 in Table 4, giving details of the main algorithm that were omitted in Table 3.

5 Perspectives and 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. The technique is based on previous work of finding minimal paths between two end points [5]. However, in our approach, we do not need to give the start and end points as initialization. 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 by 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. At the same time this set of points is obtained, *saddle points* between pairs of points are found. Once this set is obtained, paths are drawn on the image from the selected

```
Details of the Loop in Algorithm of Table 3
• Extra Initialization:
    – $R(p_0) = 0$
    – $\forall p \neq p_0, R(p) = -1$
• 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 $V_n$;
      remove $p_{n+1}$ from the set of saddle points.
    – Compute $V_{n+1} = \min(V_n, \mathcal{U}_{p_{n+1}})$. Fast Marching is initialized again to
      compute $\mathcal{U}_{p_{n+1}}$ as in Table 1 but limited to the points where $V_{n+1} \leq V_n$.
      We start with $V_{n+1} = V_n$. 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;
        * If $U(p) < V_n(p)$, set $V_{n+1}(p) = U(p)$, update $R(p) = n + 1$. If $p$
          is a saddle point, remove it from the set of saddle points. For each
          neighbor $(i, j)$ (4-connexity in 2D) of $(i_{min}, j_{min})$:
            · If $(i, j)$ is Far, add it to the Trial set and compute $U$ using Eqn.
              6;
            · If $(i, j)$ is Trial, recompute the action $U_{i,j}$, and update it.
        * If $U(p) \geq V_n(p)$, set $V_{n+1}(p) = V_n(p)$, $U(p) = \infty$, no update on
          $R(p)$ and linked neighbors is needed and no trial point is added. If
          this is the first time that region $n + 1$ meets region of index $R(p)$,
          $S(p_{R(p)}, p_{n+1}) = p$ is set as a saddle point between $p_{R(p)}$ and $p_{n+1}$.
• For each point keep as selected only at most the two linked neighbors for which
  the saddle points have smaller energy.
```

Table 4: Partial *Fast Marching* algorithm with *saddle points* update

saddle points to both points of each pair. This gives the minimal paths between selected pairs of points. The whole set of paths completes the initial set of contours and allows to close these contours.

The algorithms described in this paper apply to various potentials as well. This is only the definition of admissible points that may be different and adapted to each application. We show in Figure 12 an example of a “labyrinth” potential. The potential is small in the black area, and high along contours. Contours act in this example as barriers that stop the front propagation. Finding a minimal path between given end points would find the path inside the labyrinth as shown for example in [15, 6]. We show the resulting paths by asking to find automatically successively two and three key points. Our method enables to find automatically paths inside the labyrinth without even giving any start or end points. Thus it can be similarly applied to find edges or significant lines in real images.

Acknowledgements

This works started with the end of DEA project of Frédéric Claudel [2] and was continued in part with the DEA project of Benjamin Mauroy [14]. I thank them for their work at CEREMADE during summers of 1998 and 1999 respectively. Thanks also to Frédéric Sur for his help with Figure 11. I would like to thank Ron Kimmel for all his help with this paper and the reviewers for their useful suggestions to improve this paper.

References

[1] V. Caselles, R. Kimmel, and G. Sapiro. Geodesic active contours. *International Journal of Computer Vision*, 22(1):61–79, 1997.

[2] Frédéric Claudel. Extraction de contours implicites dans des images par des méthodes énergétiques. Technical report, CEREMADE, Septembre 1998. Rapport de DEA et Ecole Centrale, proposé et dirigé par Laurent Cohen.

[3] Laurent D. Cohen. On active contour models and balloons. *Computer Vision, Graphics, and Image Processing: Image Understanding*, 53(2):211–218, March 1991.

[4] Laurent D. Cohen and Isaac Cohen. Finite element methods for active contour models and balloons for 2-D and 3-D images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-15(11):1131–1147, November 1993.

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

[6] T. Deschamps and L.D. Cohen. Minimal paths in 3D images and application to virtual endoscopy. In *Proc. sixth European Conference on Computer Vision (ECCV'00)*, Dublin, Ireland, 26th June - 1st July 2000.

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

[8] D. Geiger, A. Gupta, L. Costa, and J. Vlontzos. Dynamic programming for detecting, tracking, and matching deformable contours. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 17(3), March 1995.

[9] G. Guy and G. Medioni. Inferring global perceptual contours from local features. *International Journal of Computer Vision*, 20(1/2):113–133, October 1996.

[10] Michael Kass, Andrew Witkin, and Demetri Terzopoulos. Snakes: Active contour models. *International Journal of Computer Vision*, 1(4):321–331, January 1988.

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

[12] R. Kimmel, N. Kiryati, and A. M. Bruckstein. Distance maps and weighted distance transforms. *Journal of Mathematical Imaging and Vision*, 6:223–233, May 1996. Special Issue on Topology and Geometry in Computer Vision.

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

[14] Benjamin Mauroy. Chemins minimaux en analyse d'images. Technical report, CEREMADE, Septembre 1999. Rapport de DEA, proposé et dirigé par Laurent Cohen.

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

[16] A. Shaashua and S. Ullman. Structural saliency: The detection of globally salient structures using a locally connected network. In *Proc. Second IEEE International Conference on Computer Vision (ICCV'88)*, pages 321–327, December 1988.

[17] L. R. Williams and D. W. Jacobs. stochastic completion fields: a neural model of illusory contour shape and salience. In *Proc. Fifth IEEE International Conference on Computer Vision (ICCV'95)*, pages 408–415, Cambridge, USA, June 1995.

[18] L. R. Williams and D. W. Jacobs. Local parallel computation of stochastic completion field. In *Proc. IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'96)*, San Francisco, USA, June 1996.

Appendix A : Algorithm for 2D Up-Wind Scheme

Notice that for solving Equation (6), only alive points are considered.

This means that calculation is made using current values of $\mathcal{U}$ for neighbors and not estimate U of other trial points. 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

$$(u - \mathcal{U}(A_1))^2 + (u - \mathcal{U}(B_1))^2 = \tilde{P}^2_{i,j} \tag{7}$$

Computing the discriminant Δ of Equation (7) we have the steps described in table 5.

1.
   - If $\Delta \geq 0$, u should be the largest solution of Equation (7);
     - If the hypothesis $u > \mathcal{U}(B_1)$ is wrong, go to 2;
     - If this value is larger than $\mathcal{U}(B_1)$, this is the solution;
   - If $\Delta < 0$, B_1 has an action too large to influence the solution. It means that $u > \mathcal{U}(B_1)$ is false. Go to 2;

   Simple calculus can replace case 1 by the test:

1bis. If $\tilde{P}_{i,j} > \mathcal{U}(B_1) - \mathcal{U}(A_1)$,
$u = \frac{\mathcal{U}(B_1)+\mathcal{U}(A_1)+\sqrt{2\tilde{P}^2_{i,j}-(\mathcal{U}(B_1)-\mathcal{U}(A_1))^2}}{2}$ is the largest solution of Equation (7)
else go to 2;

2. Considering that we have $u < \mathcal{U}(B_1)$ and $u \geq \mathcal{U}(A_1)$, we finally have $u = \mathcal{U}(A_1) + \tilde{P}_{i,j}$.

Table 5: Solving locally the upwind scheme

Technical Biography of the Author

Laurent D. Cohen was born in 1962. He was student at the **Ecole Normale Supérieure**, rue d'Ulm in Paris, France from 1981 to 1985.

He received the Master's and Ph.D. degrees in Applied Mathematics from University of Paris 6, France, in 1983 and 1986, respectively.
From 1985 to 1987, he was member at the Computer Graphics and Image Processing group at Schlumberger Palo Alto Research, Palo Alto, California and Schlumberger Montrouge Research, Montrouge, France and remained consultant for a few years with Schlumberger afterwards. He began working with INRIA, France in 1988, mainly with the medical image understanding group Epidaure.
Since 1990, he is Research Scholar with the French National Center for Scientific Research (CNRS) in the Applied Mathematics and Image Processing group at CEREMADE, University Paris-Dauphine, Paris, France. His research interests and teaching at the university are applications of variational methods and Partial Differential Equations to Image Processing and Computer Vision, like active contours, deformable models, minimal paths, surface reconstruction, Image registration, Image segmentation and restoration.