A MODEL FOR AUTOMATICALLY TRACING OBJECT BOUNDARIES

Fang Yang, Laurent D. Cohen

CEREMADE
Université Paris Dauphine PSL*
Place du Marechal de Lattre de Tassigny
75775 Paris cedex 16, France

Alfred M. Bruckstein

Technion
Computer Science Department
718 Taub Building Technion-IIT
Haifa 32000, Israel

ABSTRACT

In this paper, we propose a novel algorithm for tracing object boundaries automatically based on a model called "point flow" in image induced vector fields. An ordinary differential equation describes the movement of points under the action of an image-induced vector field and generates induced trajectories. The trajectories of the flows allow to find the edges of the image.

We test our method on real image dataset. Compared with the other classical edge detection models, our point flow method is better at providing precise and continuous curves. The experiment results testify the robustness and effectiveness.

Index Terms— Automatically Tracing Object Boundaries, Induced Vector Field, Point Flow, Differential Equations, Edge Detection

1. INTRODUCTION

Edge detection focuses on finding the sharp discontinuities and aims at capturing places where important changes occur in images. It plays an important role in image processing and computer vision. Since the early years of image processing, numerous methods were proposed to detect edges, such as Sobel operator, Prewitt operator, Canny and the Haralick edge detector, and so on [1].

In [2], the authors proposed a semi-automatic method to detect boundaries of objects, using simulation of particle motion in an image induced vector field. But users should provide the location of the starting point and the number of time steps. In addition, users have to adjust the parameters to achieve a good result. Then Makhervaks *et al* [3] used a similar model to track and detect the most important edges, in order to produce artistic one-liner renderings of objects appearing in images. Lu *et al.*[4] presented a *c*-evolute model for the particle motion in [2] to approximate the edge curves. More recently, Kimmel and Bruckstein [5] proposed to incorporate the Haralick/Canny edge detector into a variational edge integration process.

In this paper, we are interested in providing a model which can simulate the process of tracking object boundaries automatically. Imagine that the vector field is a magnetic field on the image. As the input, a number of points which are arranged randomly in the image can be considered as a series of small pieces of irons. When the magnetic field starts working, the iron pieces start moving following the direction of the magnetic field. Based on this idea, we record the trajectories of these points and use them to obtain the edges on the images. The movement of the points can be described by an ordinary differential equation, which is called "point flow".

After obtaining the vector field, we initiate the flow from a number of random points in the image plane. With the help of the vector field, the trajectories of these points attracted are towards and along the significant edges in the image. Note that the flow process will not end until a stopping criteria is met. An iterative process will allow to refine the trajectories and make the result more robust.

The contribution of this paper is that we propose a new way based on point flow for an automatic edge detection model to detect and integrate edges. Moreover, the edges that are extracted are continuous and precise.

The paper is organised as follows: Section.2 introduces the core algorithm of the point flow method, including the construction of the vector field, the edge detection and integration algorithms. Section.3 shows the experiments results on BSD500 dataset[6] and comparison with the classical edge detection methods. Section.4 provides some concluding remarks and possible directions for future work.

2. POINT FLOW METHOD

The point flow model is an ordinary differential equation (ODE) which describes the motion trajectory of a moving point under a vector field $\mathbf{V}$ within a period of time. In a $2-dimensional$ domain, the point flow model is defined as in Eq.(1):

$$\frac{d(P(t))}{dt} = \mathbf{V}(P(t)) \tag{1}$$

where $P(t) = (x(t), y(t))$ is a point function which describes the location of a moving point at time t and starting from a given point p_0 at time $t = 0$. Within time ΔT, the trajectory of P under the effect of the vector field $\mathbf{V}$ will be recorded, where $\mathbf{V}$ is a vector field that controls the speed and direction of the points.

In this paper, we use the point flow model to trace the object boundaries. The vector field $\mathbf{V}$ is a very crucial factor for tracing the boundaries. The details of designing the vector field are in the following section. The processing chain of point flow method is shown in Fig.1.

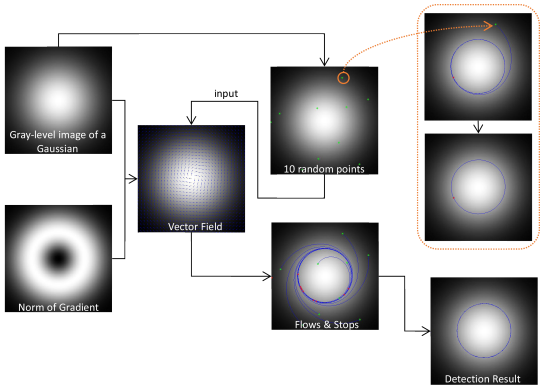


Fig. 1. The flowchart of the point flow algorithm. The left column presents a 2D Gaussian image and the magnitude of its gradient. A linear combination of their gradients are used to form the vector field $\mathbf{V}$. 10 random points are used as the starting points, moving under the effect of the vector field. The top-right dashed-box shows the movement of a single point, the blue curve is the trajectory and the red point is the end point. The flow terminates when it hits its own trajectory. Then, we re-initiate the flow from the end (red) point to obtain a complete and precise contour of the Gaussian image. The other points move in the same way. Right below shows the result.

2.1. Construction of Vector Field

Given an image $I : \Omega \to \mathbb{R}^2$, based on Eq.(1), for a starting point $p_0 = P(t_0)$, under the action of the vector field $\mathbf{V}$ for a time of ΔT, we can get a trajectory l_{p_0}. To make l_{p_0} march to/on the object boundaries or edges as much as possible, we need to construct a vector field which not only directs the flow towards the image edge, but also keeps the flow on the edges. In other words, there should be two kinds of local forces $\mathbf{V}_1$ and $\mathbf{V}_2$ which form the vector field: $\mathbf{V}_1$ pushes the points towards the high gradient places, and once near an edge, $\mathbf{V}_2$ makes the points move along the edge. So $\mathbf{V}$ combines both: $\mathbf{V} = \zeta \cdot \mathbf{V}_1 + \xi \cdot \mathbf{V}_2$.

Since the gradient vector is usually orthogonal to the edge orientation, $\mathbf{V}_2$ can be perfectly fitted by the orthogonal of the gradient of the image I, because it has a direction along the edges, so $\mathbf{V}_2 = \nabla I^{\perp}$. For $\mathbf{V}_1$, the second-order derivative of I can be used, namely, $\mathbf{V}_1 = \nabla(\|\nabla I\|)$. The advantage of the second-order derivative lies in that it generates a force that drives the points to the edges from both sides of the edge. Any lower- or higher-order derivative can not provide such precise information. This term is similar to one of the terms in the snake model evolution equation [7]. This term in the snake model is called "image force" that pushes the given curve to the significant lines which correspond to the desired features. The combination of these two terms can form exactly the vector field desired. And $\mathbf{V}$ can be written as follows:

$$\mathbf{V} = \zeta \cdot \nabla\|\nabla I\| \pm \xi \cdot \nabla I^{\perp} \tag{2}$$

where ζ and ξ are two constants: $0 \leq (\zeta, \xi) \leq 1$. They control the proportion between the two terms $\mathbf{V}_1$ and $\mathbf{V}_2$ which can be used to balance these two components thus makes $\mathbf{V}$ appropriate and suitable for detecting edges. Empirically we choose $\zeta = \xi = 0.5$, which fit for most of the cases. $\mathbf{V} = \frac{1}{2}(\mathbf{V}_1 \pm \mathbf{V}_2)$, here $\pm$ is used to change the direction of $\mathbf{V}$ by $180°$, which is an important step for edge integration. It means that a line can be followed in both directions. Here we use $\mathbf{V}$ to represent $\frac{1}{2}(\mathbf{V}_1 + \mathbf{V}_2)$ and $\mathbf{V}'$ to represent $\frac{1}{2}(\mathbf{V}_1 - \mathbf{V}_2)$.

2.2. Edge Integration

As stated above, after the construction of the vector field, a set of random points are chosen to flow as the starting points in the image plane. At the beginning, all starting points move in accordance with speed $\mathbf{V}$. The trajectories of the movement of these points are recorded in $P(\cdot)$. And they will not stop moving until a stopping criteria is met.

Here we define three stopping criterions for the flow:

1. When the flow hits itself. This kind of end points is the first type of end points, labelled as "*E1*".

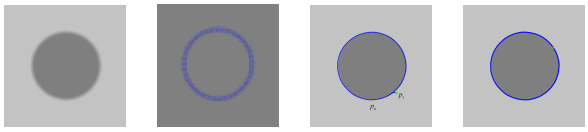
2. When the flow hits the boundary of the image. This kind of end points is the second type of end points, labelled as "*E2*".

3. When the flow hits a pixel where the gradient is zero. If it is at the source point where the gradient is zero, we will remove this source point. For the other cases, we will not consider the flow unless it merges with other flows. This case is detailed in the next section.

Endpoints labelled as "*E1*"

For a trajectory l_{p_s}, it starts from p_s and ends at p_e. If p_e is labelled as "*E1*", we re-initiate the movement from p_{e_1} in the same vector field $\mathbf{V}$. In this situation, l_{p_e} will surely hit itself and according to the stopping criteria 1, it will stop and form a closed curve. This closed curve is the boundary of a shape. Fig.2 illustrates the integration process of the first kind of end points.

Endpoints labelled as "*E2*"



(a) (b) (c) (d)

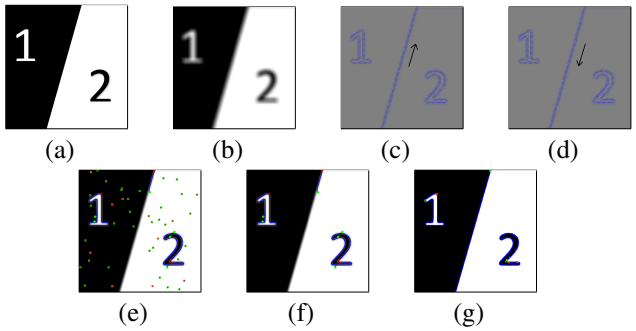
Fig. 2. From left to right are: the original image; the vector field $\mathbf{V}$; a trajectory from the green point to the red point; the final result.

Let us call the boundary of the image ∂I. When a trajectory l_{p_s} moves to a point p_e and $p_e \in \partial I$, the motion stops. In order to extract a complete edge, we propose to reflow from p_e, while in an opposite direction, namely, $\mathbf{V}'$.

Let us take Fig.3 as an example. We initiate the point flow from 50 random points under the vector field Fig.3(c), shown in Fig.3(e), many points end immediately when they start to move, because they satisfy the third stopping criteria. These points are removed in Fig.3(f) and they will no longer play a role in the next steps.

For the two numbers (or shapes) "1" and "2" in the image, obviously, the boundaries are closed curves. The way to detect and integrate the edges on these two shapes are in the same situation as "*EI*".

For the slash which separates the black and white region, the two terminals of this slash lie at the top and bottom boundary of the image, we call this slash the "split line". From Fig.3(f) we can see that there is a short curve starting from nearby the "split line" and ends at the top boundary of the image. The flow restarts from this end point under $\mathbf{V}'$ (Fig.3) until it meets the bottom boundary of the image, the endpoint $p_{e'} \in \partial I$. Thus, all the edges on the image are extracted.



(a) (b) (c) (d)

(e) (f) (g)

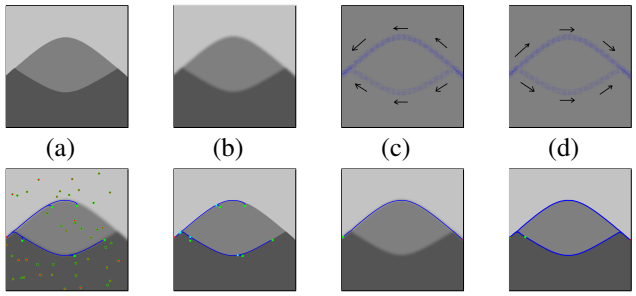
Fig. 3. from left to right, above to below, these figures are: (a) the original image; (b) image filtered by a 2D Gaussian; (c) vector field $\mathbf{V}$; (d) vector field $\mathbf{V}'$; (e) primary flow result from 50 random points (green points are the starting points and red ones the end points); (f) result after removing the non-edge points; (g) final result. (the black arrows on (c) and (d) illustrate the direction of the vector field of the corresponding boundary)

An Additional Step

This part is designed for the situation when two or more different streams converge and flow to the same endpoint p_e. Obviously, for closed curves, this case can be excluded.

Fig.4 serves as an example for this case. For the starting (green) points in Fig.4(e), they flow under the action of $\mathbf{V}$ (Fig.4(c)). After removing the third type of endpoints, for which the source and end points are at the same location, all flows merge into one stream and come to the same end point p_e, shown in Fig.4(f). According to the integration method mentioned in the part above, we need to re-initiate the flow from the endpoint on the vector field $\mathbf{V}'$ (Fig.4(d)). On its way back, it will meet the intersection, and here comes the dilemma – which way should it choose. Normally, the flow l_{p_e} goes into the part where there is stronger vector field, as shown in Fig.4(g). However, this will lead to a loss of important information. Therefore, we propose a way to complete the detection result.

In Fig.4(f), there are five starting points, which generate five trajectories. When any two trajectories intersect, we record the first point where they meet, as the cyan points in Fig.4(f). These points are recorded in $\{p_{c_{i,j}}\}$, where i and j stand for the ith or jth trajectory, $(i, j) \in \{1, ..., N\}$ and $i \neq j$ ($N = 5$ in this case). If a point $p_{c_{i,j}}$ is on or very close to l_{p_e}, we will not consider it. Only those points which are far away from l_{p_e} will be taken into account. So the intersection points that are on or close to the trajectory on Fig.4(g) will be removed. The two points on the lower edge are regarded as two new starting points for another flow process. This time, it will flow on both vector field $\mathbf{V}$ and $\mathbf{V}'$ until a stopping criteria is met. In addition, in order to avoid repetition, if one new starting point is covered by the trajectories of the previous ones, it will be removed from the set of new starting points, and so forth. Fig.4(h) shows the final complete integration result.



(a) (b) (c) (d)

(e) (f) (g) (h)

Fig. 4. from left to right, above to below, these figures are: (a) the original image; (b) image filtered by a 2D Gaussian; (c) vector field $\mathbf{V}$; (d) vector field $\mathbf{V}'$; (e) primary flow result from 50 random points (green points are the starting points and red ones the end points); (f) result after removing the non-edge points; the cyan points are the intersections between any two flows; (g) result by simply flowing back from a "*E2*" endpoint; (h) final result. (the black arrows on (c) and (d) illustrate the direction of the vector field of the corresponding boundary)

3. EXPERIMENTS

3.1. Data Settings

We test our point flow method on the images from the famous Berkeley Segmentation Dataset[6]. And we have also evaluated our method by using the 100 validation images from the same dataset. The number of random source points is set to be $N = 2000$ for each image. For most of the images, we use the grayscale to compute the vector field. For color images, we use the highest gradient at each pixel from the three channels RGB as the gradient of that pixel. For certain images, we use the HSV color space to compute the vector field. Before flowing, we use a Gaussian filter to smooth the images, the standard deviation of the filter is $\sigma = 2$, and the size is 4×4.

In addition, there are numerous ways to approximate numerically the solution to the first-order ODE Eq.(1), such as the Euler method, backward Euler method, first-order exponential integrator method and so on [8]. To make the solutions smoother and with less oscillations, we use the Runge-Kutta algorithm, also known as RK4, to solve Eq.(1).

Our detection provides a "hard" boundary result, not a probability map. So on the precision-recall curve map, what we present is a single point. We compare our method with the classical canny detector [9], the state-of-the-art pb detector [10], and the graph-based segmentation method untangling cycle [11].

Last but not least, our method presents a sub-pixel level detection result. While during the evaluation process, we have to assign each point on the trajectory a precise pixel location. This will lead to a loss of sub-pixel information.

3.2. Experiments Results and Discussions

We have tested our method on the widely known BSD dataset. Fig.5 shows some detection results using our proposed method, and the results are very inspiring. In terms of the evaluation, shown in Fig.6, our method nearly performs the best among the non-learning methods. The method is based on an implicit model and therefore, it exploits the model to generate the best segmentations possible for images complying with this model. Clearly, the BSD dataset are images selected from a wide range of images, they do not necessarily comply with our model.

Moreover, our method provides a precise sub-pixel level result. However, this advantage could not be reflected by the metric in the Berkeley benchmark.

4. CONCLUSION AND FUTURE WORK

This paper proposes a method for automatically modeling the process of tracking object boundaries in images. In this paper, we are just using the color (or graylevel) feature to construct the vector field. On this stage, our method is comparable among the numerous edge detection methods. In future, we

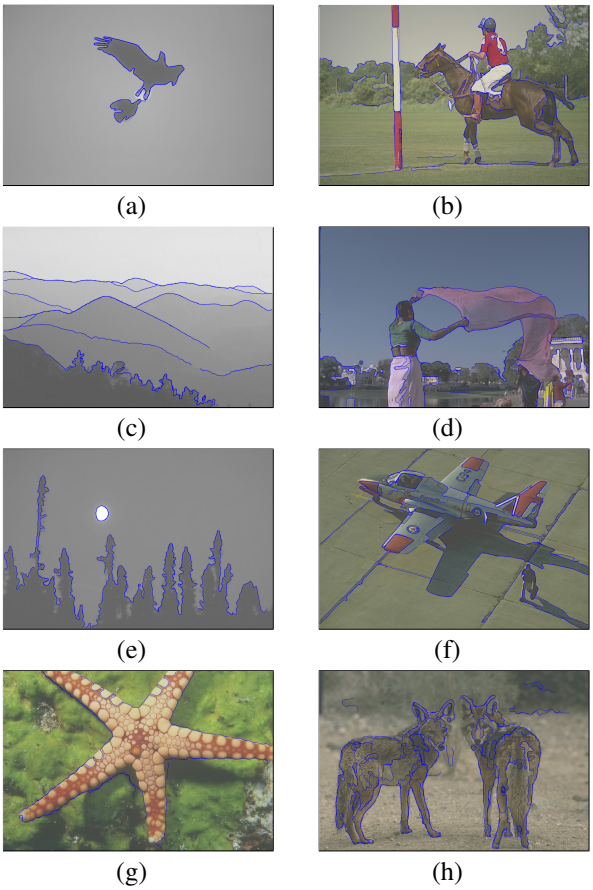


Fig. 5. some results on the BSD Dataset. (a), (c) and (e) are tests on gray images; (b), (d) and (f) are tests on color images; (g) and (h) are tests on the first and second channel of HSV space seperately.

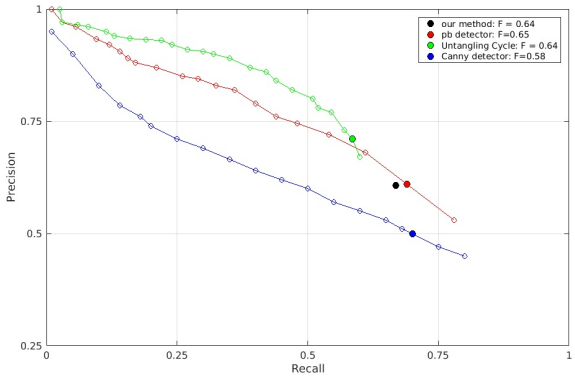


Fig. 6. Precision recall curve on the Berkeley benchmark, compared to Pb detector, untangling cycles and Canny detector.

hope to combine the other features such as the texture feature to construct vector fields for complicated scenes.

5. REFERENCES

[1] D. Ziou and S. Tabbone, "Edge detection techniques - an overview," *International Journal of Pattern Recognition and Image Analysis*, vol. 8, pp. 537–559, 1998.

[2] N. Eua-Anant and L. Udpa, "Boundary detection using simulation of particle motion in a vector image field," *Image Processing, IEEE Transactions on*, vol. 8, no. 11, pp. 1560–1571, 1999.

[3] V. Makhervaks, G. Barequet, and A. Bruckstein, "Image flows and one-liner graphical image representation," *Annals of the New York Academy of Sciences*, vol. 972, no. 1, pp. 10–18, 2002.

[4] C. Lu, Z. Chi, G. Chen, and D. Feng, "Geometric analysis of particle motion in a vector image field," *Journal of Mathematical Imaging and Vision*, vol. 26, no. 3, pp. 301–307, 2006.

[5] R. Kimmel and A. M. Bruckstein, "Regularized laplacian zero crossings as optimal edge integrators," *International Journal of Computer Vision*, vol. 53, no. 3, pp. 225–243, 2003.

[6] D. Martin, C. Fowlkes, D. Tal, and J. Malik, "A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics," in *Proc. 8th Int'l Conf. Computer Vision*, vol. 2, July 2001, pp. 416–423.

[7] M. Kass, A. Witkin, and D. Terzopoulos, "Snakes: Active contour models," *International journal of computer vision*, vol. 1, no. 4, pp. 321–331, 1988.

[8] J. C. Butcher, "Numerical methods for ordinary differential equations in the 20th century," *Journal of Computational and Applied Mathematics*, vol. 125, no. 1, pp. 1–29, 2000.

[9] J. Canny, "A computational approach to edge detection," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, no. 6, pp. 679–698, 1986.

[10] D. R. Martin, C. C. Fowlkes, and J. Malik, "Learning to detect natural image boundaries using local brightness, color, and texture cues," *IEEE transactions on pattern analysis and machine intelligence*, vol. 26, no. 5, pp. 530–549, 2004.

[11] Q. Zhu, G. Song, and J. Shi, "Untangling cycles for contour grouping," in *Computer Vision, 2007. ICCV 2007. IEEE 11th International Conference on*. IEEE, 2007, pp. 1–8.