PointFlow: A Model for Automatically Tracing Object Boundaries and Inferring Illusory Contours

Fang Yang[1], Alfred M.Bruckstein[2], and Laurent D.Cohen[1]

[1] CEREMADE, CNRS, UMR 7534, Université Paris Dauphine, PSL Research University, 75016 PARIS, FRANCE
[2] Computer Science Department 718 Taub Building Technion – IIT Haifa 32000, Israel

Abstract. In this paper, we propose a novel method for tracing object boundaries automatically based on a method called PointFlow in image induced vector fields. The PointFlow method comprises two steps: edge detection and edge integration. Basically, it uses an ordinary differential equation for describing 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 and integrate edges and determine object boundaries. We also extend the work of the original PointFlow method and make it adaptable to images with complicated scenes. In addition, the PointFlow method can be applied to infer certain illusory contours.
We tested our method on real image dataset. Compared with the other classical edge detection and integration models, our point flow method is better at providing precise and continuous curves. The experimental results clearly exhibit the robustness and effectiveness of the proposed method.

Keywords: Automatically Tracing Object Boundaries, Induced Vector Field, PointFlow, Differential Equations, Illusory Contours

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 the Sobel operator, Prewitt operator, Canny and the Haralick edge detector and so on [12]. The early edge detectors focus on the change of image brightness or image graylevel. However, many visually salient edges or contours do not simply correspond to gradients, but also to the texture edges or illusory contours. Therefore, researchers try to apply multiple features such as brightness, color, texture *.etc* to their edge detectors [10, 9, 1].

In [4], a semi-automatic method to detect boundaries of objects is proposed by using simulation of particle motion in an image induced vector field. But users of the method were required to provide the location of starting points and the number of time steps to be carried out. In addition, users needed to adjust the parameters to achieve a good result. In [7], a similar model is used to track and detect the most important edges, in order to produce artistic one-liner renderings of objects appearing in images. In [6] a c-evolute model is presented for the particle motion in [4] 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 process of tracking object boundaries automatically. Imagine that a magnetic vector field is induced by the image. As the input, a number of points which are arranged randomly in the image and are considered as small magnetized iron pieces. When the field is applied, the iron pieces start moving following the direction of the magnetic field. 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.

The construction of the vector field is a crucial step for PointFlow method. To trace the image boundaries, the vector field must be designed edge-oriented. After obtaining the vector field, we initiate the flow from a number of random points in the image plane. These points are then attracted towards and along the significant edges in the image. Note that the flow process will not end until a stopping criterion is met. An integration process allows to refine the trajectories and make the result more robust.

Additionally, the PointFlow method can be extended to infer the illusory contours under the assumption that the illusory contours are a missing part of a regular shape and it can be approximated by an arc of certain radian.

The contribution of this paper is that we propose a PointFlow method to detect and integrate edges. Commendably, the edges that are extracted are continuous and precise. Moreover, the PointFlow method can be applied to the inference of illusory contours.

The paper is organised as follows: Section.2 introduces the core algorithm of the PointFlow method. Section.3 details how to construct the vector field. Section.4 shows some experiments results on real images. Section.5 describes how the PointFlow method could be used to infer the illusory contours. Section.6 displays some inference results by using the PointFlow method. Section.7 provides some concluding remarks and possible directions for the future work.

2 PointFlow Method

The PointFlow method first uses an ordinary differential equation (ODE) to describe the motion of a moving point under a vector field $\mathbf{V}$ within a period of

time. In a $2D$ domain, the ODE is defined as follows:

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

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. The flow will not stop until some stopping criterion is met. Figure.1 displays the flowchart of applying the PointFlow method to trace image boundaries.

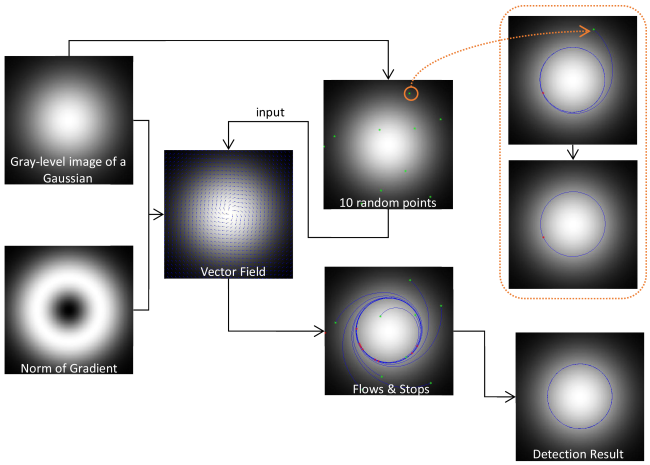


Fig. 1. The flowchart of the PointFlow. The left column presents a 2D Gaussian image and the magnitude of its gradient. A linear combination of their gradients are used to form the vector field $\mathbf{V}$. 10 random points are used as the starting points, moving under the effect of the vector field. 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 hit 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.

3 Construction of Vector Field

3.1 Vector Field by Using Graylevel

To design an edge-oriented vector field, two kinds of forces are necessary: one towards the edges and the other one along the edges. This can be realized simply by combining the tangent vector of the points on the image (along the edges) $\mathbf{V}_1 = \nabla I^\top$ and the gradient of the norm of the gradient of the image $\mathbf{V}_2 = \nabla(\|\nabla I\|)$ (towards the edges). Then the vector field is written as: $\mathbf{V} = \zeta\mathbf{V}_1 + \xi\mathbf{V}_2$ or $\mathbf{V}' = \zeta\mathbf{V}_1 - \xi\mathbf{V}_2$, empirically, $\zeta = \xi = 0.5$.

Three stopping and reflowing criteria are defined for the flow:

1. When the trajectory of flow hits itself. The kind of end points are the first type of end points, labelled as "*E1*". For points labeled as "*E1*", we re-initiate the movement from these points in the same vector field $\mathbf{V}$. The trajectories by re-initiating will result in closed curves, which are boundaries or contours of objects in images.
2. When the flow hits the boundary of the image. These kinds of end points are the second type of end points, labelled as "*E2*". For points labeled as "*E2*", we re-initiate the movement from these points in the vector field $\mathbf{V}'$. Thus a complete boundary from the boundary of the image will be detected.
3. When the flow hits a pixel where the gradient is zero. If it is at the source point where the gradient is very close to zero, we will remove this source point.

For the other cases, we will not consider the flow unless it merges with other flows. This is a case for junction points. Generally speaking, if a junction point is not revisited by any flow, it will start moving on both direction $\mathbf{V}$ and $\mathbf{V}'$ until it meets a stopping criterion.

Figure.2 displays some tests on real natural gray images. From (a) and (b), we can see that nearly all the curves and boundaries are perfectly extracted. But in (c), the contour of the seastar is not extracted while a lot of white spots on its back are detected. This is because in (a) and (b) the scenes are monotonous, and the subject can be easily distinguished by the graylevel. While in (c), the most salient changes in graylevel are the differences between the white spots and the background.

(a) (b) (c)

Fig. 2. Experiment on some gray images.

3.2 Vector Field by Using Multiple Features

As described above, in natural images, it is not always sufficient by only using the gradient of the graylevel to form the vector field. It is necessary to combine some other features to form the vector field. In this section, the BSDS500 dataset [1] is used for extracting the high level feature such as the texture and spectral features.

In [9], the authors proposed a *Pb* (probability of boundary) edge detector where they introduced several features, i.e. color, texture to detect the contours. In [1], the authors not only use the features in [9] but also introduced a spectral feature and combine them into a global feature to detect the contours, the

detector is known as the *gPb* (global probability of boundary) contour detector. To compute the gradient on each pixel of each feature channel, it is implemented by setting discs on each pixel (x, y) of each feature channel and comparing the histogram difference of each half-disc g and h using different orientations.

$$\chi^2 = \frac{1}{2} \sum_i \frac{(g(i) - h(i))^2}{g(i) + h(i)} \tag{2}$$

Normally, eight orientations are used to describe the directions of the gradient, where the orientation $\theta = [0, \pi/8, \pi/4, 3\pi/8, \pi/2, 5\pi/8, 3\pi/4, 7\pi/8]$.

To improve the performance of PointFlow in complicated scenes, we use the same feature in [1]. The color channels contain three channels which correspond to the CIE Lab colorspace including the brightness, color a and color b channels. The fourth channel is the texture channel, to obtain the texture feature, the primary thing is to compute the texton map. Firstly, each input training image from the BSDS dataset is convolved with a set of 17 Gaussian derivative, shown in Figure.3.

Fig. 3. 17 filters for computing the texture [1].

Then each pixel is associated with a 17-dimensional vector of responses. A K-means cluster is then used to define the clustering center, with $K = 64$. For the test image, we convolve them with the set of 17 Gaussian derivatives, too. The pixel which is nearest to one of the 64 cluster center will be given the corresponding texton label.

Let us denote the gradient signal $G_i(x, y, \theta)$, where i represents the feature channel and θ is the orientation. We also use the multi-scale and spectral features that were described in [1].

For the multi-scale features, each channel gradient G_i is computed at three scales $s = [\frac{\sigma}{2}, \sigma, 2\sigma]$, where s controls the radius of the disc for computing the histogram difference in Eq. (2). $\sigma = 5$ pixels for brightness channel and $\sigma = 10$ pixels for color a, color b and texture channel. The multi-scale gradient signals at each orientation can be combined linearly as follows:

$$mG(x, y, \theta) = \sum_s \sum_i \alpha_{i,s} G_{i,s}(x, y, \theta) \tag{3}$$

where $\alpha_{i,s}$ are the weights at different scale s and different feature channel i and it is learned by gradient ascent on the F-measure on the BSDS500 dataset.

The spectral features are used to obtain the most salient curves and it is obtained by a spectral clustering technique. A sparse symmetric matrix W which describes the affinity of two pixels of which the distance is within a fixed radius

r is provided according to mG. The spectral signals sG then are obtained by solving a Laplacian system of W, details can be found in [1].

By combining the multi-scale and spectral features, we can get the global signal at each orientation $gG(x, y, \theta)$.

Figure.4 shows the feature map gG of eight orientations.

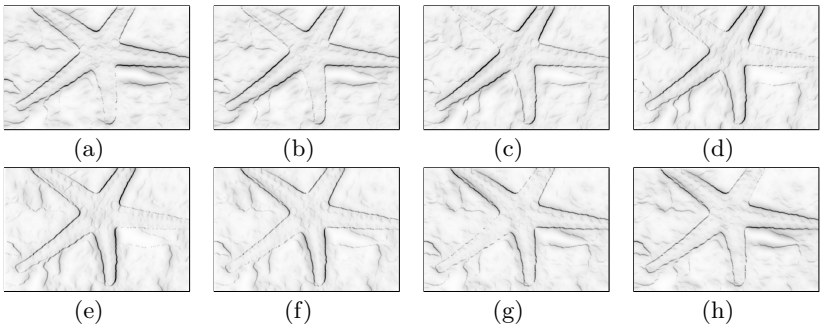


Fig. 4. Gradient magnitude of eight orientations, from (a) to (h), the orientation $\theta = [0, \pi/8, \pi/4, 3\pi/8, \pi/2, 5\pi/8, 3\pi/4, 7\pi/8]$

After computing the multiple features at each orientation, we can construct the vector field based on these features. For each single pixel, we will retain the largest value among the eight values as the gradient magnitude, and the corresponding orientation denotes the direction of the gradient:

$$\mathbf{v}(x, y) = gG_{\max}(x, y, \theta) * (\cos(\theta), \sin(\theta)) \tag{4}$$

so that: $\mathbf{V}_2 = \mathbf{v}^{\perp}$. To compute the second-order derivative, we use the gradient of $gG_{\max}(x, y)$, that is: $\mathbf{V}_1 = \nabla(gG_{\max})$.

Figure.5 shows an example of using multiple features to construct the vector field and its corresponding contour detection result. Compared with Figure.2 (c), the result in Figure.5 (c) is more clean and complete.

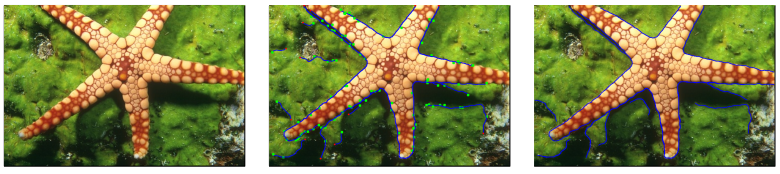


Fig. 5. Experiment on a real color image. From left to right are the original color image, the trajectories generated from 2000 random points and the result of detection of contours.

4 Experiments on Edge Detection

4.1 Data Settings

We test our PointFlow method on the images from the Berkeley Segmentation Dataset[8]. The number of random source points is set to be $N = 2000$ for each image. For vector fields using graylevel, 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 [2]. To make the solutions smoother and with less oscillations, we use the Runge-Kutta algorithm to solve Eq.(1).

4.2 Experiments Results and Discussions

Figure.6 are some results on the vector field generated by using only graylevel. From the results, we can see that for simple scenes, such as (a) and (d), by using graylevel to construct the vector is enough to obtain a complete and clean detection result. While in the complicated scenes, though that a lot of details are extracted, the result are not quite satisfying, this is because the object boundaries are affected by a lot of factors, e.g. colors, shadows, textures and so on.

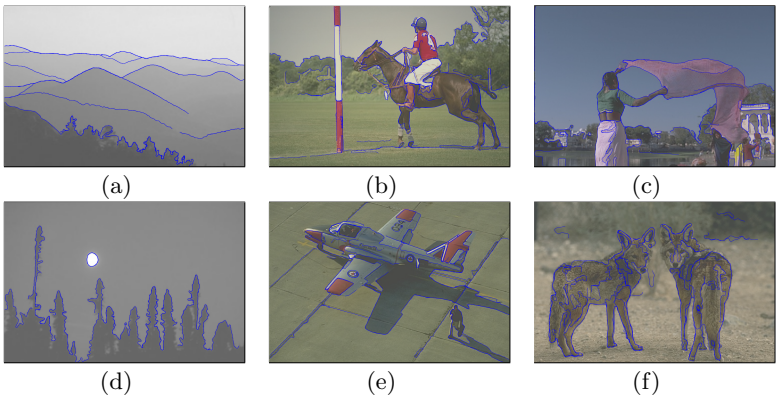


Fig. 6. some results on the BSD Dataset.

In addition, Figure.7 shows some results obtained by using different vector fields. The top row displays the results by only using gray-level feature to construct the vector field and the bottom row shows the results by using multiple features to construct the vector field. From Figure.7, we can see that by using the multiple features (obtained from the gPb detector), we can get more complete and clean results, while by using the single feature (graylevel or color), we are able to obtain more details.

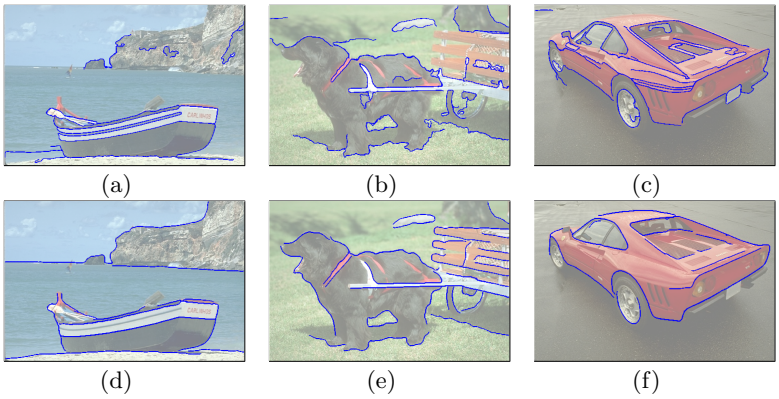


Fig. 7. some comparisons between using single feature (graylevel) and multiple features . Top row displays the results by only using color feature to construct the vector field; bottom row shows the results by using multiple features to construct the vector field.

5 Inference of Illusory Contours

The illusory contours (or subjective contours) are visual illusions that evoke the perception of an edge without brightness or color changes on that edge. Speaking of the illusory contours, the first image comes into our mind would be the Kanizsa's Triangle, shown in Figure.12 (a). We can perceive from the spatially separated fragments in Figure.12 (a) that there are hidden triangles and circles, but in the view of images, there are no complete triangles or circles. Then we cannot help to ask that why we can perceive these unreal outline, can we make the computers sense these phenomena. According to [11], it is believed that for us human beings, the early visual cortical regions such as the primary visual cortex (V1) and secondary visual cortex (V2) are responsible for forming illusory contours.

For the computers, how can they perceive the illusory contours like humans remains our problem. By observing the illusory contours shown in Figure.11 (a) and Figure.12 (a), (d), we can find that the illusory contours are usually an unseen curve that connects two corner points. Here comes the question: is it possible for a point to flow on the illusory contour rather than on the boundary of an object when it meets a corner point? The answer is yes.

5.1 Inertia-Driven PointFlow and Circle Hough Transform

In our opinion, the illusory contours are always small missing parts of regular and predictable shapes. For these kind of illusory contours, it can be completed by an arc with a specific radian. Even for a straight line, it can be described as an arc of a circle of which the radius is infinite. So the main task of inferring the illusory contours is to find an appropriate arc that connects two specific terminals.

In this paper, we assume that the terminals of an illusory contour are always caused by abrupt changes in curvature of a closed curve. These terminals are also called the corner points.

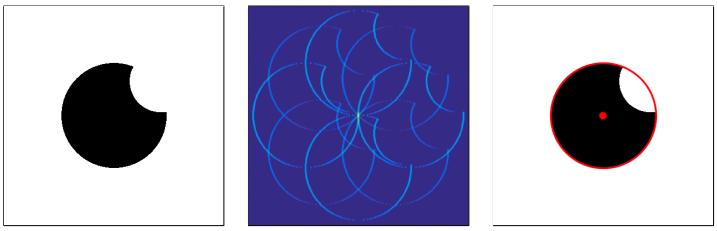
After obtaining the corner points, we make the point flow at the same velocity (angular velocity) $\mathbf{v}$ as it did before it hits the corner point, instead of flowing on the image induced vector field.

Due to the fact that the point flow on a field without any forces after hitting the corner point, the behavior of the point is like inertia-driven, so we call this process "Flow with inertia".

The idea that a missing curve in illusory contour images can be completed by an arc reminds us of the Circle Hough Transform (CHT)[3]. The CHT is a technique for detecting circular objects in images. In a 2D space, a circle of which the center locates at (a, b) with a radius r can be described by:

$$(x-a)^2 + (y-b)^2 = r^2 \tag{5}$$

The parameter space is 3D, (a, b, r). And the corresponding circle parameters can be identified by the intersection of a number of conic surfaces which are caused by the circle on the image.



(a) (b) (c)

Fig. 8. Detection of circles by CHT, from left to right, (a) the original image; (b) projection on (a, b, r) space ($r = 95$); (c) circle detection result, shown in red.

From the example in Figure.8, we can see that the CHT has the ability to infer obvious circles. In the experiment section, we will compare the circle detection results by our inertia-driven flow and the circle hough transform.

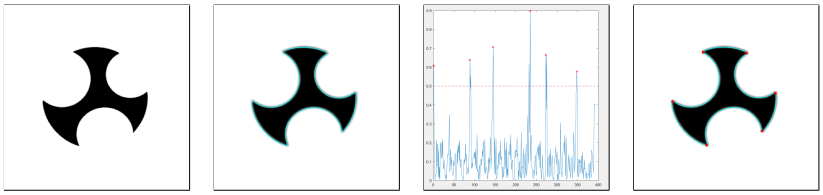
5.2 Corner Points via Flowing

The trajectories formed by the motion of these points are continuous, thus we can detect the corner points by taking the change of curvature into consideration, where the curvature at a point p of a curve l is defined as:

$$k(p) = \left\| \frac{\mathrm{d}A}{\mathrm{d}T} \right\| \tag{6}$$

where A is the angle and $\mathrm{d}A$ stands for the change of angle, T represents time and $\mathrm{d}T$ is the change of time. In this paper, each point driven by the vector field moves in a unit time every step.

A potential corner point may exist at places where there is abrupt change of curvature of a trajectory. It can be determined that whether the potential corner point is a true corner point or not by setting a threshold for the curvature. Figure.9 demonstrates the process of finding corners by taking the curvature into consideration.



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

Fig. 9. An example for detecting the corner points. (a) the original image; (b) the cyan curve is the detection result by using PointFlow method; (c) the plot of curvature of the detected curve; (d) the corresponding corner points.

5.3 Flow with inertia

After finding the corner points, we can infer the illusory contours by deciding that whether two corner points can be connected by an arc. As mentioned above, each movement of a single point p takes a unit time, then after ΔT, there should be $\Delta T + 1$ points on the trajectory. In addition, we are introducing a kind of "inertia" here, which means that when a point flows to a corner point, it will no longer move according to the vector field, but continues to flow according to the trajectory it just passed through.

The following three steps describe how to infer the illusory contours by PointFlow in detail.

1. Compute the angular velocity. For a trajectory L_j which passes through two corner points p_{cor_1} and p_{cor_2} successively, in order to guarantee the accuracy and robustness, N points $\{p_n\}, n = [1, ..., N]$ are chosen on L_j to compute the angular velocity. First, the angular displacement from these N points to corner point p_{cor_2} is computed.

$$\phi(l_n) = \mathrm{acos}\left(\frac{\langle \vec{a}, \vec{b} \rangle}{\| \vec{a} \| \| \vec{b} \|}\right).$$

where l_n is a piece of L_j from p_n to p_{cor_2}, $\vec{a}$ stands for the tangent vector of the curve at the corner point p_{cor_2} and $\vec{b}$ represents the tangent vector

at nth point. A toy model can be found in Figure.10 (a). So the angular velocity for l_n is:

$$\omega(l_n) = \frac{\phi(l_n)}{\text{length}(p_{cor_2}, p_n)} \tag{7}$$

2. Flow with inertia. Based on the assumption that the missing part of the illusory contours can be simulated by an arc, we propose a kind of "inertia" which contains an angular velocity $\omega(l_n)$ and the initial speed $\mathbf{v}_0 = \mathbf{V}(p_{cor_2})$. Now p_{cor_2} is considered as the starting point of the trajectory generated by the inertia. With the inertia, the velocity $\mathbf{v}$ of the point can be obtained by a rotation matrix:

$$\mathbf{v}_t = \mathbf{v}_0 \begin{bmatrix} \cos(\omega(l_n)) & \sin(\omega(l_n)) \\ -\sin(\omega(l_n)) & \cos(\omega(l_n)) \end{bmatrix}^t \tag{8}$$

where t is the time of flowing, and it also represents the number of flowing steps. When $\omega(l_n) = 0$, the inertia will generate a straight line, when $\omega(l_n) \neq 0$, the inertia will generate an arc with corresponding radian.

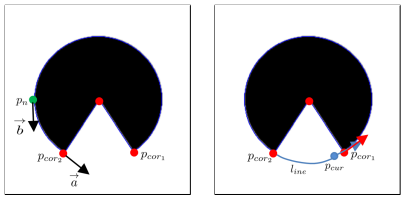
3. Let us denote the trajectory generated by inertia l_{ine}, and the starting point of l_{ine} is the corner point p_{cor_2}. After time ΔT_1, which is a time limit provided by the users, if l_{ine} is still not connected to any corner point, l_{ine} will be ignored. If it is connected to some corner point according to the three conditions below, it is an illusory contour. Decisions will be made at each move of l_{ine}, and the end point of l_{ine} at each step is labeled as p_{cur}. To decide whether l_{ine} can be an illusory contour or not, there are three conditions:
   - The length of l_{ine} is larger than a threshold that we set:
   $$\text{len}(l_{ine}) > \beta$$
   - The distance between p_{cur} and a corner point p_{cor} should be small enough:
   $$\text{len}(p_{cur}, p_{cor}) < \epsilon$$
   - l_{ine} can be connected to p_{cor} in a smooth enough way. Let us take Figure.10 (b) as an example. Let $\vec{c}$ be the tangent vector of l_{ine} at p_{cur} (the blue arrow), $\vec{d}$ the tangent vector at p_{cor_2} (the red arrow). The angle φ between $\vec{c}$ and $\vec{d}$ can be represented by the cosine value: $\varphi = (\frac{\langle \vec{c}, \vec{d} \rangle}{\|\vec{c}\| \|\vec{d}\|})$. If $\|\varphi\| > \delta$, $\delta = 0.9$ empirically, then the direction of the flow l_{ine} is in accordance with L_j.

   If the above three conditions are satisfied, l_{ine} can be considered as the arc which fill in the missing part of the illusory contours.

6 Experiment on the Inference of Illusory Contours

6.1 Data Settings

We test our method for inferring the illusory contours on three illusory images.



(a) (b)

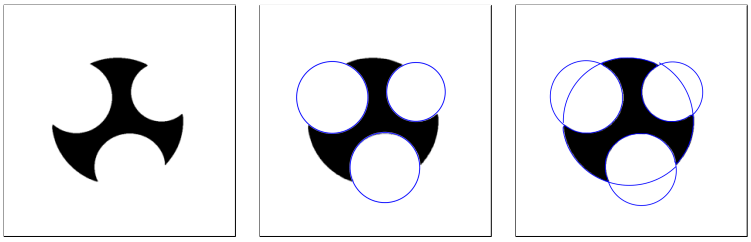
Fig. 10. An example for the flow driven by inertia, (a) describes how the angular displacement is obtained in Step.1; (b) shows how l_{ine} can be an illusory contour.

The first image is shown in Figure.11(a), the size is 400×400. We can perceive that there are 4 circles in this image: a big black one and three small white ones. For inferring the circles, we use twice the Circle Hough Transform (of which the radius ranges from 30 to 100 pixels and 50 to 120 pixels) and our inference method respectively.

The other two images are shown in Figure.12 (a) and (d), the size of both images is 400×400. It can be perceived by users that in Figure.12 (a) there exist a hidden triangle and three small circles and in Figure.12 (d) there are four circles as well as a square.

6.2 Experiment Result and Analysis

Figure.11 (b) and (c) display the circle detection result by using the CHT and PointFlow respectively. From the result (b), we can see that the CHT detect only the small circles. Due to that the big black circle has a lot of missing parts, no matter what radius we use, it cannot be detected. While in (c), our method has inferred and completed all the four circles. By using the CHT to detect circles, users have to provide a range of radii and a very large radius may result in bad results. While inference by PointFlow is automatical and effective.



(a) (b) (c)

Fig. 11. Experiment on a synthetic image, from left to right, (a) the original image (b) the circle detection result by using CHT (c) the inference result by using our method.

From Figure.12(c) and (f), it can be seen that all the illusory parts (either the straight lines of the triangle and the square or the curved arcs of the circles) are inferred by using our inertia-driven PointFlow method.

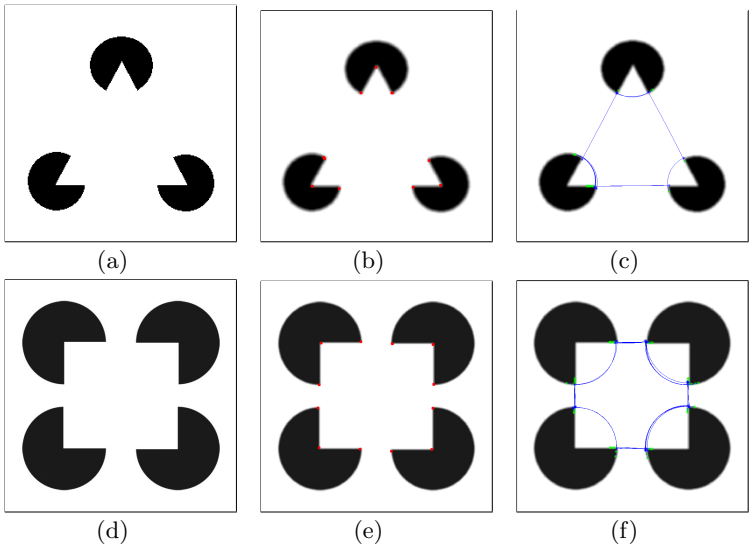


Fig. 12. Experiment on synthetic images, from left to right: (a) and (d) are the original image, (b) and (e) are the corresponding detected corner points, (c) and (f) the inferred illusory contours.

The results in Figure.11 and Figure.12 demonstrate that our inference algorithm is efficient at inferring the straight and smooth arcs. The disadvantage of this method lies in that it will fail when there is an obstacle on an illusory contours, or the illusory contours could no longer be represented by an arc.

7 Conclusion and Future Work

This paper proposes a novel method called PointFlow for automatically modeling the process of tracking object boundaries in images and inferring illusory contours. The PointFlow method is realized by using an ordinary differential equation, where an image-induced vector field is the most crucial factor for boundary detection and integration. In this paper, the vector field can be constructed simply based on the gradient of the graylevel of the image, or via combining multiple features. Compared with the classical edge detectors, the PointFlow achieves a very satisfying result. For the inference of the illusory contours, we have introduced an "inertia-driven" flow, it has an ability to find the missing part of the illusory contours. In future, we would like to adapt the PointFlow method to infer more complicated illusory contours by using higher-level features.

References

1. Pablo Arbelaez, Michael Maire, Charless Fowlkes, and Jitendra Malik. Contour detection and hierarchical image segmentation. *IEEE transactions on pattern analysis and machine intelligence*, 33(5):898–916, 2011.
2. John C Butcher. Numerical methods for ordinary differential equations in the 20th century. *Journal of Computational and Applied Mathematics*, 125(1):1–29, 2000.
3. Richard O Duda and Peter E Hart. Use of the hough transformation to detect lines and curves in pictures. *Communications of the ACM*, 15(1):11–15, 1972.
4. Nawapak Eua-Anant and Lalita Udpa. Boundary detection using simulation of particle motion in a vector image field. *IEEE Transactions on Image Processing*, 8(11):1560–1571, 1999.
5. Ron Kimmel and Alfred M Bruckstein. Regularized laplacian zero crossings as optimal edge integrators. *International Journal of Computer Vision*, 53(3):225–243, 2003.
6. Chenggang Lu, Zheru Chi, Gang Chen, and Dagan Feng. Geometric analysis of particle motion in a vector image field. *Journal of Mathematical Imaging and Vision*, 26(3):301–307, 2006.
7. Vadim Makhervaks, Gill Barequet, and Alfred Bruckstein. Image flows and one-liner graphical image representation. *Annals of the New York Academy of Sciences*, 972(1):10–18, 2002.
8. 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 *Proceeding on 8th Internationall Conference of Computer Vision*, volume 2, pages 416–423, July 2001.
9. David R Martin, Charless C Fowlkes, and Jitendra Malik. Learning to detect natural image boundaries using local brightness, color, and texture cues. *IEEE transactions on pattern analysis and machine intelligence*, 26(5):530–549, 2004.
10. Yossi Rubner and Carlo Tomasi. Coalescing texture descriptors. In *ARPA Image Understanding Workshop*, pages 927–935, 1996.
11. R Von Der Heyclt, E Peterhans, and G Baurngartner. Illusory contours and cortical neuron responses. *Science*, 224, 1984.
12. Djemel Ziou and Salvatore Tabbone. Edge detection techniques - an overview. *International Journal of Pattern Recognition and Image Analysis*, 8:537–559, 1998.