Active Contour Models

By

Taein Lee

A PROJECT

submitted to

Oregon State University

in partial fulfillment of
The requirements for the
Degree of

Master of Science in Computer Science

Presented September 29, 2005
Commencement June 2006

Abstract

Active contour models have been widely applied to image segmentation and analysis. It has been successfully used in contour detection for object recognition, computer vision, computer graphics, and biomedical image processing such as X-ray, MRI and Ultrasound images.

The energy-minimizing active contour models or snakes were developed by Kass, Witkin and Terzopoulos in 1987. Snakes are curves defined in the image domain that can move under the influence of internal forces within the curve itself and external forces derived from the image data. Snakes perform well on certain types of images (such as well-defined, convex shapes). There have been several improvements proposed to the original snake or active contour model. These improvements include balloon snakes, adaptive snakes, and GVF snakes. In this project, I reviewed and implemented their algorithms as well as the original snake model.

GCBAC (Graph Cut Based Active Contour) is one of alternative solutions to the object extraction problem. Although the GCBAC belongs to family of active contour models, it differs fundamentally from original active contours. In this project, I also review and implement the GCBAC algorithm as well.

Table of Contents

List of Figures

1. Introduction

Image segmentation is a process in which regions or features sharing similar characteristics are identified and grouped together. Segmentation may use statistical classification, threshold, edge detection, region detection, or any combination of these techniques to solve segmentation problems. The techniques includes region-based, threshold-based, edge-based or connectivity-based [20].

Region-based techniques rely on common patterns in intensity values within a cluster of neighboring pixels. The cluster is referred to as the region, and the goal of the segmentation algorithm is to group regions according to their anatomical or functional roles. The threshold-based techniques rely on local pixel information. It is effective only if the intensity levels of the objects fall squarely outside the range of levels in the background. Since the spatial information of an image is ignored, it became problemic at blurred region boundaries. Edge-based techniques rely on discontinuities in image values between distinct regions, and the goal of the segmentation algorithm is to accurately detect the boundary separating these regions. Connectivity-based techniques rely on a curve known as active contour formed by several control points on the image. The active contour, then, deform itself by the energies derived from an image and locations of control points. The goal of the segmentation algorithm is to navigate an active contour toward the boundary of a target object in an image. In this paper, I set focus on active contour models derived from Connectivity-based techniques. I review some of methodologies and their approaches to the image segmentation problems.

Active contour models have been widely applied to image segmentation and analysis. It has been successfully used in contour detection for object recognitions, computer vision, computer graphics, and biomedical images processing such as X-ray, MRI and Ultrasound images. The primary purpose of the active contour models is to locate boundary of an object on an image by using an active contour. The active contour is navigated by the defined energy functions. Several energy functions have been proposed and experimented on active contour models to achieve the objective.

One of the active contour models, known as “snake”, was originally developed by Kass, Witkin and Terzopoulos [1]. The snake is described as “an energy minimizing spline guided by external constraint forces and influenced by image forces that pull it toward features such as lines and edges.” Although the original snake has several limitations on its performance, the concept of the snake has been used to create bare-bone of the successors of the snake. Several new ideas have been proposed and added to the concept of the original snake. In this project, I review some of their methodologies as well as the original snake.

Adaptive snakes, discussed in the section 3, were proposed by Choi, Lam and Siu [5]. A new energy term called “Adaptive forces” is added into the energy function of the original snake. The adaptive forces are used to overcome the problem of detecting concavities in the object.

Pressure snakes or Balloon snakes discussed in the section 4 were proposed by Cohen [7]. A new energy term known as “Balloon force” is added to the energy function. The energy functions of the traditional snake is designed an active contour to converge toward the boundary of the object. Balloon force adds capabilities of inflating/deflating to an active contour by assigning positive or negative signs to the control points of the active contour.

GVF snakes, discussed in the section 5, were proposed by Xu and Prince [9]. A new external force known as “Gradient Vector Field” is introduced. Similar to the external force of the traditional snake, the gradient vector field is derived from intensities of an image. However, it fundamentally differs from the traditional external force. It adds some sort of energies in homogeneous region on an image where there is no energy in general.

In the section 6, I review one of alternative solutions to the object extraction problem. It is called “GCBAC” (Graph Cut Based Active Contour) proposed by Xu, Bansal and Ahuja [13]. The GCBAC belongs to family of active contour models; however, it differs fundamentally from the original snake and its successors. Unlike the metrologies of snake family, it does not use a non-parametric curve as an active contour. First, the GCBAC translates an image to an adjacency graph. An active contour is formed by cutting theses edges in the graph.

Lastly I review the extraction of multiple objects in an image. In general, the snake is designed for detecting a contour of a single object in an image. I discussed the one of algorithms proposed by Choi, Lam and Siu [5] to detect contours of multiple objects in an image in Section 7.

2. Original Snake Active Contour Model

The original snake was developed by Kass Witkin and Terzopoulos in 1987 [1]. The name “snake” was named after its behavior on an image. While minimizing their energy, it slithers on the image.

A snake is expressed as a planar parametric curve in Eq. (2-1). The parameter **s** is snake control points known as snaxels. These snaxels are linked together to form an active contour (Fig.2-1). The snake is not a method to automatically detect the boundary of the desired object in an image. It requires appropriate parameters setting and initial

locations of the snaxels according to the subjective boundary. Therefore, some prior knowledge about the image is required from higher level system.

$$v(s) = [x(s), y(s)] \quad s \in [0,1] \tag{2-1}$$

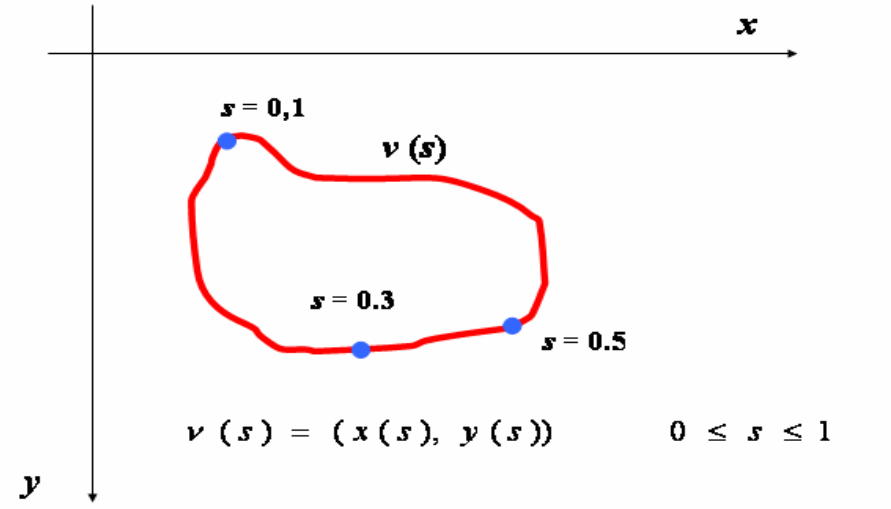


Fig. 2-1 Parametric curve

Equation (2-1) represents the energy function of the snake. As the snake is defined as an energy minimization spline [1], it deforms itself to minimize the energy. The energy function is designed for the snake to converge toward the boundary of the target. It behaves similar to a rubber band placed outside of the object and shirks to reach the boundary of the target.

The first two terms in the energy function of the snake in Eq. (2-2) are called *internal energy*. The tension and rigidity of the snake are controlled by these forces. The third term is called *external energy*. It is derived by the image and it attracts the snake to the target contour.

$$E_{snake} = \int_0^1 (\alpha \cdot E_{elastic}(v(s)) + \beta \cdot E_{bending}(v(s)) + \gamma \cdot E_{image}(v(s)))ds \tag{2-2}$$

2-1 Internal Energy

The internal energy is further divided into two energy components: elasticity and bending forces. The internal energy is the sum of these forces and the energy function is expressed as:

$$E\,interal = \left[\alpha(s) \cdot \left| \frac{dv(s)}{ds} \right|^2 + \beta(s) \cdot \left| \frac{dv(s)^2}{ds^2} \right|^2 \right] / 2 \qquad \textbf{(2.1-1)}$$

The first and second derivatives of the contour represent these energy terms and called "Elastic forces" and "Bending forces", respectively. The elastic force controls the tension of the snake. It discourages stretch of the active contour and it is responsible for shrinking the contour (*red arrow* in Fig.2-1).

The bending force is defined as "The bending energy makes snake acts like a thin plate" [1]. It controls the rigidity of the snake. It controls only the curvature, not the length of the contour. During the deformation process, it tries to be a smooth curve or straight line (*blue-dot arrow* in Fig.2.1-1).

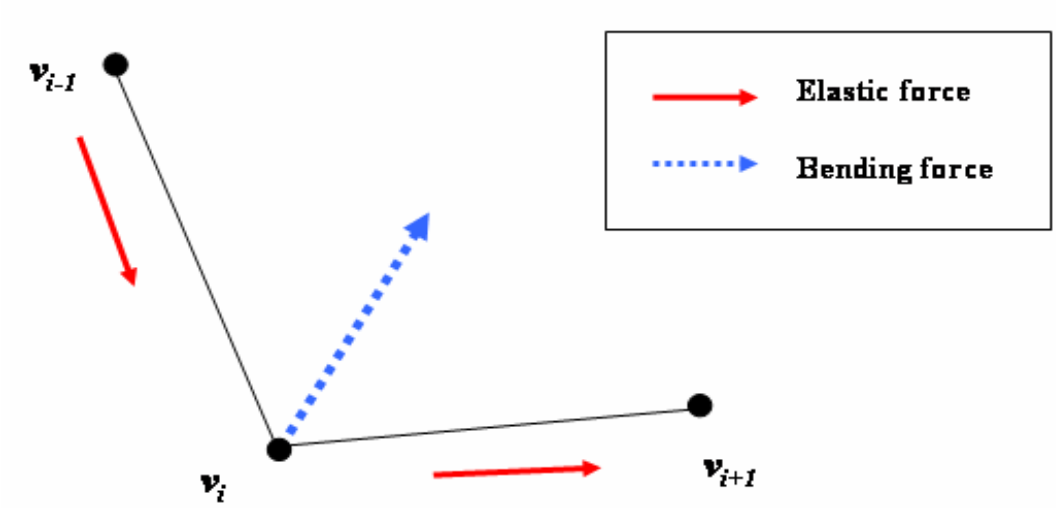


Fig. 2.1-1 Internal energy

The parameters α(s) and **β**(s) in front of each term represents weighting functions. In general, values of these weighting functions are constants for all snaxels. Selecting an appropriate set of these constants creates one of difficulties of the snake. They have large impact in snake's behaviors and totally control the performance of deformation process. Each object in an image requires different set of constants value for snake to perform well. The one way to solve this problem is to make the snake dynamically change these values to suitable values during deformation process. However, it requires a computer to recognize shapes or topologies of an object in an image automatically. Therefore, the solution is left for further improvement of the snake. Currently, these

parameters are up for a user to select at the initialization process.

2-2 External Energy

The external energy is derived from the image data. This image-driven force attracts the snake to move toward the target contour. The energy term is defined in the following equations [1].

$$E_{external}(s) = -\gamma(s) \cdot \left| \nabla(I(s)) \right|^2 \qquad \textbf{(2.2-1)}$$

$$E_{external}(s) = -\gamma(s) \cdot \left| \nabla(G_\sigma(s) * I(s)) \right|^2 \qquad \textbf{(2.2-2)}$$

∇ represents gradient operator and $\boldsymbol{I}(\boldsymbol{s})$ is the intensity of the image at **s**. $G_\sigma(s)$ is two-dimensional Gaussian function with standard deviation **σ**. The weighting function γ (s) is used to control the image force.

Gaussian filter is applied on the original image to increase the capture range of the snake in Eq. (2.2-2). Gaussian filter makes an image blurry. Figure 2.2-1 illustrates an example of blurring a part of an edge. The image force diffuses from the edge (*blue-dot line*) to the edge (*red lines*). As a result, it widens the capture range of the snake. Two snapshots of the application in Fig. 2.2-2 show the comparison of image forces generated with non-filter and Gaussian filter. The *red arrows* in the pictures point to the direction of image forces. The red dots represent the non image force. Therefore, they are seen on homogeneous area where no image force exists

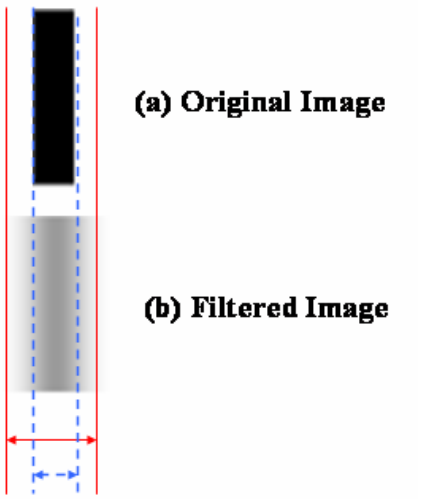


Fig.2.2-1 Image Forces

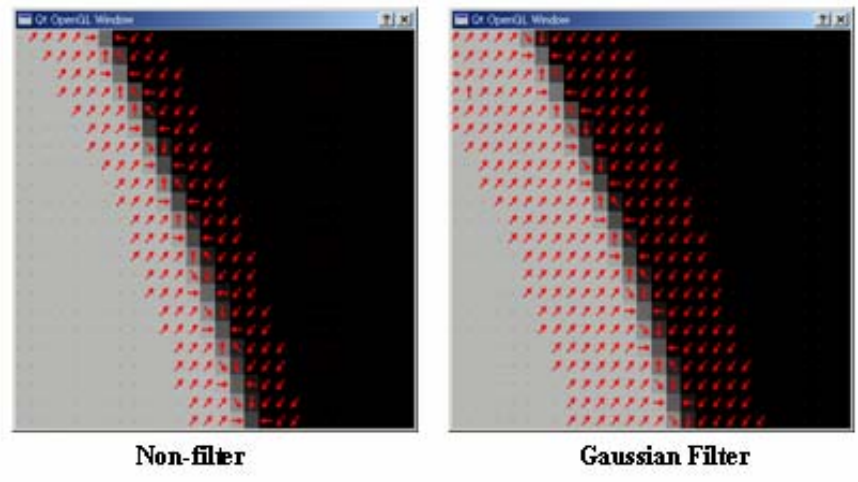


Fig. 2.2-2 Image forces with non-filter and Gaussian filter

In general, Eq. (2.2-2) is used to compute external force since square of gradient magnitude of intensity itself has smaller capture range. The standard deviation **σ** plays a key role on capture range of the snake. The larger **σ** causes the boundaries of objects become blurry. This is sometimes necessary to help an active contour to move toward to the desired boundary.

2-3 Algorithms of Snakes

The most of algorithms of the active contour models are consists of the following three phases. The snakes are not an exception.

Algorithms of the active contour models
in higher-level of view

- Initialization process
- Deformation process
- Termination of snake

Initialization process: In the initializing process, a user sets the initial locations of the snaxels around the boundary of the target object. At the same time, the set of weighting parameters, α, β and γ are chosen. Afterward, it starts to deform itself toward the true boundaries of the object. The initial contour must be close to the subject boundary since the snake could move toward noises or other undesired edges or lines on an image if it is placed far from the true boundary.

Deformation process: During deformation process, it deforms itself to minimize the sum of the energy terms defined in Eq. (2-2). For each iteration in this process, a new location is searched among neighboring pixels for each snaxel. A snaxel moves to a pixel that has lower energy or it stays in the same location if there is none. There are two approaches to compute new locations for snaxels. The greedy algorithm is one of the methods to find a new location for each snaxel. The other way is to use the technique of the dynamic programming. The greedy algorithm finds a locally optimal solution meanwhile the dynamic programming finds a globally optimal solution. The implementation details of these methods are discussed in Section 9.4 and 9.5. The deformation process continues until the snake is caught in one of the terminating criteria. The terminating criteria are discussed in the following section.

Termination of the snake: At the some point of the time, we need to stop the deformation of the snake. Naturally the snake ceases its deformation when all the snaxels can not find new locations in the neighboring pixels, or simply it converges to zero and disappears from the sight. However, it could go into an infinity loop when the snaxels shift along the boundary or some of snaxels oscillate. Therefore we need set a certain criterion for the termination of the snake. The simplest way is to set a threshold on the maximum number of iterations executed in the deformation process. This guarantees that the snake is terminated and it never goes into an infinity loop. It, however, requires a user to set the appropriate the number of iterations prior to the deformation process. Since the number of iterations varies depends on the shape and the size of the target object, it is difficult to estimate how many time of iterations are needed to detect the subject contour. As a result, the snake would be terminated before it reaches or close to the true boundary when the small number of the maximum iterations was set. Another way is to set threshold on the number of snaxels moved in between iterations. For instance, we could set the threshold as *90%* to the number of snaxels. Then the snake terminates its deformation when *90%* of snaxels are not moved. In this way, the threshold does not depend on the shape or the size of the target object. Consequently, it does not need to require a user to changing the number of iterations on each target object. However, neither terminating criteria would be useful because it is, sometimes, observed the snaxels shift along the boundary, and the contour only moves very slightly [5]. Wong, Yuen and Tong [6] proposed other solution for the termination of the snake. It is called contour length criterion, “CL-criterion“. The CL-criterion is based on the length of the active contour. It keeps track of the last *10* iterations and it

observes the changing ratio of the length of the active contour. *10* iterations are needed. This is due to the sudden increase and decrease of the rate of change of normalized total length may cause the spikes and valley during the iterations [5]. Choi, Lam and Sui [5] proposed similar terminating criterion, called contour area criterion, "CA-criterion", which makes use of the normalized total are to determine the convergence of the process. They claim that the CA-criterion has better stability in convergence, as its fluctuation between successive iterations is much smaller than the CL-criterion. Equation (2.3-1) below is used to compute the ratio of CA-criterion at the k^{th} iteration.

$$ratio\ \eta^k = \frac{A^k - A^{k-1}}{A^k} \qquad \textbf{(2.3-1)}$$

Equation (2.3-2) is used to calculate an area of the active contour. It is base on the Green's Theorem in a plane. It makes CA-criterion works faster since it does not require square roots to obtain the lengths of the segments between all the snaxels.

$$\begin{aligned} Area\ of\ \ polygon &= \frac{1}{2} \cdot \sum_{i=1}^{n} \begin{vmatrix} x_i^k & y_i^k \\ x_{i+1}^k & y_{i+1}^k \end{vmatrix} \\ &= \frac{1}{2} \cdot \sum_{i=1}^{n} (x_i^k \cdot y_{i+1}^k - x_{i+1}^k \cdot y_i^k) \end{aligned} \qquad \textbf{(2.3-2)}$$

where ***n*** is the total number of the snaxels and ***k*** is at the ***k*th** iteration.

Table 2.3-1 shows the result of the performance of the snake terminated by the CL- and CA-criterion on the three sample images.

	Penguins		B-2		Pochacco	
	CL	CA	CL	CA	CL	CA
Number of iterations	75	72	47	43	41	39
Runtime (ms)	118.5	111.6	56.4	50.8	68.5	63.6
Number of snake points	142	142	109	110	146	144

Table 2.3-1 Comparison with CL- and CA-criterion on three images (The experience is conducted on a Pentium-II 400 MHz PC) [5]

2-4 Limitations of the original snake

The original snake works well on certain images. However, there are several limitations on this method. The one of major problems comes from its poor capture range. Although Gaussian filter is applied on the original image, the external forces derived from boundaries are sometimes not strong enough or wide enough to attract the snake toward true boundary of the target object. Accordingly, an initial contour must be set close to the boundary of the desired object. If the snake was initialized too far away from edges of the target object, it will either move randomly, due to lack of gradient force, or it will be attracted to noises or its other near by edges, lines or points.

Another major problem is that the snake can not progress into the concavities of an object. Since there is no gradient force in the area under *white-dot-circle* on Fig. 2.4-1, the snake totally depends on an internal energy. The internal forces of the snaxels under the region, however, try to pull and straighten the snake. Consequently, the snake cannot move down to the concavity of the object. The red line in Fig. 2.4-1 shows the result of the snake performed on the U-Shape object. You can see the snaxels under the concavity at the top of U-shape object ceases its movement.

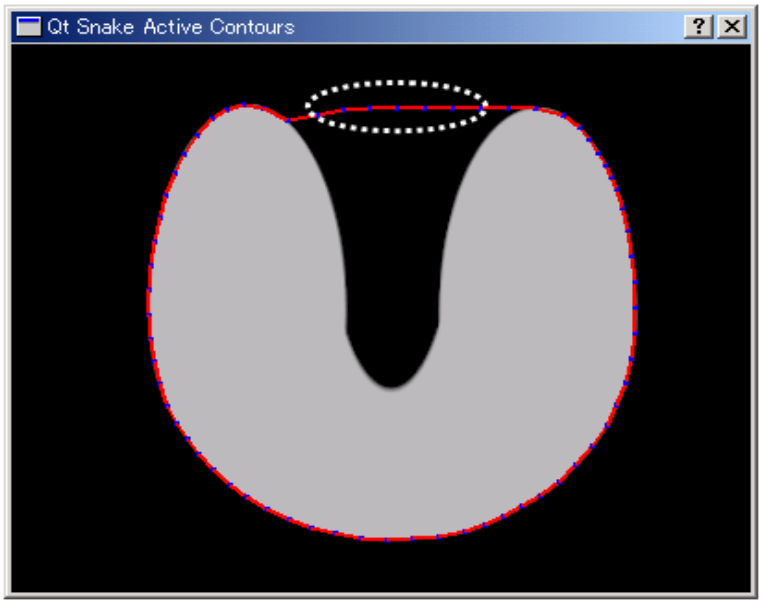


Fig. 2.4-1 The concavity problem of the snake on the U-shape image

3. Adaptive Snakes

Adaptive snakes proposed by Choi, Lam and Siu [5]. A new energy term called "Adaptive forces" was introduced. The adaptive force is added to the energy function of

the snake in Eq. (2-2). The force will be introduced at a snaxel if its surrounding image forces are smaller than a threshold [5]. The direction of an adaptive force is perpendicular to a line v_{i+1} and v_{i-1}, which is a normal vector n in Fig. 3-1.

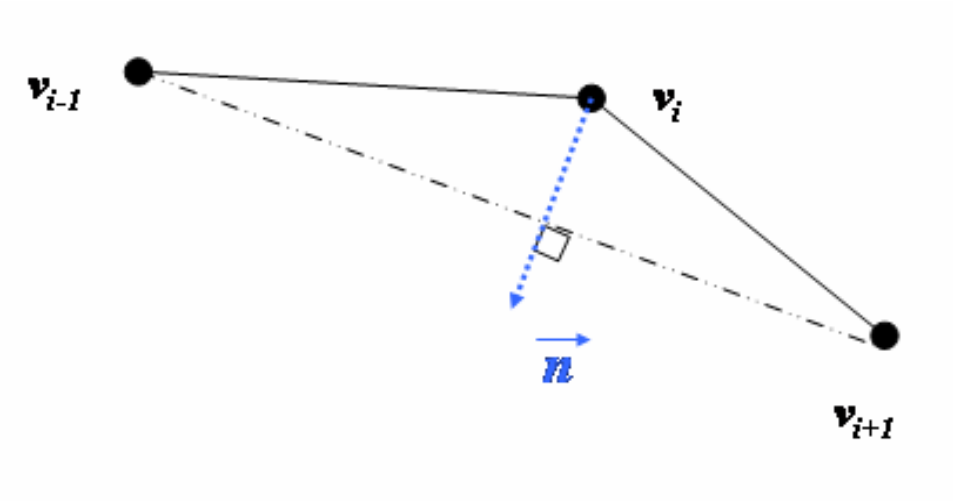


Fig. 3-1 Adaptive force

$$v_i^{'} = v_i + k \cdot \vec{n} \qquad \textbf{(3-1)}$$

where k is the amplitude of an adaptive force

Equation (3-1) is used to compute a new location of i^{th} snaxel. An adaptive force, $k \cdot n$, will be add to a snaxel if image forces of the neighboring pixels are low. In the algorithm of the traditional snake, a snaxel moves to a neighboring location, which is one pixel away from the current location. In contract, Adaptive snake steps more than one pixel when k is set to more than 1. Figure 3-2 shows the movement of the active contour with the step size of *3* and *5*. Therefore, it makes the snake converge faster in the region where the image forces are weak. Furthermore, it helps the snake to step over certain noises on an image. This is because noises in general are low image forces. However, the larger step size k sometimes creates a problem of making the distances of two adjacent snaxels uneven. Figure 3-3 shows two examples of creating the problems. Consequently, two operations, deletions and insertions, are necessary to keep the distances of the adjacent snaxels more constant. A new snaxel is inserted when two adjacent snaxels are far from each other. Meanwhile, the current snaxel is deleted if the distance between the next and the previous snaxels are small.

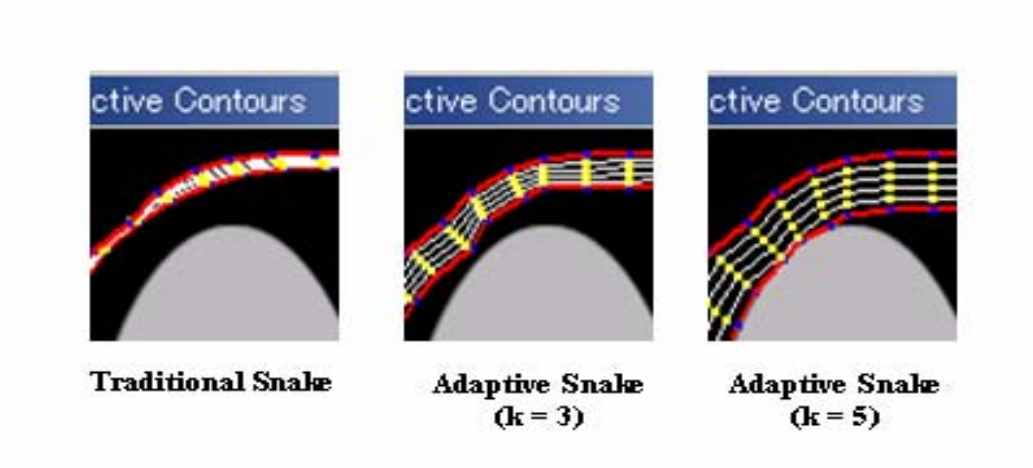


Fig. 3-2 Step sizes of the snake and Adaptive snake

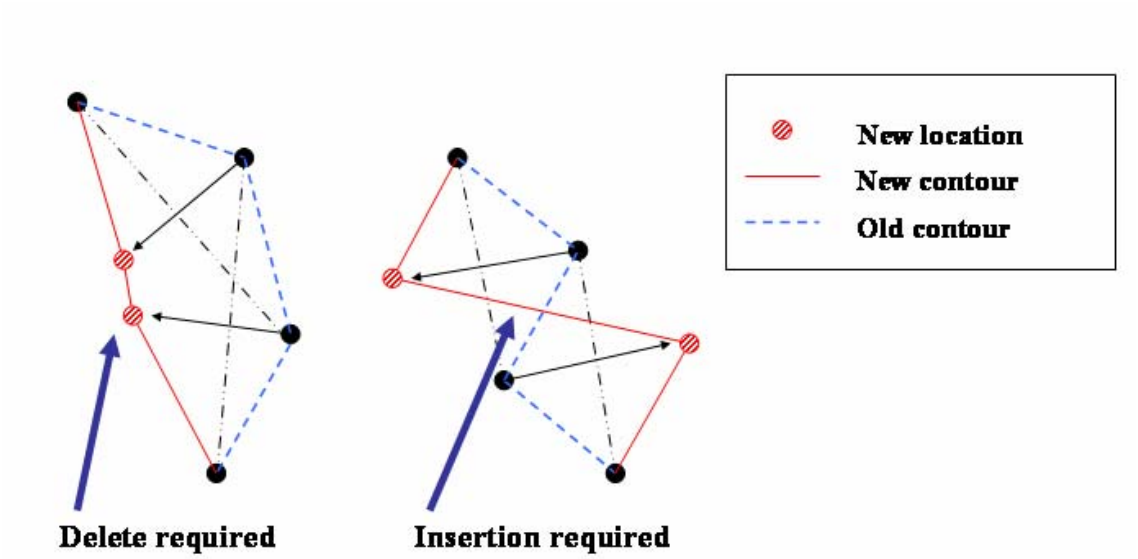


Fig. 3-3 Creating problems of uneven spaces

3.1 Discussion of Adaptive Snakes

Unlike the traditional snake, the adaptive snake can go into the concavities of the object. Figure 3.1-1 shows partial images of Adaptive snake detecting the boundary of the object with a concavity. The complete gif-animations of Adaptive snake are shown in Section 10.3.

The adaptive snake converges faster when the larger step size is chosen. It takes *49* and *90* iterations to detect the boundary of the U-shape object when the adaptive force ***k*** is set to be 3 and 5 respectively. Meanwhile, the larger step size creates another problem.

As I mention in the previous section, the adaptive forces can step over some noises or undesired edges or lines; however, it may step over the true boundary as well. Figure 3.1-2 shows the result of the adaptive snake with step size of 9. Once the active contour get inside of the object, it is difficult to reach back to the subjective contour since the energy function of the snake is designed for an active contour to converge. Therefore, it needs some sort of forces that inflate/diverge an active contour. In the next section, another active contour model that utilizes such forces is discussed.

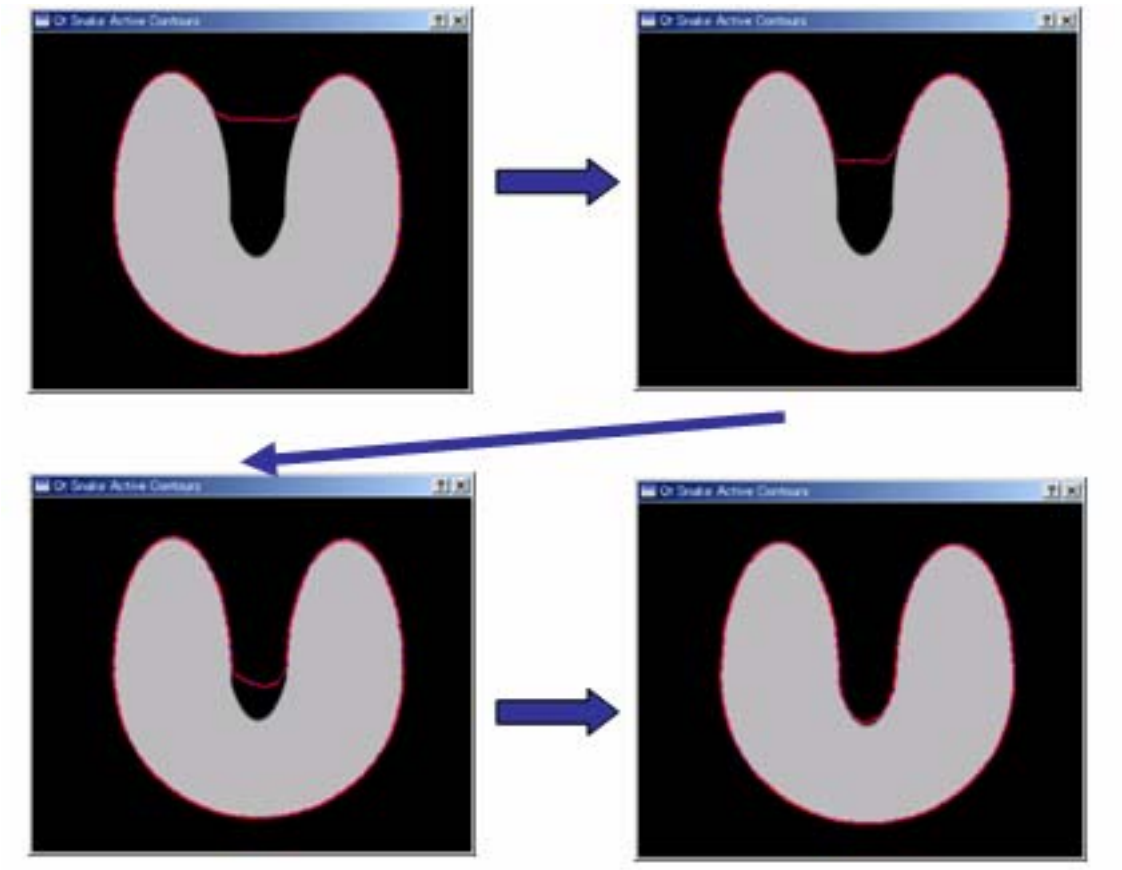


Fig. 3.1-1 Detect boundary at concave region (***k*** = 3)

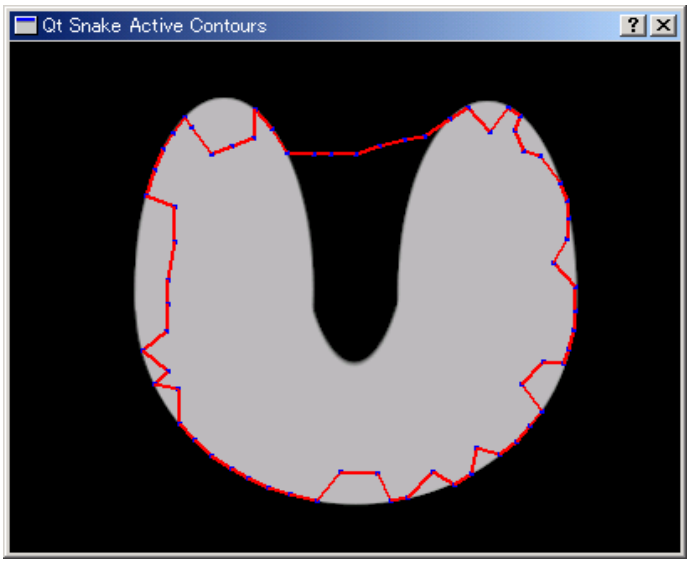


Fig. 3.1-2 Problem of the larger step size of the adaptive snake (***k*** = 9)

4. Pressure or Balloon Snakes

The concept of Pressure snakes or Balloon snakes was introduced by D. Cohen [7]. The proposed force is replaced with the external force of the original snake. The new external force is defined as:

$$E_{\boldsymbol{balloon}}(\boldsymbol{v}(s)) = \boldsymbol{k} \cdot \vec{\boldsymbol{n}} \perp (\boldsymbol{v}(s)) - \left[\boldsymbol{P} \cdot \frac{\nabla \boldsymbol{f}}{\|\nabla \boldsymbol{f}\|} \right]^2 \qquad \textbf{(4-1)}$$

P is a new constant for the gradient magnitude and ***k*** is the amplitude of a balloon force. $\vec{\boldsymbol{n}}$ is a normal unitary vector at ***v***(*s*). The normal unitary vector is the same normal vector as illustrated in Fig. 3-1. The sign of ***k*** controls the direction of the balloon snake. The balloon snake inflates when the sign of the ***k*** is positive and deflates when it is negative. It is necessary to set ***P*** slightly larger than ***k*** so that an edge point can stop the inflation forces. The inflations of the balloon forces solve the problem of the active contour lays inside of the subject boundary. The *blue* lines in Fig.4-1 are the active contour inflating toward the target boundary by the balloon forces. Meanwhile, the *red-dot* lines illustrate the active contour deflating toward the boundary.

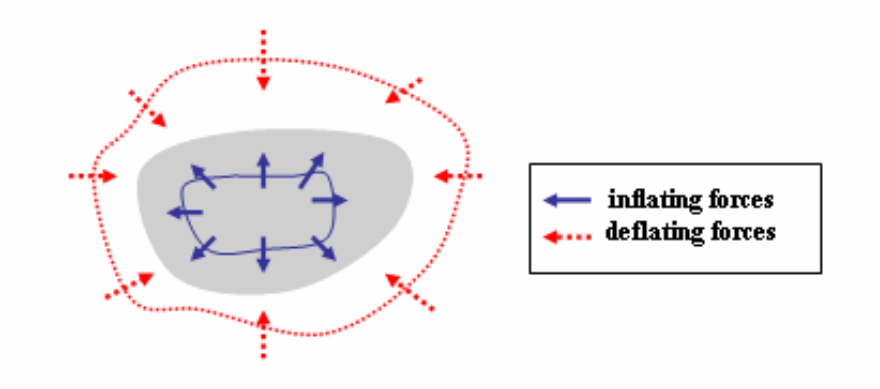


Fig. 4-1 Inflating/deflating balloon forces

4.1 Discussion of Balloon snakes

The balloon snake is more robust to the initial position than the original snake. The original snake is designed an active contour to shrink to reach the target boundary. Therefore, an active contour needs to be initialized the outside of the target contour. It would not reach the target boundary if the active contour lies inside of the target contour. In the balloon model, the initial position can lie either inside or outside the target contour. This is because the direction of the active contour can be control by a user. Although the requirement of an initial location of an active contour are loosen, human intervention is needed to decide whether an inflationary or deflationary forces on an active contour before the energy minimizing process starts. This indicates that it requires a user to have some prior knowledge of the shape of the target object as well as the relative position of the initial location of the active contour and the target boundary.

The balloon snake has added the flexibility of the initialization of the active contour. However, it still has not solved the problems of poor capture range. The initial contour must be close to the true boundary of the target. GVF snakes discussed in the next section can solve the problem of poor capture range of the snake models.

5. GVF (Gradient Vector Flow) Snakes

Several new energy terms are added to the energy function of the tradition snake to improve the performance of the snake models. Nevertheless, the problems associated with detecting concavities region of the target and initializing an active contour have not been solved yet. This is all causes by the weak capture range of the snake. To overcome these problems, it was necessary to change the concept of the external force. In this section, I review the GVF snake developed by Xu and Prince [9].

5.1 Gradient Vector Field

Xu and Prince introduced a new external force to solve these problems. The proposed energy force is derived from a vector fields as known as Gradient Vector Field [9]. Instead of using the gradient magnitude of an image as an external force, it uses spatial diffusion of the gradient of an edge map of an image [8]. The GVF snake involves a vector field derived by solving a vector diffusion equation that diffuses the gradient magnitude obtained form an image. They define the gradient vector field and the energy function as:

$$\boldsymbol{V}(\boldsymbol{x},\boldsymbol{y}) = \left[\boldsymbol{u}(\boldsymbol{x},\boldsymbol{y}),\boldsymbol{v}(\boldsymbol{x},\boldsymbol{y})\right] \qquad \textbf{(5.1-1)}$$

$$\mathrm{E}_{GVF} = \iint \boldsymbol{\mu}\cdot(\boldsymbol{u}_x^2 + \boldsymbol{u}_y^2 + \boldsymbol{v}_x^2 + \boldsymbol{v}_y^2) + \left|\nabla \boldsymbol{f}\right|^2 \cdot \left|\boldsymbol{V} - \nabla \boldsymbol{f}\right|^2 \boldsymbol{dxdy} \qquad \textbf{(5.1-2)}$$

The parameter $\boldsymbol{\mu}$ is a regularization parameter governing the tradeoff between the first and second terms. The value of the $\boldsymbol{\mu}$ is set according to the amount of noise present in the image. It is increased when more noises present in the image [9].

The second term in the integral in Eq. (5.1-2) is called "*data term*". , we can see the data term dominates the energy when $|\nabla \mathrm{f}|$ is large. To minimize the energy, we can set $\boldsymbol{V}$ nearly equal to $|\nabla \mathrm{f}|$. Meanwhile, when $|\nabla \mathrm{f}|$ is small, the energy is dominated by sum of the square of the partial derivatives of the vector field. The first term in the integral is call "*smoothing term*" that makes vector flow $\boldsymbol{V}(\boldsymbol{x}, \boldsymbol{y})$ varying smoothly. In the external force of the traditional snake, no forces are found in the homogenous regions where the gradients are nearly zero whereas it creates some forces in these regions. This helps to navigate the snake to move into the concavities of the subjective object.

5.2 Generalized GVF Snakes

Although the GVF snake performs better than the original snake at boundary concavities, it has some difficulties when the boundary concavities are long and thin [18]. To remedy the problem, Xu and Prince [12] proposed the generalized GVF snake. The both $\boldsymbol{\mu}$ and $|\nabla \mathrm{f}|^2$ in Eq. (5.1-2) are replaced with more general weighting functions; $\boldsymbol{g}(|\nabla \boldsymbol{f}|)$ and $\boldsymbol{h}(|\nabla \boldsymbol{f}|)$. The equation of the proposed GVF field is defined as:

$$\boldsymbol{V} = \boldsymbol{g}(\left|\nabla \boldsymbol{f}\right|)\nabla^2 \cdot \boldsymbol{V} - \boldsymbol{h}(\left|\nabla \boldsymbol{f}\right|)(\boldsymbol{V} - \nabla \boldsymbol{f}) \qquad \textbf{(5.2-1)}$$

The weighing functions $g(|\nabla f|)$ and $h(|\nabla f|)$ applied to the smoothing and data terms, respectively. Eq. (5.2-1) is more general form of the equation (5.1-2). Suppose that the weighing functions $g(|\nabla f|)$ and $h(|\nabla f|)$ are defined as Eq. (5.2-2).

$$\begin{aligned} g(|\nabla f|) &= \mu \\ h(|\nabla f|) &= |\nabla f|^2 \end{aligned} \qquad \textbf{(5.2-2)}$$

Substituting the weighting functions above with Eq (5.2-1), the equation becomes exactly the same equation of the original GVF snake defined in Eq. (5.1-2). Since $g(|\nabla f|)$ is constant, smoothing take place everywhere. Meanwhile, $h(|\nabla f|)$ becomes larger near strong the edge. As a result, it dominates at boundaries. Therefore, the GVF snake should provide good edges localization [12]. However, it becomes problematic when two edges are close, such as when there is a long and thin indentation along the boundary. This is because the GVF snake tends to smooth between opposite edges, losing the forces necessary to attract the active contour move into the region. To overcome this problem, we need to select the weighting functions such that $g(|\nabla f|)$ gets smaller as $h(|\nabla f|)$ becomes larger. There are many ways to specify pairs of the weighing functions. Xu and Prince suggest the following weighting functions for the generalized GVF snake.

$$\begin{aligned} g(|\nabla f|) &= e^{-(|\nabla f|/K)} \\ h(|\nabla f|) &= 1 - g(|\nabla f|) \end{aligned} \qquad \textbf{(5.2-3)}$$

They claim that the weighing functions above will conform to the edge map gradient at strong edges, but will vary smoothly away from the boundaries. The parameter K controls the degree of tradeoff between field smoothness and gradient conformity.

5.3 Discussion of GVF Snakes

Figure 5.3-1 shows the comparison of external forces obtained by the original snake and the GVF snake. From the figures, it is obvious that the external forces derived from the GVF snake has much larger capture range than the original snake. In the homogenous regions of the U-Shape object, there is no external force found in the original snake. Meanwhile, there exist external forces that would lead an active contour to the boundary of the object in the GVF snake.

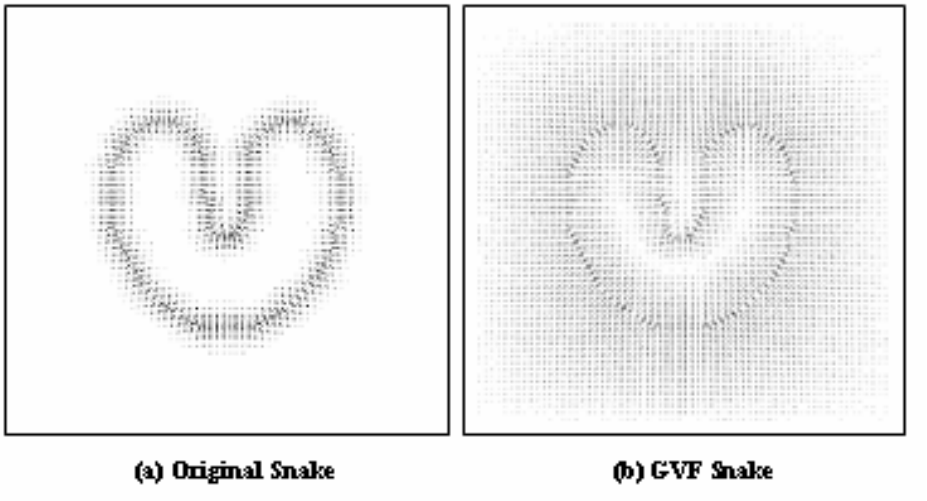


Fig. 5.3-1 External forces of the original snake and GVF snake

Images taken from [9]

The *red-dot circle* on the left in Fig. 5.2-1 shows the external forces at the concavity regions of the U-Shape object. The two figures form right are the close-up images of force vectors derived from the original snake (a) and the GVF snake (b). We can see in Fig.5.2-2 (a) that there is no force in the regions under the *blue-dot circle*. This is why the original snake cannot move into the concavity region. Meanwhile, there exist the forces under the *blue-dot circle* in Fig.5.2-2 (b). In addition to that, the directions of the force vectors in the regions are pointing downward. These force vectors make the active contour move down to the concavity regions of the object.

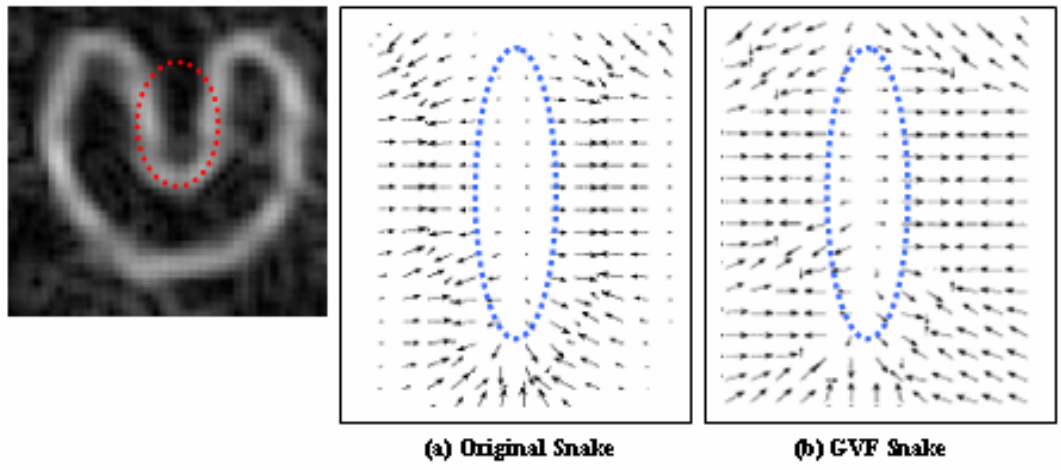


Fig. 5.3-2 External forces at concavity regions of U-Shape object

Images taken from [9]

6. GCBAC (Graph Cut Based Active Contour)

The concept of the GCBAC (Graph Cut based Active Contour) is proposed by Xu, Bansal and Ahuja [13]. It is other alternative way to extract an object boundary on an image with an active contour. Unlike the traditional snake or its successors, an active contour represented in the GCBAC is a non-parametric curve. It differs fundamentally from the traditional active contour models. The GCBAC constructs an adjacency graph from an image. Each pixel on the image is represented as vertex of the graph. The weights of edges between vertices are assigned according to the value of the gradient of an image. With the adjacency graph, the GCBAC enable to extract a subjective boundary on the image. The implementation details are discussed in Section 12.

6.1 Algorithm of GCBAC

The basic idea of the GCBAC is to find global minimum within CN (Contour Neighbor). CN is defined as a belt-shaped neighborhood region around a contour [13]. The *yellow belt-shape* in Fig. 6.1-1 shows CN with the width of 10. The width of a CN can be specified by a user. During the deformation process, a CN is generated from the current active contour. Within the CN, it creates a multi-source and multi-sink minimum-cut problem. The vertices on the inner boundary of the CN are identified as multi-source and similarly, the vertices on the outer boundary are identified as multi-sink.

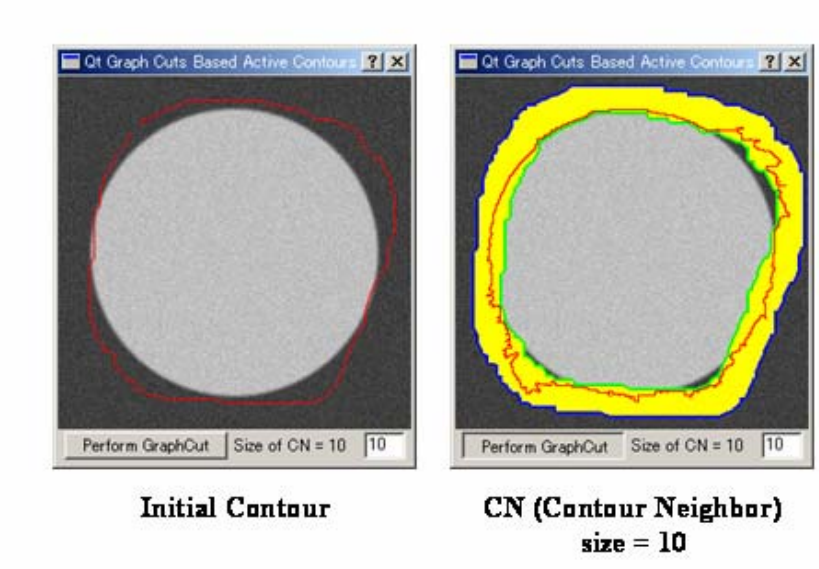


Fig. 6.1-1 CN (Contour Neighbor) of an active contour (size = 10)

The multi-source and multi-sink minimum-cut problem, then, is reduced to a one s-t minimum-cut problem in the graph theory. There are several algorithms available to solve the s-t minimum-cut problem. Such algorithms are Augmenting Path method by Ford-Fulkerson [14], Push-relabeled method by Goldberg and Targen [15] and Approximation method by Karger and Stein [16].

The solution of s-t minimum-cut gives a path within a CN, which become a new active contour. A new CN is generated from the new active contour. This process continues until the active contour reaches the true boundary of the target. The following is the GCBAC algorithm.

GCBAC algorithm in higher-level of view

- Initialization process
- Deformation process
- Termination of the GCBAC

Initialization process: The GCBAC initially constructs an adjacency graph from an image. All pixels on the image are converted to vertices on the graph. It, then, obtains an initial contour around the target object from a user. The initial contour is set to the current active contour.

Deformation process: During deformation, the GCBAC takes the following three steps.

Step.1	Dilate the current contour to create a CN
Step.2	Identify inner, outer and intermediate vertices to create multi-source multi-sink minimum-cut problem.
Step.3	Reduce the problem and compute the s-t minimum cut to obtain a new active contour.

Termination of GCBAC: The termination criterion of the GCBAC is based on the satiability of the cost of the minimum-cut. At the end of iterations, the cost of the minimum-cut is computed and stored. The program is terminated when the costs of the minimum-cut of the previous and the current active contours have small differences.

6.2 Discussion of GCBAC

The time complexity of the GCBAC is dominated by the time required to solve the

s-t minimum-cut problem. The time complexity of the s-t minimum-cut varies according to the algorithm. The fastest current algorithm of the s-t minimum-cut problem is developed by Karger and Stein [16]. It runs $O(n^2 log^3 n)$ time. The algorithm I used to implement the GCBAC is Ford-Fulkerson method [14]. It runs $O(mn|C|)$ time, where ***n*** is number of snaxels and ***m*** is number of edges. $|\boldsymbol{C}|$ represents the capacity of flows. The algorithm is discussed in Section 12.

The connectivity of graph is another factor that influences on the running time of the s-t minimum-cut problem and the accuracy of the boundary of the target. Fig. 6.2-1 shows 4- and 8-connectivity of the graphs.

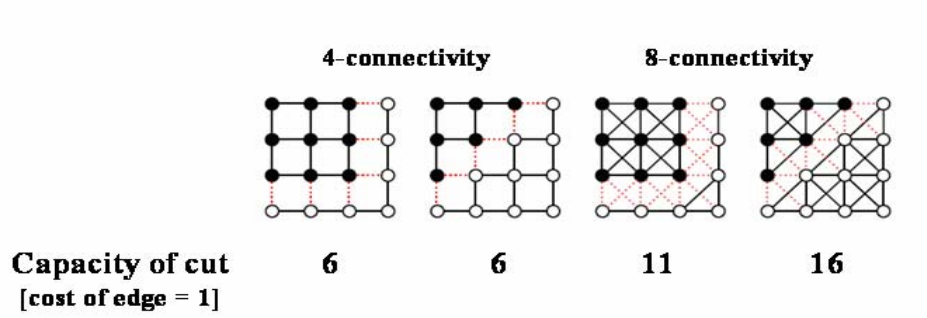


Fig. 6.2-1 Connectivity of an adjacency graph

Suppose that the costs of edges are set to 1 and *red-dot lines* are the cutting edges. The numbers below each 4 x 4 grid are capacities of the cut. Notice that the first two grids from left have the same costs of 6s. This implies that the same capacity of the cut on 4-connectivity of a graph has possibilities of representing different boundaries. In contrary, the grids for 8-connectivity have the distinct values for different boundaries. Accordingly, an active contour on 8-connectivity adjacency graph has more accuracy of representation. The tradeoff is that the computation time on s-t minimum-cut problem. Obviously, the running time increases as the number of edges increases.

The GCBAC has several advantages over the traditional snake. First, the set of weighting parameters on each energy terms are not required. Although the GCBAC still needs an initial contour around the target object from a user; however it is not necessary to set weighting parameters required in the original snake and its successors.

Another advantage is that the GCBAC does not create a self-crossing active contour. A solution of s-t minimum-cut problem returns a path that separates CN into two parts. A new active contour is formed with the outer pixels of the path, a *red-dot line* in Fig.

6.2-2. Consequently, it remove the problems with self-crossing, which sometime seen in the final contour of the snakes.

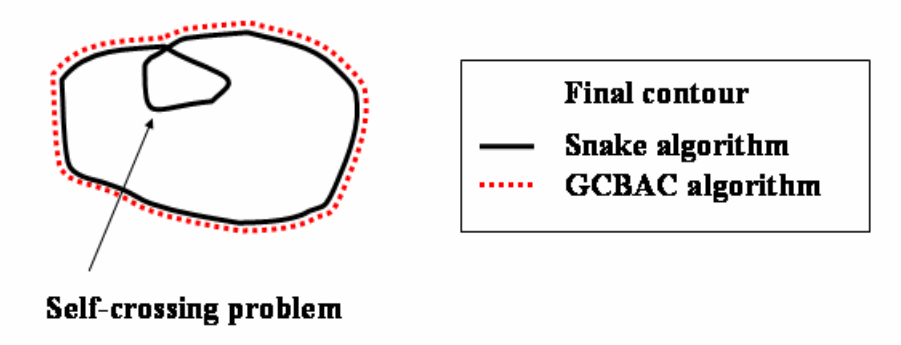


Fig. 6.2-2 Self-crossing active contour

Since a solution of s-t minimum-cut problem is a globally optimal solution in CN, the algorithm of the GCBAC has capability to jump over local minima within CN. As a result, an active contour is not attracted to the local minima, such as noises or undesired features within CN. The implementation of the GCBAC is discussed in Section 12.

7. Extracting multiple objects in an image

Generally the snake active contour or the GCBAC is designed for detecting a boundary of a single object on an image. Choi, Lam and Siu [5] proposed an algorithm to detect the contours of multiple objects in an image. The proposed algorithm has a number of limitations. The primary limitation is the relative positions of the target objects. The algorithm works merely on multiple objects that have some distances between themselves. In other words, each extracting objects must not be overlapped each other such as objects in Fig. 7.1-1.

7.1 Critical Points and Splitting and connecting operations

The algorithm starts searching for critical points on the current active contour. The critical points are all snaxels that satisfies the following conditions:

$$\begin{aligned} &\textit{snaxel } v_i \textit{ is a critical point} \\ &\quad \textit{if } (E(v_i) > T_{image} \textit{ and } E(v_{i-1}) > T_{image} \textit{ and } E(v_{i+1}) > T_{image}) \textit{ or} \\ &\quad (E(v_i) > T_{image} \textit{ and } E(v_{i-1}) < T_{image} \textit{ and } E(v_{i+1}) < T_{image}) \\ &\quad \textit{where } T_{image} \textit{ is a threshold on image force} \end{aligned} \qquad \textbf{(7.1-1)}$$

Figure 7.1-1 illustrates two pairs of the critical points on the active contour. Since the distance of the pair of the critical points on the left are further than the pre-defined distance threshold $\boldsymbol{D_T}$, these critical points are ignored. Meanwhile, the distance of the pair of critical points on the right is lower than the distance threshold. Therefore, at these critical points on the right, the "*splitting and connecting*" operations are performed to form two active contours. The *red-dot lines* in Fig. 7.1-1 represent new connections after the "*splitting and connecting*" operation. The algorithm is discussed in Section 13.

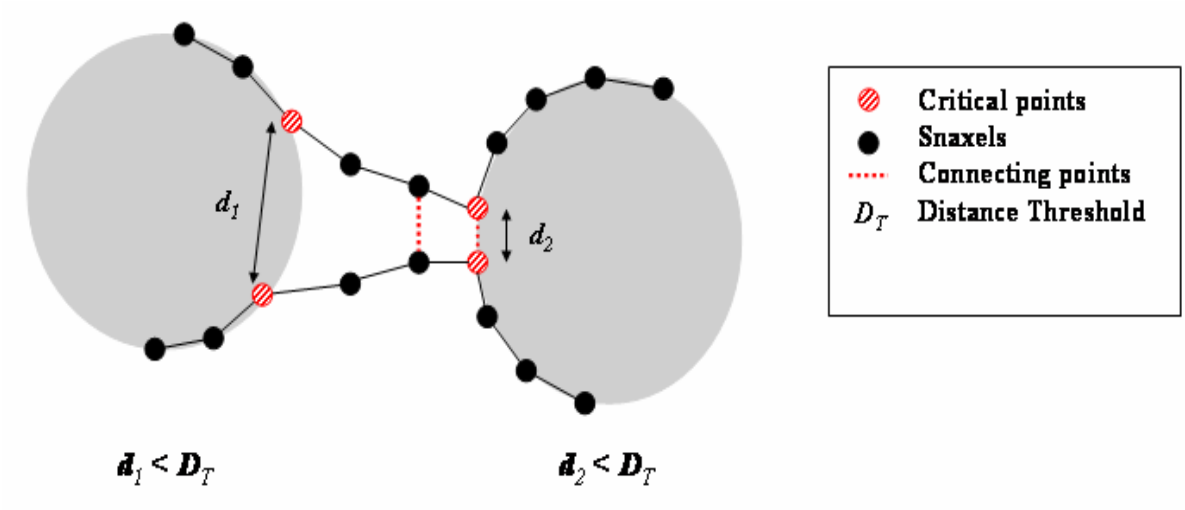


Fig. 7.1-1 Critical Points

7.2 Removal of invalid Contours

The operation of "*splitting and connecting*" sometimes creates invalid contours, as illustrated in Fig. 7.2-1. Therefore, these invalid contours must be removed at the between iterations in the deformation process. To identify an invalid contour, an area of each active contour is calculated at the end of iteration. In general, an invalid contour converges to a certain point by internal forces of the snake due to no external force inside of the contour. Some invalid contours stop converging to a point and cease their movement (Fig. 7.2-2a). Other contours grow in a reverse direction. This is because snaxels are too close each other, some snaxels skip over other snaxels (Fig. 7.2-2b). The area of contours shown in Fig. 7.2-1 becomes zero. Meanwhile the area of contours shown in Fig. 7.2-2 becomes negative. Since the areas of these contours signify the invalidity of the active contours, the threshold is set on the area of contour at end of

iteration for the removal of the invalid contours.

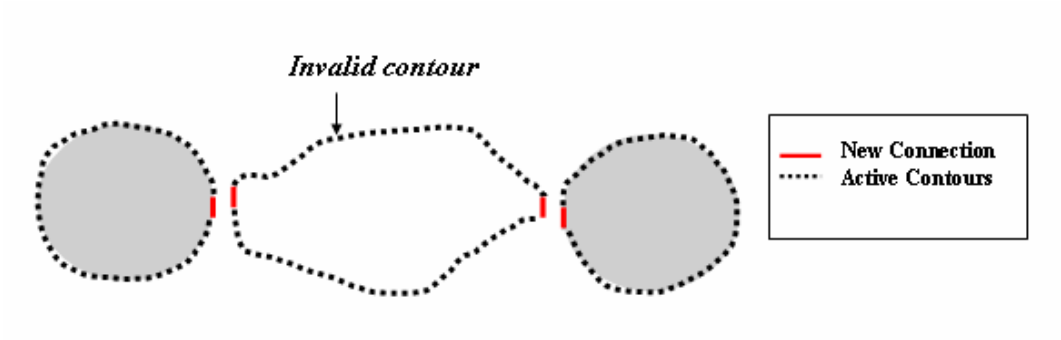


Fig. 7.2-1 Creation of an invalid contour

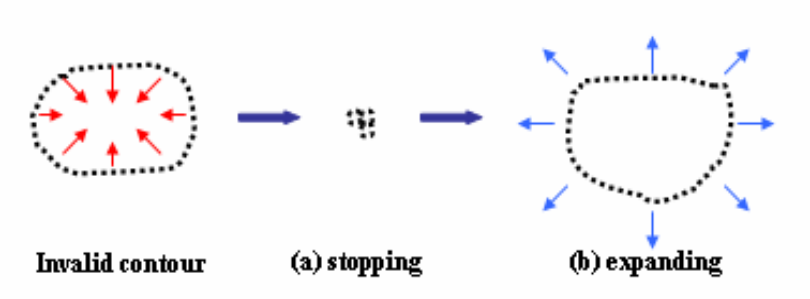


Fig. 7.2-2 Behavior of an invalid contour

7.3 Discussion of extracting multiple objects

The proposed algorithm by Choi, Lam and Siu performs well with snakes such as the adaptive or the GVF snake. However, it won't works with the original snake. Figure 7.3-1 shows the result of the original snake performed on the image with two objects. Since the original snake dose not have enough forces to move the contour inward (*blue arrows*), the two pairs of critical points (*yellow circles*) won't be close enough to fall under the distance thresholds. For this reason, the original snake can not extract multiple objects with this algorithm.

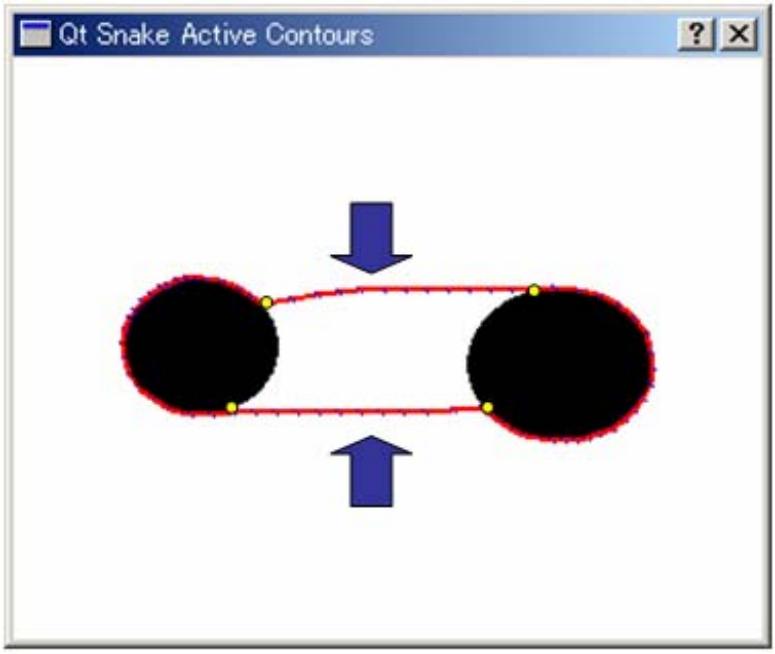


Fig. 7.3-1 Result of the original snake on multiple objects

8. Implementation of Application

I implemented an application that helps to understand algorithms of active contour models. Besides these algorithms, I've implemented some other functions that help to understand and analyze an image. Table 7-1 below shows the list of available functionalities of the application. The screenshots of the application are provided in Section 16.

Functions		Screenshots
Filters	Gaussian Filter	Fig.16.2-1
	Median Filter	Fig.16.2-2
	User-defined Filter	Fig.16.2-3
Histogram	Histogram equalization	Fig.16.3-1
	Histogram operations	Fig.16.3-2
Property	Show the information of an image	Fig.16.4-1
	Show gradients of an image	Fig.16.4-2
	Show GVF filed of an image	Fig.16.4-2
ACM	Original snake	Fig.16.5-1
	Adaptive snake	Fig.16.5-1
	GVF snake	Fig.16.5-1

	GCBAC (Graph Cut Based Active Contour)	Fig.16.5-2

Table 8-1 List of the available functions in the application

8.1 Libraries

I used QT library from Trolletch [21] to build framework and GUI components of the application. QT is a complete C++ application development framework that include a class library and tools for cross-platform development and internationalization [21]. Qt has rich class library for image manipulation and GUI, graphical user interface, components. Rapid Application Development known as RAD, is a method of quickly developing software. RAD program allow you to create GUI in minutes. A user simply drags GUI components to proper position onto an empty form, and the RAD tool takes care of the source code for you. Qt provides such a tool called "QT Designer", which handles such task for you.

I also used OpenGL library [22] to visualize primary on showing the gradient vector and the gradient vector field derived from image. OpenGL provides a cross-platform API for writing applications that produce 2D and 3D computer graphics. The interface consists of about 250 function calls to draw complex 2D or 3D scenes from simple primitives.

9. Implementation of the original snake

The implementation of the snake involves energy terms, approaches of deformation process, program termination and obtaining an initial contour from a user. Each of them is discussed in the following sections.

Figure 9-1 shows partial image of the Snake Control Panel. The sliders at the top control the weights of internal and external forces of the snake. The box in the middle is set the intervals of snake control points. The approaches of the snake algorithm are selected at the algorithm section, discussed in Section 9.4 and 9.5. The button "Roll back" moves the current active contour to the previous locations. The "Save Contour" saves the current contour and the "Retrieve Contour" retrieves the saved contour. The details of the Snake Control Panel are shown in Section 16.5.

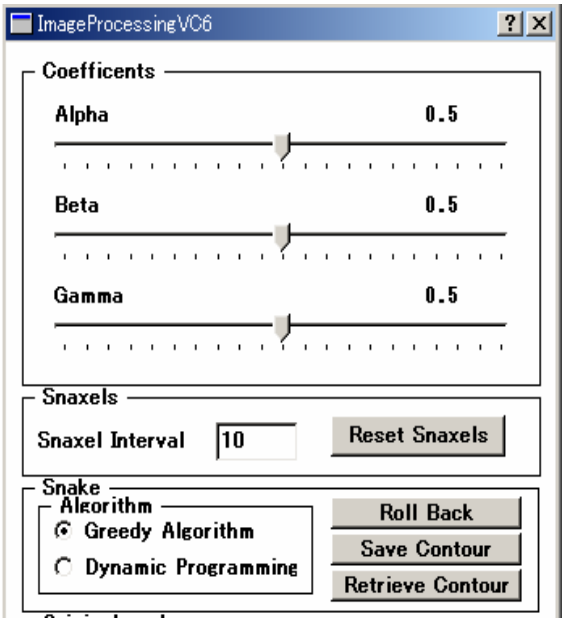


Fig.9-1 Snake Control Panel

9.1 Internal Energy terms

The internal energy terms are defined in Eq. (2.1-1). They are the first and second derivative of the parametric curve of the snake. Discreting each energy term of the internal energy, we have the following formulas.

$$\begin{aligned} E_{elastic}(s) &= \alpha(s)\cdot\left|\frac{dv(s)}{ds}\right|^2 \\ &\approx \alpha(s)\cdot\left|\frac{v_i - v_{i-1}}{2}\right|^2 \end{aligned} \qquad \textbf{(9.1-1)}$$

$$\begin{aligned} E_{bending}(s) &= \beta(s)\cdot\left|\frac{dv(s)^2}{ds^2}\right|^2 \\ &\approx \beta(s)\cdot\left|(v_{i+1} - v_i) - (v_i - v_{i-1})\right|^2 \\ &\approx \beta(s)\cdot\left|v_{i+1} - 2\cdot v_i + v_{i-1}\right|^2 \end{aligned} \qquad \textbf{(9.1-2)}$$

$$v_i = (x_i, y_i) \qquad i = 0 \ldots N-1 \quad \textit{where } N \textit{ is number of snaxels}$$

The elastic force involves two adjacent snaxels. The distances of two adjacent snaxels are used as an elastic energy term. Meanwhile, the bending force involves three adjacent snaxels. The curvatures of these three adjacent snaxels are computed and used as a bending energy term.

9.2 External Energy term

The external energy is derived from the image data. The gradient magnitude of intensities of an image is used as external energy. Equation (9.2-1) is used to compute external energy at each snaxel [23]. In general, the Gaussian filter is applied on the image to increase capture range. Thus, the original image is filtered out prior to the computation of the gradient magnitude.

$$\nabla f\ (x,y) = mag(\nabla f) = \begin{bmatrix} \frac{dv}{dx} \\ \frac{dv}{dy} \end{bmatrix} \qquad \textbf{(9.2-1)}$$

$$= \sqrt{(\frac{dv}{dx})^2 + (\frac{dv}{dy})^2}$$

9.3 Initial contour at the initialization process

At the initialization process, a user drags mouse to create an initial contour. The points under the contour are stored in a list. The snaxels are chosen from some of points in the list according to the interval specified by a user. Since snake works well when each segment of snaxels has equal length, the interval gives even spaces between snaxels. Figure 9.3-1 shows the initial contours with intervals of *15* and *40*.

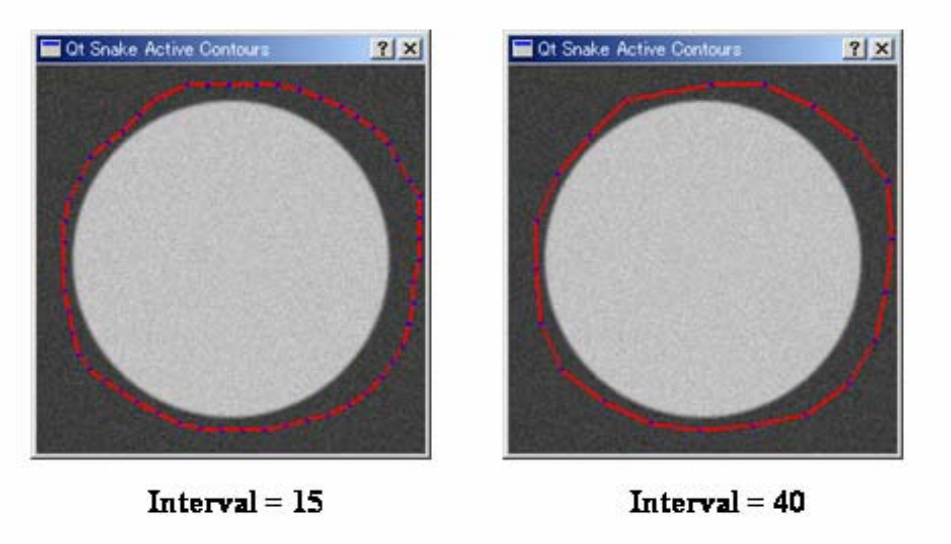


Fig. 9.3-1 Initial contours and their intervals

9.4 Greedy Algorithm in the deformation process

The greedy algorithm for the snake is relatively simple and easy to be implemented. During the iterations, the sum of the internal and external energies is computed at a snaxel and its 8 neighboring pixels. A location that has the smallest energy is set to be a new location. Therefore, a snaxel moves to one of the eight possible neighboring pixels, the *red arrows* in Fig 9.4-1 *left*, or stay the same location if it cannot find the smaller energy than that the current snaxel has.

The faster greedy algorithm is proposed by Lam and Yan [3]. Instead searching all neighboring locations, it searches 4 possible pixels (*blue-dot arrows* in Fig 9.4-1 *right*). They claim that the neighbors of the location having the smallest value of the energy function also have the small values. Therefore the computation required in searching for a new location can be greatly reduced by searching the neighbors in alternate patterns in Fig. 9.4-1. By alternating the two search patterns I and II, the whole space searched by the greedy algorithm.

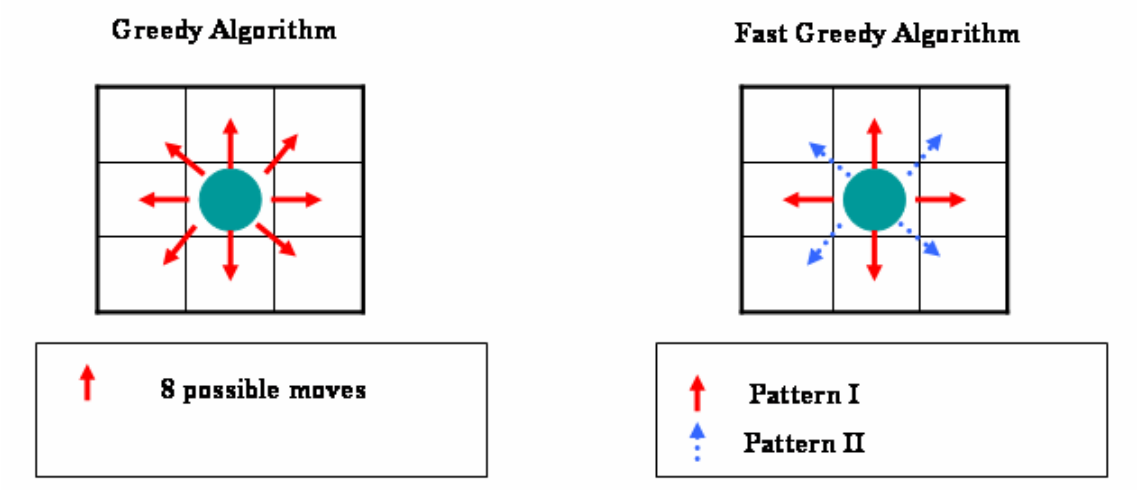


Fig. 9.4-1 Possible movements at a snaxel during iteration

The experimental results of comparison of two algorithms in Table 9.4-1 are provided by Lam and Yan [3]. The experiment was conducted on San Sparc 2 workstation.

α, β, γ		Greedy	Fast greedy
(1.0, 1.0, 1.0)	Iterations	17	20
	Time [s]	1.0730	0.7118
(1.0, 1.0, 1.2)	Iterations	12	18
	Time [s]	0.7618	0.6422
(1.0, 1.2, 1.0)	Iterations	16	18
	Time [s]	1.0105	0.6335
(1.2, 1.0, 1.0)	Iterations	17	18
	Time [s]	1.1105	0.6338

Table 9.4-1 Experimental results: Number of iterations and executing time for the greedy and the fast greedy algorithms [3]

The time complexity of both the greedy and the fast greedy algorithms is $O(n \cdot t)$ *time* where n is the number of snaxels and t is the number of iterations executed in the deformation process. Hence, this is inexpensive operation. However, no matter the greedy or the fast greedy algorithms, neither algorithm would guarantee the optimal solution. During iterations, the snake searches locally optimal choices from neighboring pixels at each snaxel. To compute global optimal solution, all possible combinations of snaxels must be computed. Next section shows how to solve it efficiently.

9.5 Dynamic Programming in the deformation process

Unlike the greedy algorithm, the dynamic programming finds a globally optimal solution. The idea of the dynamic programming is to break a large problem down into incremental steps so that, at any given stage, the optimal solutions are known to sub-problems. When the technique is applicable, this condition can be extended incrementally without having to change previously computed optimal solutions to sub-problems. It removes redundant computation time of the algorithm.

For the energy-minimizing algorithm, it normally takes $O(9^n)$ *time* to compute all combinations of the possible moves of the snaxels, where n is the number of snaxels on the active contour. With the technique of the dynamic program, it can be reduced the computation time significantly.

Since the internal energy involves three consecutive snaxels, the equation of the total energy of the snake is defined as:

$$E_{snake}(v_1, v_2, v_3, \cdots, v_n) = E_1(v_1, v_2, v_3) + E_2(v_2, v_3, v_4) + \cdots + E_{n-2}(v_{n-2}, v_{n-1}, v_n) \quad \textbf{(9.5-1)}$$

$$\textit{where } E_{i-1}(v_{i-1}, v_i, v_{i+1}) = E_{external}(v_i) + E_{internal}(v_{i-1}, v_i, v_{i+1})$$

In order to apply the dynamic programming to Eq. (9.5-1), a stage variable S is defined as follows:

$$S_i(v_{i+1}, v_i) = \min_{vi-1} S_{i-1} + E_{external}(v_i) + E_{internal}(v_{i-1}, v_i, v_{i+1}) \quad \textbf{(9.5-2)}$$

For each stage, a new energy and the energy in the previous stage S_{i-1} are added and stored in the current stage S_i. The total accumulated energies are stored at S_{n-2}. The trace back starts with the minimum energy found at S_{n-2}.

It takes $O(9^2)$ *time* to update stage variable for each neighboring snaxels. Therefore, the total time needed for a snaxel become $O(9 \cdot 9^2)$ *time*. Since the number of snaxels is n, the time complexity of the snake algorithm with the dynamic programming becomes $O(n \cdot 9^3)$ *time*. Comparing to the $O(9^n)$ *time* without the dynamic programming, it makes significant improvement.

9.6 Termination of the program

At some point of the time, the program has to be terminated. Several terminating criteria are discussed earlier. I employed the following three terminating criteria.

Terminating Criterion	Threshold
# of snaxels moved	**80%** 80% of snaxels stays the same location.
CA-Criterion	**0.001%** Ratio of change of the area in two consecutive iterations
CL-Criterion	**0.001%** Ratio of change of the length in two consecutive iterations

Table 9.6-1 Terminating criteria and their thresholds

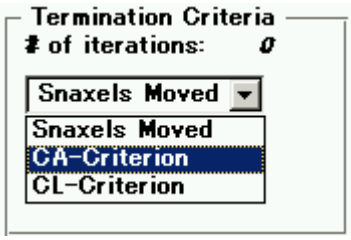


Fig. 9.6-1 Terminating criteria on *Snake Control Panel*

Figure 9.6-1 is the snapshot of the termination criteria selection on the Snake Control Panel. The snake is terminated by one of the criteria in the list box. The threshold for "Snake Moved" is set to be *80%*. The snake is terminated when *80%* of snaxels are stable. Meanwhile, the thresholds for CA- and CL-Criterion are set to *0.001%*. This means that the snake is terminated when the ratio of change in area or length of the snake is less than *0.001* in two consecutive iterations. In addition to these criteria, the snake is terminated in every *50*-iteration. This prevents the snake to go into an infinity loop since the snake sometime doesn't fall into any of these terminating criteria.

9.7 Results of the original snake

The results of the original snake are shown in the following link.

Link : Link to Image Gallery

10. Implementation of Adaptive snakes

The implementation of the adaptive snake involves computation of an adaptive force at snaxels and insertion or deletion of snaxels during the deformation process. These functionalities are integrated to the algorithm of the original snake. The implementation details are discussed in the following sections.

Figure 10-1 shows parameter setting for Adaptive snake on the Snake Control panel. A user can set the strength of the amplitude of the adaptive force at box under "Adaptive Force". The adaptive force is applied on a snaxel only if the snaxel has lower image force than the threshold specified in the box under "Threshold". The default value for the adaptive force is set to *3* pixels and the threshold of the image force is *0.05*.

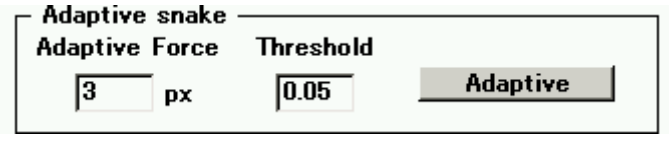


Fig. 10-1 Snake Control Panel for Adaptive Snake

10.1 Adaptive force

To compute an adaptive force on i^{th} snaxel, a unit normal vector $\vec{n}$ at v_i need to be calculated. Since the normal vector $\vec{v}'$ at v_i is perpendicular to the vector ($v_{i+1} - v_{i-1}$), it satisfy Eq. (10.1-1). Then the normal vector is normalized, Eq (10.1-2) to obtain the unit normal vector $\vec{n}$ at v_i. A new location of a snaxel is computed in Eq. (10.1-3)

$$\vec{v}' \cdot (v_{i+1} - v_{i-1}) = 0 \qquad \textbf{(10.1-1)}$$

$$\vec{n} = \frac{\vec{v}'}{\|\vec{v}'\|} \qquad \textbf{(10.1-2)}$$

$$v_i'\,(new\ location) = k \cdot \vec{n} + v_i \qquad \textbf{(10.1-3)}$$

10.2 Insertion and deletion of snaxels

Since the nature of the algorithm of the adaptive snake, it sometimes creates uneven spaces between snaxels. This is not a desirable phenomenon for the snake to work well. Therefore, two operations, insertion and deletion, are need to be applied after a new active contour is generated. The conditions of the operations and their thresholds are listed in Table 10.2-1 and 10.2-2. Figure 10.2-1 shows some situations that the operations of insertions and deletions are needed.

Operations	Condition
Delete the snaxel v_i	$d_1 < t_1 \cdot \bar{d} \ \| \ d_2 < t_2 \cdot \bar{d}$
Insert a new snaxel in the middle of v_i and v_{i+1}	$d_1 < t_3 \cdot \bar{d}$
* $\bar{d}$ *is an average distance of the snaxels*	

Table 10.2-1 Conditions of deletions and Insertions

Thresholds	Default value
T_1	0.8
T_2	0.8
T_3	1.8

Table 10.2-2 Distance thresholds

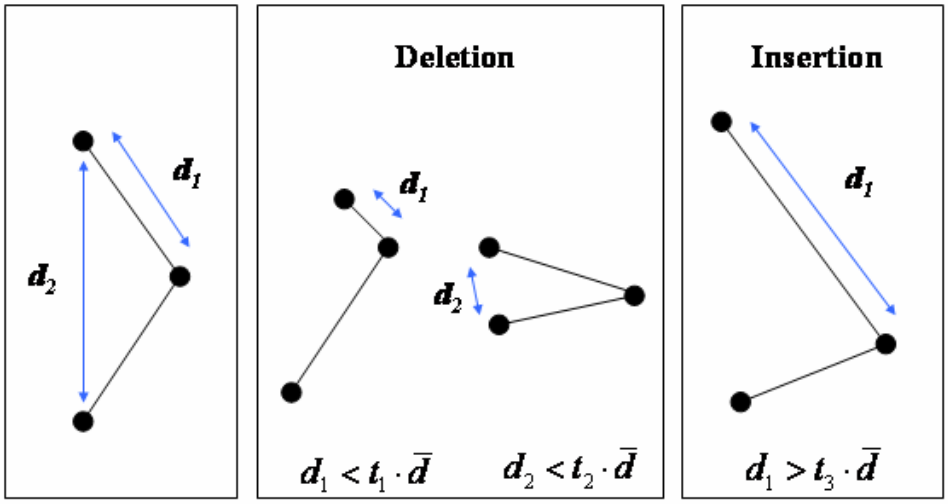


Fig. 10.2-1 Deletion and insertion of snaxels

10.3 Results of Adaptive snakes

The results of Adaptive snake are shown in the following link.

Link : Link to Image Gallery

11. Implementation of GVF snakes

The implementation of the GVF snake is primary focused on the creating the GVF field derived from the given image. The algorithm of the GVF snake is the same as the original snake. The only difference is that the GVF snake uses external force form the GVF field instead of the gradient magnitude, which is used in the original snake.

Figure 11-1 shows parameter setting for GVF snake on Snake Control panel. A user can set a number of iteration to compute a gradient vector field and a value of the ***mu***, The ***mu*** is set according to the amount of noise present in the image. The button, "Update Field", update the GVF field only if the number of the iteration has been changed.

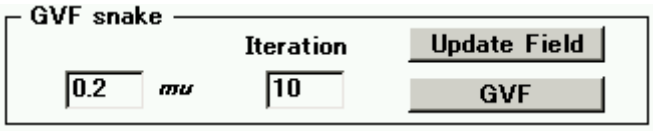


Fig. 11-1 Snake Control Panel for GVF Snake

11.1 Gradient Vector Field

The gradient vector field is derived from an image. The gradient vector field is computed with the traditional GVF snake. The following equations are used to compute the gradient vector field. Using the calculus of variations [24], the GVF field can be computed by the following Euler equations [11].

$$\mu \cdot \nabla^2 - (u - f_x) \cdot (f_x^2 + f_y^2) = 0 \qquad \textbf{(11.1-1a)}$$

$$\mu \cdot \nabla^2 - (u - f_y) \cdot (f_x^2 + f_y^2) = 0 \qquad \textbf{(11.1-1b)}$$

where ∇^2 is the Laplacian operator

Equation (11.1-1) can be solved by treating u and v as functions of the time and solving [11].

$$u_t(x, y, t) = \mu \cdot \nabla^2 u(x, y, t) - b(x, y) \cdot u(x, y, t) + c^1(x, y) \qquad \textbf{(11.1-2a)}$$

$$v_t(x, y, t) = \mu \cdot \nabla^2 u(x, y, t) - b(x, y) \cdot v(x, y, t) + c^2(x, y) \qquad \textbf{(11.1-2b)}$$

where

$$b(x, y) = f_x(x, y)^2 + f_y(x, y)^2$$
$$c^1(x, y) = b(x, y) \cdot f_x(x, y)$$
$$c^2(x, y) = b(x, y) \cdot f_y(x, y)$$

To set up the interactive solution, let the indices i, j and n correspond to x, y and t respectively, and let the spacing between pixels be Δx and Δy and the time step for each iteration be Δt. Then the required partial derivatives can be approximated as

$$u_t = \frac{1}{\Delta t}(u_{i,j}^{n+1} - u_{i,j}^{n})$$
$$v_t = \frac{1}{\Delta t}(v_{i,j}^{n+1} - v_{i,j}^{n})$$
$$\nabla^2 u = \frac{1}{\Delta x \cdot \Delta y}(u_{i+1,j} + u_{i,j+1} + u_{i-1,j} + u_{i,j-1} - 4 \cdot u_{i,j})$$
$$\nabla^2 u = \frac{1}{\Delta x \cdot \Delta y}(v_{i+1,j} + v_{i,j+1} + v_{i-1,j} + v_{i,j-1} - 4 \cdot v_{i,j})$$

Substituting these approximation into Eq. (11.1-2) gives the following the

interactive solution.

$$u_{i,j}^{n+1} = (1 - b_{i,j}\Delta t)u_{i,j}^{n} + r \cdot (u_{i+1,j}^{n} + u_{i,j+1}^{n} + u_{i-1,j}^{n} + u_{i,j-1}^{n} - 4u_{i,j}^{n}) + c_{i,j}^{1} \quad \textbf{(11.1-3)}$$

$$v_{i,j}^{n+1} = (1 - b_{i,j}\Delta t)v_{i,j}^{n} + r \cdot (v_{i+1,j}^{n} + v_{i,j+1}^{n} + v_{i-1,j}^{n} + v_{i,j-1}^{n} - 4 \cdot v_{i,j}^{n}) + c_{i,j}^{2} \quad \textbf{(11.1-3)}$$

$$\textit{where } r = \frac{\mu \Delta t}{\Delta x \Delta y}$$

In my programming code, I set the spacing Δx and Δy, and the time step Δt to be 1. The following codes are used inside the iteration process.

```
for (k = 0; k < number of iteration; k++)
{
    …
    for (i = 0; i < width * height; i++)
    {
        …
        du[i] = (1 − b[i]) * du[i] + mu · lapU[i] + c1[i];
        dv[i] = (1 − b[i]) * dv[i] + mu · lapV[i] + c2[i];
        …
    }
    …
}
where dv, du, b, c1, and c2 are double arrays
```

Figure 11.1-1 shows the comparison of capture ranges of the original snake and the GVF snake under the *blue circle* on the star object. The GVF field in Fig.11.1-1(b) is computed with *60* iterations. It is observed that the external forces generated by the original snake have narrow capture range.

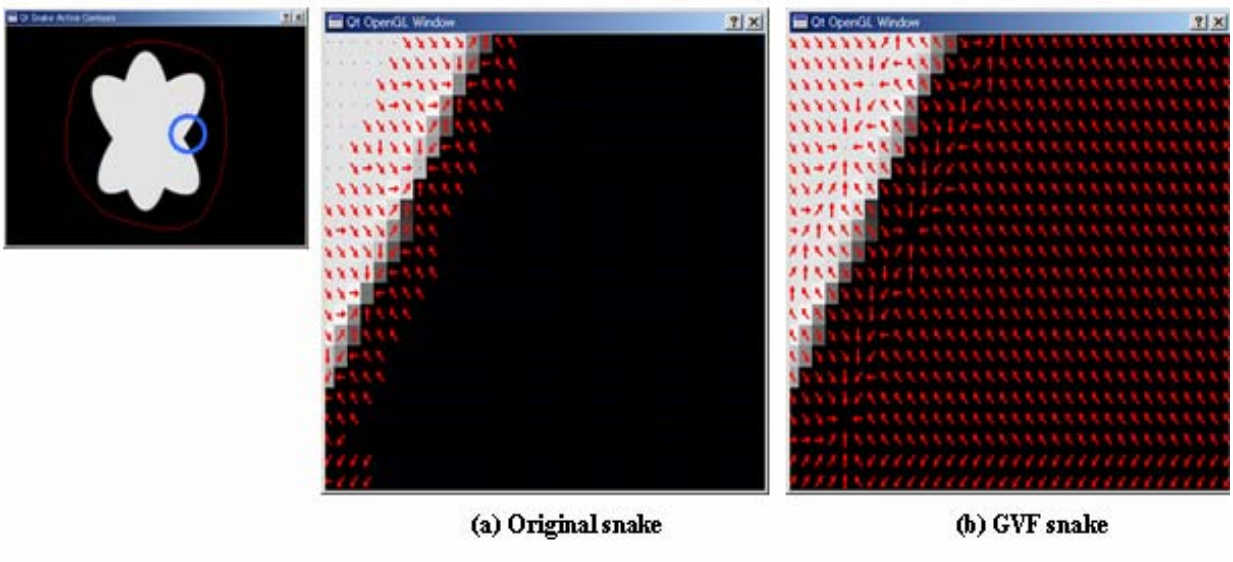


Fig.11.1-1 Capture ranges of the original snake (a) and the GVF snake (b)

11.2. Results of GVF Snakes

The results of the GVF snake are shown in the following link.

Link : Link to Image Gallery

12. Implementation of GCBAC

Similar to the snake, the GCBAC requires a user to draw an initial contour around the subject object. It takes the same procedure of the snake to obtain an initial contour. The keys components of the implementation of the GCBAC are dilation, cost of an edge and the s-t minimum-cut problem. Each component is discussed in the following sections.

12.1 Dilation

The dilation process generates a CN (Contour Neighbor) of the current contour. The *red points* in Fig. 12.1-1 are the points on the initial contour. A CN is generated with a matrix centered of these initial points. The size of a matrix is specified by a user. Fig. 12.1-2 shows CN computed with size of 5 and 10. The regions with yellow color are CN of the active contour. The number of vertices and edges within the CN are listed on the table on Fig. 12.1-2.

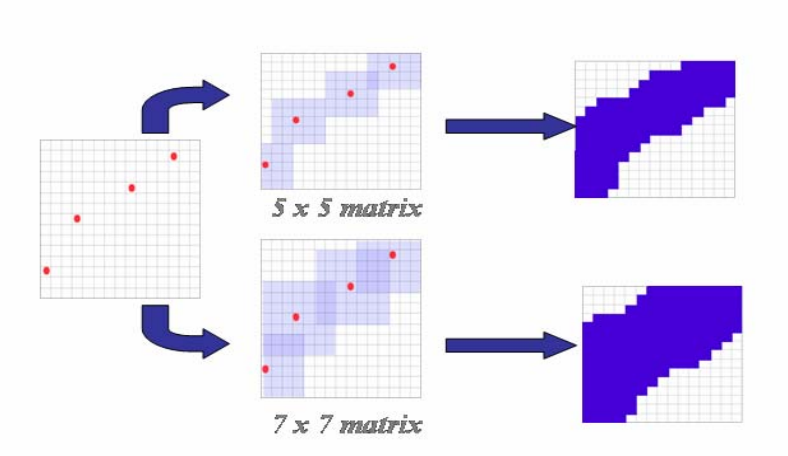
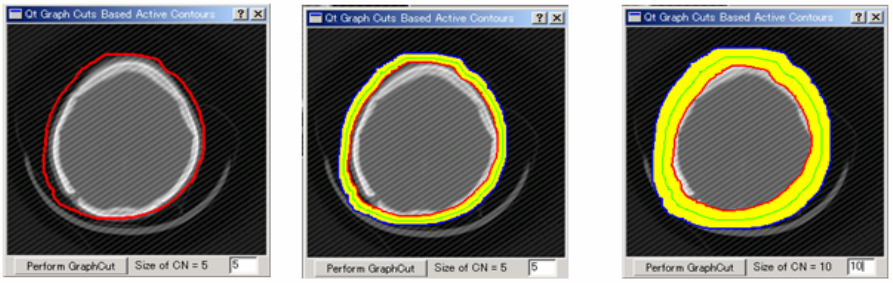


Fig. 12.1-1 Dilation with 5x5 and 7x7 matrix



(a) Initial contour **(b) CN Size = 5** **(c) CN Size = 10**

Image size 271 x 259

	# of Edges	Outer px	Inner px	CN px
(b)	21458	1181	1104	5456
(c)	51007	1259	961	12450

Fig. 12.1-2 Size of CN on sample image

12.2 Cost of edges

Equation 12.2-1 is the formula to compute the costs of the edges on the graph. The $C(i, j)$ is the cost of the edge between vertices ***i*** and ***j***.

$$C(i,j) = \left(g(i,j) + g(j,i)\right)^6$$
$$g(i,j) = \exp\left(\frac{-grad_{ij}(i)}{\max_k(grad_{ij}(k))}\right) \qquad \textbf{(12.2-1)}$$

note :

- $grad_{ij}(k)$ ***is the gradient at location k in the direction of*** $i \rightarrow j$
- $C(i,j) == C(j,i)$

Equation (12.2-1) is designed to assign the high value to the edges on the high gradient. These edges on the high gradient are most likely boundaries of the objects or noises in the image. Meanwhile, the equation assigns the low value to the edges on the homogenous area. Generally, the background or the surfaces of the objects on the image have no variance of gradients; therefore, these areas become homogenous. Since the algorithm of the s-t minimum-cut problem, discussed in the next section, starts to eliminate form the edges with low value, the solution of the minimum-cut would be the edges with high value. This is desirable because the boundary of the target object exist on the edges on high gradient. As a result, the active contour formed by these edges, reaches the contour of the target object on the image.

12.3 s-t minimum-cut problem

As I mentioned earlier, there are several algorithms available to solve the s-t minimum-cut problem. I used Ford-Fulkerson algorithm based on augmenting path method in my implementation of the GCBAC. The basic flow of the algorithm takes the following steps.

Step 1: *Create the s-t minimum-cut problem form the given CN.*

First, it creates a source and a sink node. Then, the source node is connected to vertices on outer boundary of the CN. Similarly, the sink node is connected to the vertices on the inner boundary of the CN. The costs of the edges between these connections are assigned the maximum values. This is because we want to find the cutting edges within the CN.

Step 2: *Find a path form the source to sink node.*

The BFS (Breath First Search) algorithm is used to find a path from source to sink node. Once a path is found, it determines the minimum capacity ∇c on the path.

Step 3: *Flow the minimum cost to the path and update the residual graph*

It updates the residual graph by flowing ∇c found in the step 2 to all the edge on the path. This guarantees at least one edge are cut.

Step 4: *Repeat until no path from the source to sink node is found.*

It continues step 2 and 3 until there is no path from the source to the sink node. Once no path is found, the program computes a new active contour, determined by the cutting edges

12.4 Termination

The cost of the min-cut is computed each iteration in the deformation process. The current cost is compared with the previous cost of the min-cut. The program is terminated when the difference of these costs is lower than pre-defined threshold.

12.5 Results of GCBAC

The results of the GCBAC are shown in the following link.

Link : Link to Image Gallery

13. Implementation of extracting multiple objects

Since extracting multiple objects needs to keep track multiple active contours simultaneously, the data structure must be change. Figure 13-1 illustrates the data structure for extracting single and multiple objects. Instead of using a list of snaxels, *snake* (std::list), a list of lists of snaxels, *snakeList* (std::list of std::list) is used for multiple objects extraction.

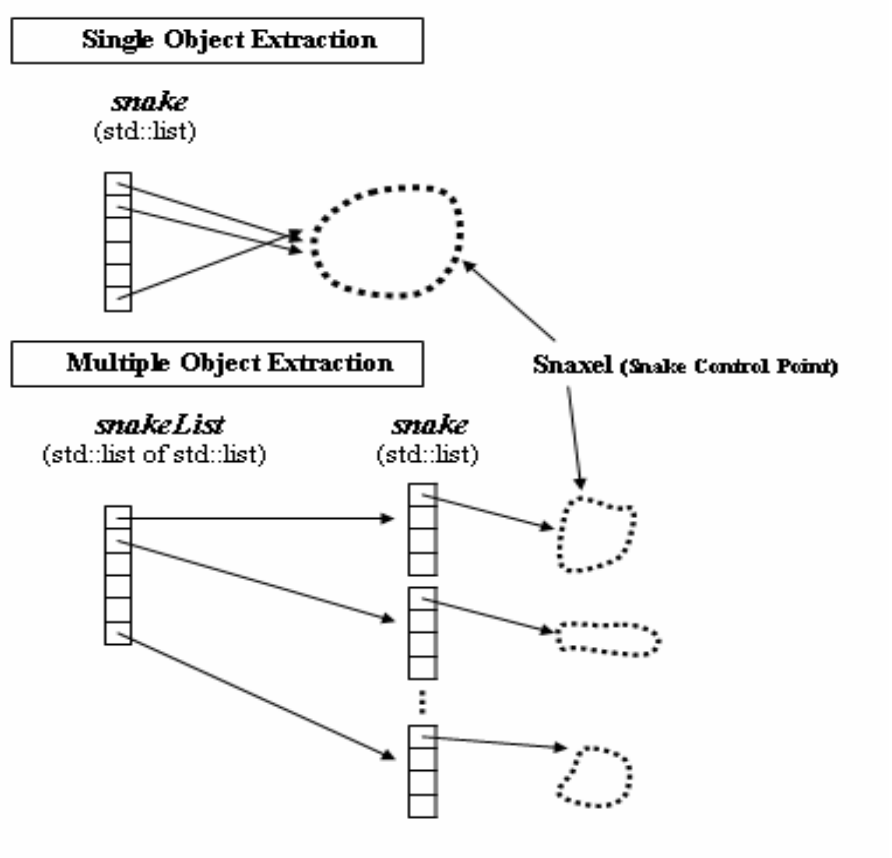


Fig. 13-1 Data structure of the snake

13.1 Algorithm of extracting multiple objects

The following three steps are performed to extracting multiple objects in an image. These steps are taken at the end of each iteration in the deformation process of the snake algorithm.

Step1: Identify all critical points

Each snaxel on the current active contour is searched and determined if it is a *critical points.* Any snaxel that satisfied the conditions of a critical point defined in Eq. (7.1-1) is marked as a critical point and pushed back to *cpList*. The *cpList* is defined as:

$$\boldsymbol{cpList}\ (\boldsymbol{critical\ \ point\ list}) = \{\boldsymbol{c}_i(\boldsymbol{s}) \mid \boldsymbol{i} = 0,1,2,\ldots \boldsymbol{n}-1\} \qquad \textbf{(13.1-1)}$$

Step2: Find pairs of the critical points

In this step, pairs of critical points are determined in the *cpList*. The table

below shows algorithm to find a pair of critical points in the *cpList*.

Algorithm to identify a pair of critical points [5]	
Step 1	Set $\boldsymbol{i} = 0$ as the starting critical point C_i.
Step 2	Set $\boldsymbol{j} = \boldsymbol{i} + 1$ for the other critical point C_j.
Step 3	Compute the distance, $\boldsymbol{d_{ij}}$, between the two critical points C_i and C_j.
Step 4	If d_{ij} is less than a threshold, C_i and C_j are marked as a pair of connected points.
Step 5	If $\boldsymbol{j} < \boldsymbol{n}$, then $\boldsymbol{j} = \boldsymbol{j} + 2$ and go to Step 3. Otherwise, go to Step 6.
Step 6	If $\boldsymbol{i} < \boldsymbol{n}$, then $\boldsymbol{i} = \boldsymbol{i} + 1$ and go to Step 2. Otherwise, go to Step 7.
Step 7	END

Figure 13.1-1 illustrates example of how the pairs of critical points are searched for the first two critical points C_0 and C_1 in the *cpList*. The *Black arrows* from left to right are search path for C_0 and the *red-dot arrows* are for C_1. Once a pair of critical points is found, it is added to *pcpList*. The *pcpList* is defined in Eq. (13.1-2).

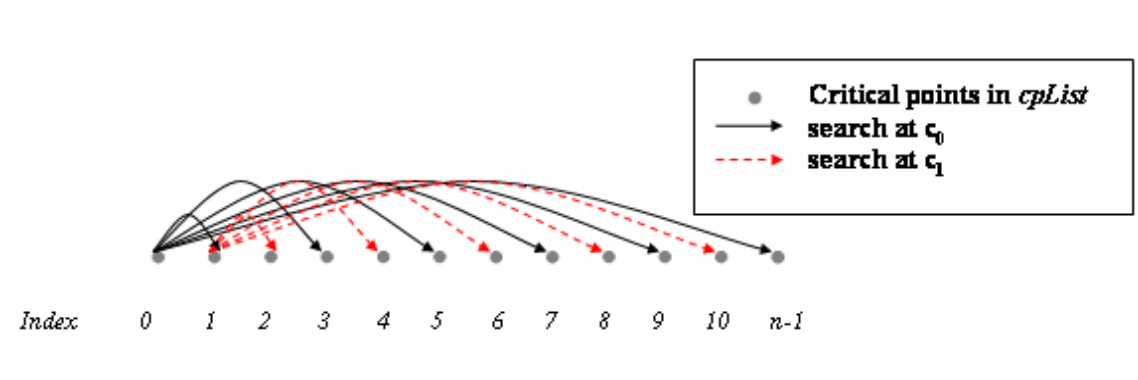


Fig.13.1-1 Searching paths for the first two critical points, C_0 and C_1

$$\boldsymbol{pcpList\ (pairs\ of\ critical\ point\ list)} = \{\boldsymbol{P_k}(\boldsymbol{v_i}, \boldsymbol{v_j}) \mid \boldsymbol{i < j < n, k < m}\} \quad \textbf{(13.1-2)}$$

Step3: Perform "Splitting and Connecting" operation

All the pairs of critical points found at the previous step are connected. A new contour is pushed back to the *snakeList*.

13.2 Removal of invalid contours

The invalid contour are remove at the between iterations. The threshold on the area, discussed in Section 7.2, is set to *0.1* as default value. Any contours below the threshold are removed from the *snakeList.*

13.3 Results of extracting multiple objects

The results of extracting multiple of objects are shown in the following link.

Link : Link to Image Gallery

14. Summary

In this paper, I have reviewed and discussed some of the mythologies of active contour models; the original snake and its successor, and the GCBAC. I have implemented these metrologies to understand more about mechanism of their algorithms. Many improvements based on the traditional snake or alternative approaches to detect boundaries of object(s) have been proposed and made. The performance of each algorithm is superior to others in terms of time complexity and accuracy on certain image. However, they either solve one or more problems but creating new difficulties.

The primary goal of the active contour models is to detect the boundaries of the target object(s) in the image. The one of ultimate goals of active contour model is to achieve the goal with less human interventions. Since all the approaches I have reviewed requires a user to input one or more parameters, a user need to have prior understand of the image. The wrong parameter settings could end up with painful results. Therefore, automatically finding or predict the values of these parameters would be the next step.

15. Reference

[1] M. Kass, A.Witkin, and D. Terzopoulos. "Snakes: Active contour models", International Journal of Computer Vision, 1(4):321-331, 1988.

[2] Ivins, J; Porrill, J: 1994. "Statistical Snakes: Active Region Models", Fifth British Machine Vision Conference (BMVC'94; York, England): vol 2, pp 377-386.

[3] K.M. Lam and H.Yan: Electronics Letter 6th January 1994, Vol.30 No.1

[4] C. Xu, A. Yezzi, Jr., and J. L. Prince, "On the Relationship between Parametric and Geometric Active Contours", in Proc. of 34th Asilomar Conference on Signals, Systems, and Computers, pp. 483 -489, October 2000.

[5] Wai-Pak Choi, Kin-Man Lam and Wan-Chi Siu, "An adaptive active contour model for highly irregular boundaries", Pattern Recognition, Vol. 34, pp. 323-331, 2001.

[6] Y.Y. Wong, P.C. Yuen, C.S. Tong, “Contour length terminating criterion for snake model”, Pattern Recognition 31 (5) 597-606, 1998.

[7] L. D. Cohen. "On active contour models and balloons", CVGIP: Image Understanding, 53(2):211-218, March 1991.

[8] C. Xu and J.L. Prince, "Gradient Vector Flow: A New External Force for Snakes", Proc. IEEE Conf. on Comp. Vis. Patt. Recog. (CVPR), Los Alamitos: Comp. Soc. Press, pp. 66-71, June 1997.

[9] C. Xu and J. L. Prince, "Snakes, Shapes, and Gradient Vector Flow", IEEE Transactions on Image Processing, 7(3), pp. 359-369, March 1998.

[10] C. Xu and J. L. Prince, "Global Optimality of Gradient Vector Flow", Proc. of 34th Annual Conference on Information Sciences and Systems (CISS'00), Princeton University, March 2000.

[11] C. Xu and J. L. Prince, "Gradient Vector Flow Deformable Models", Handbook of Medical Imaging, edited by Isaac Bankman, Academic Press, September, 2000.

[12] C. Xu and J. L. Prince, "Generalized Gradient Vector Flow External Forces for Active Contours", Signal Processing --- An International Journal, 71(2), pp. 131-139, December 1998.

[13] Ning Xu, Ravi Bansal and Narendra Ahuja, "Object Segmentation Using Graph Cuts Based Active Contour", IEEE International Conference on Computer Vision and Pattern Recognition, June, 2003

[14] L. R. Ford, Jr. and D. R. Fulkerson, "Maximal flow through a network", Canadian Journal of Mathematics, vol.8, pp. 399-404, 1956.

[15] A. Goldberg and R. Tarjan. A new approach to the maximum flow problem. Journal of the Association for Computing Machinery, 35(4):921--940, October 1988.

[16] D. R. Karger and C. Stein, “A new approach to the minimum cut problems”, J.

ACM, vol.43, no.4, (1996), pp. 601-640.

[17] J. Hao and J. B. Orlin. "A Faster Algorithm for Finding the Minimum Cut of a Graph", In Proc. SODA pages 165-174, 1992

[18] Zeyun Yu and Chandrajit L. Baj, "Normalized Gradient Vector Diffusion and Image Segmentation", Lecture Notes In Computer Science; Vol. 2352. Proceedings of the 7th European Conference on Computer Vision-Part III, Pages: 517 - 530 Year of Publication: 2002.

[19] C. Xu, D. L. Pham, and J. L. Prince, "Medical Image Segmentation Using Deformable Models", SPIE Handbook on Medical Imaging -- Volume III: Medical Image Analysis, edited by J.M. Fitzpatrick and M. Sonka, May 2000.

[20] Ramani Pichumani, "Survey of Current Techniques", http://homepages.inf.ed.ac.uk/rbf/CVonline/LOCAL_COPIES/RAMANI1/node17.html, Jul 1997.

[21] Trolltech, Qt library, http://www.trolltech.com/company/index.html

[22] OpenGL Architecture Review Board, OpenGL Library, http://www.opengl.org

[23] Rafael C. Gonzalez, Richard E. Woods, “Digital Image Processing 2nd edition”, 2002

[24] R. Courant and D. Hibert, "Methods of Mathematical Physics". vol. 1. New York: Interscience, 1953.

16. Acknowledgements

I wish to acknowledged and thank those people who contributed to this project. I must express my deep gratitude to my advisor Prof. Eric N. Mortensen. First of all, I thank him for being my major professor and giving me the direction to my goal. His valuable feedback contributed greatly to this project.

I thank Dr. Timothy A. Budd and Dr. Thinh Nguyen for being my committee members and sparing time to read my document.

Lastly, I thank my parents for supporting me over years of my school time. Without their financial support and encouragements, I would not have been achieved my goal.

17. User manual

16.1 General

Fig.16.1 is the screenshot of the main window of the application. The application provides several image tools that apply on an image. Although the functionalities of the application are primary focused on the active contour mode, it provides functions to analysis an image as well. The following sections present brief explanations these functionalities along with their screenshot.

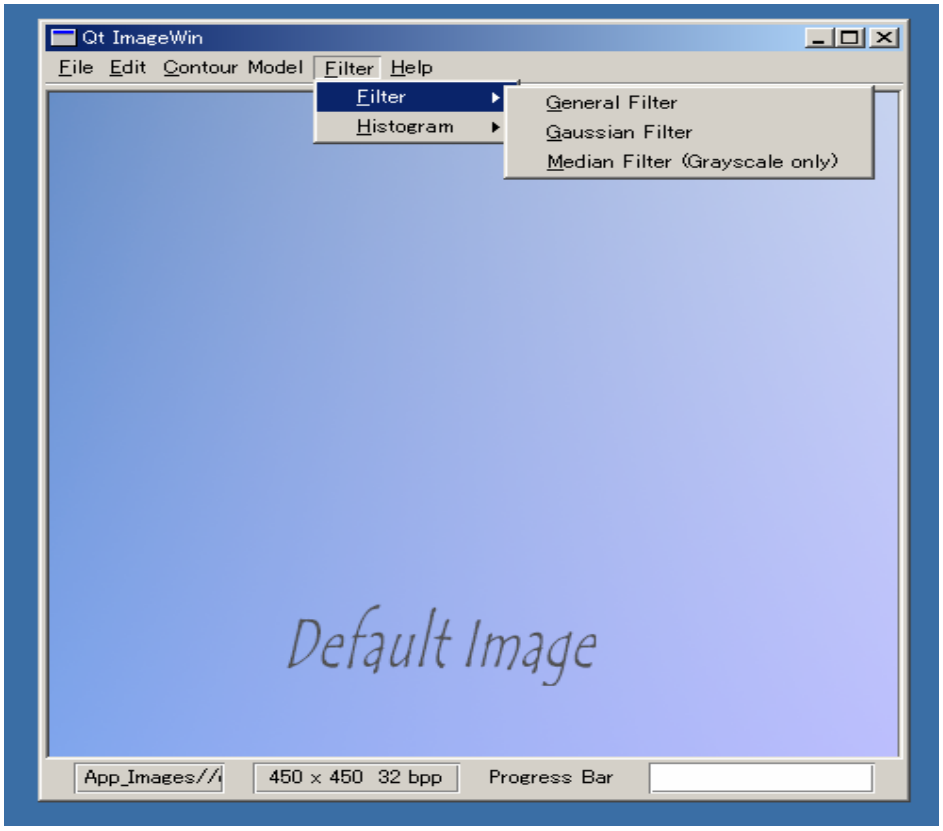


Fig. 16.1 Screenshot of the main window the application

16.2 Filters

A user can apply filters on an image. A filter is selected by clicking the tab bar at top of the filter control panel. A tab bar can switch between the parameter windows of General, Gaussian and Median filters. A filter is applied by clicking “Apply Filter” button on the Filter Control Panel. The operation can be undone by selecting the “Undo” button.

1. General Filter

A user can apply a user-defined filter by directly typing the coefficients of the mask of the filter in the *edit-line box* in the middle of the Filter Control Panel. It can retrieve the saved filter.

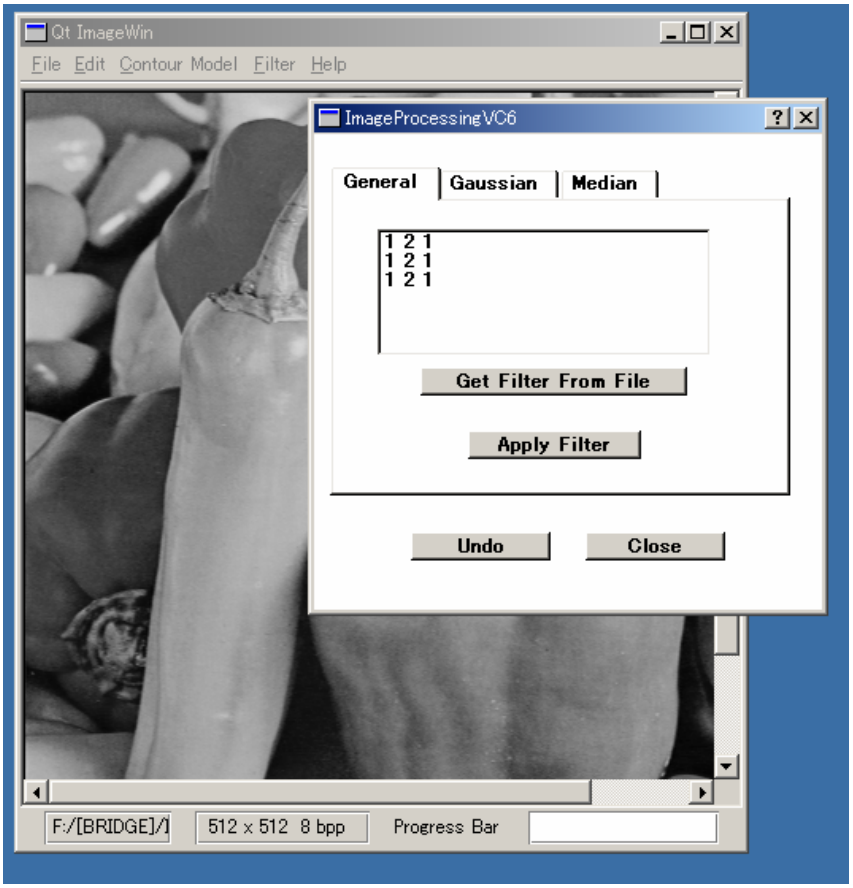


Fig. 16.2-1 Screenshot of General Filter

2. Gaussian Filter

A user can apply a Gaussian filter on the image. A size of the mask of Gaussian filter is set by a user. The size must be positive and odd integer. The coefficients of a Gaussian mask are viewed by clicking “View Filter” button. The popup window shows the matrix of the coefficients. It can closed by *double-click* anywhere on the popup window. The standard deviation of the Gaussian function is fixed to *1.0*.

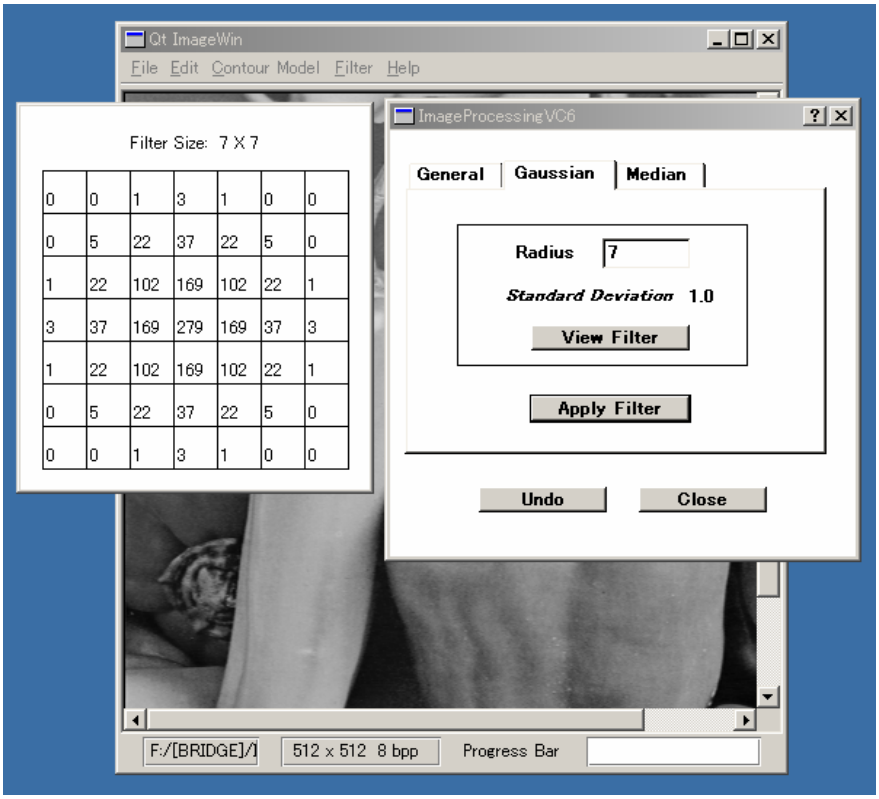


Fig. 16.2-2 Screenshot of Gaussian Filter

3. Median Filter

A user can apply a Median filter on an image. A size of the mask of a Median filter is set by a user. The size must be positive and odd integer. The Median filter can apply only on a *gray-scale* image.

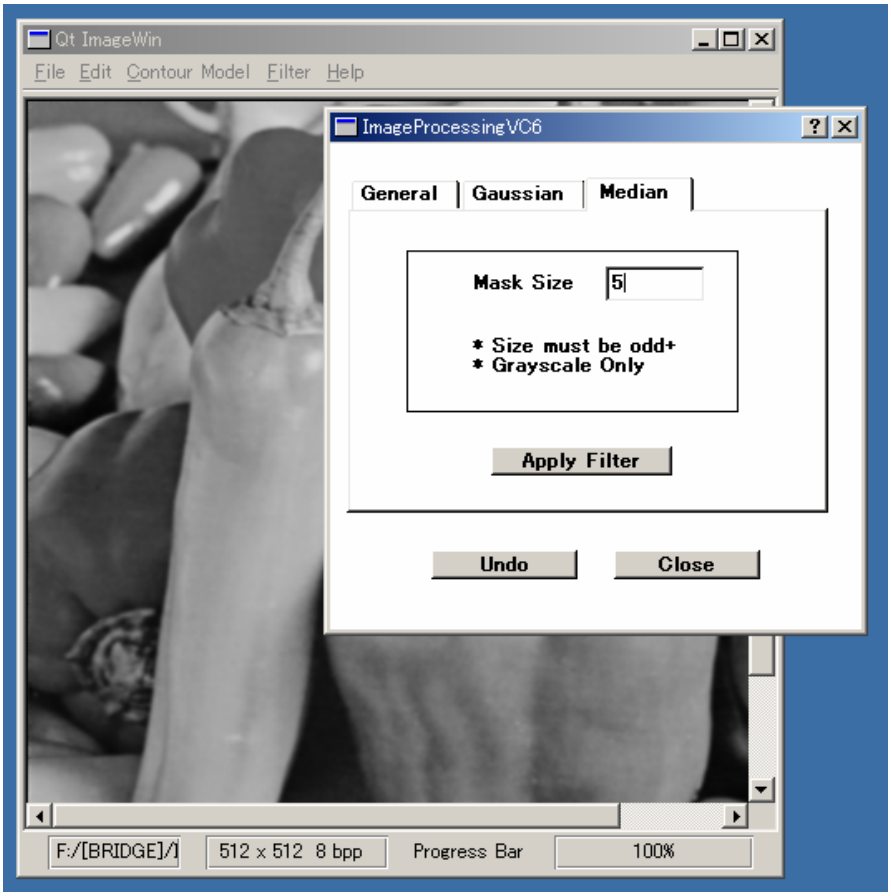


Fig. 16.2-3 Screenshot of Median Filter

16.3 Histogram

A user can view the histogram of an image. The plot at the top of the histogram control panel shows the frequency of the intensities of the image.

1. Histogram Equalization

Fig.16.3-1a and 16.3-1b show before and after the histogram equalization is applied on the image.

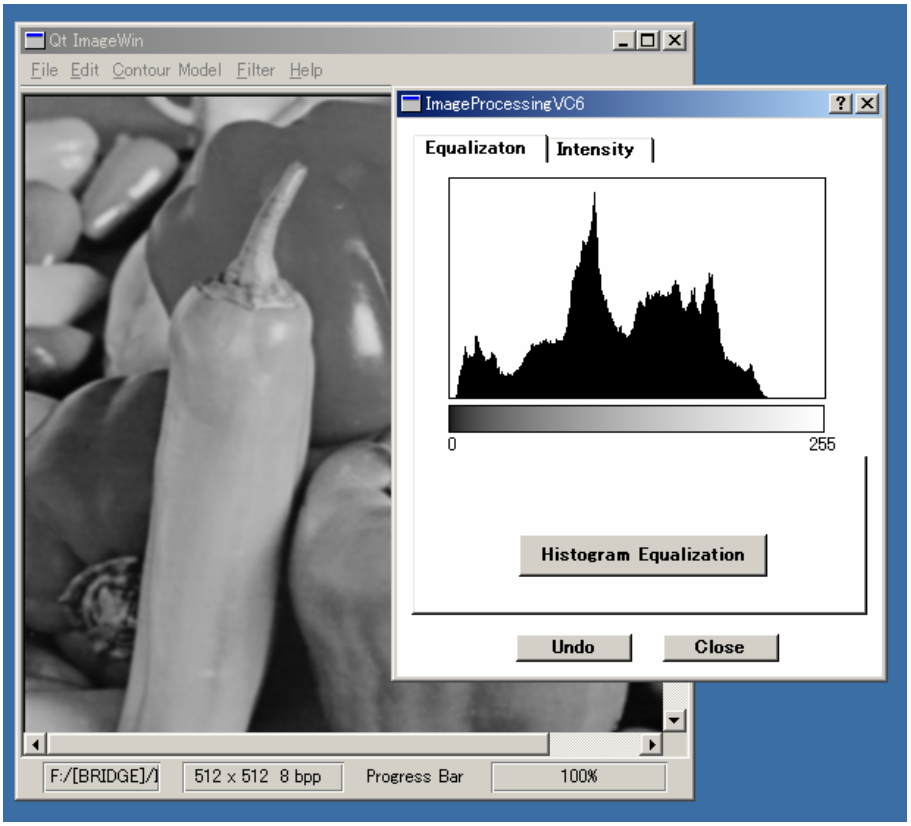


Fig. 16.3-1a Screenshot of Histogram Equalization

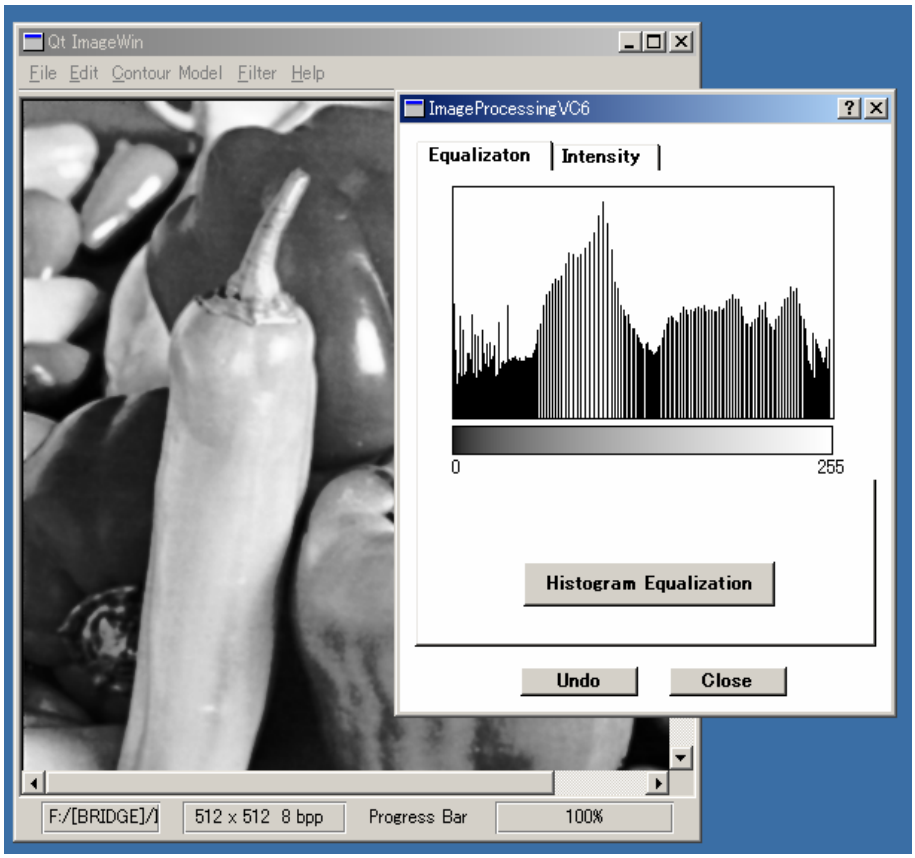


Fig. 16.3-1b Screenshot of Histogram Equalization after applying

2. Histogram Operations

The two operations, related the histogram, are provided. The “Shift” operation shifts the intensities of an image. A user can specify the range and the amount of the intensities of the image to shit. The “Cut” operation cuts the intensity of the image. A user can set the range of the intensities of the image.

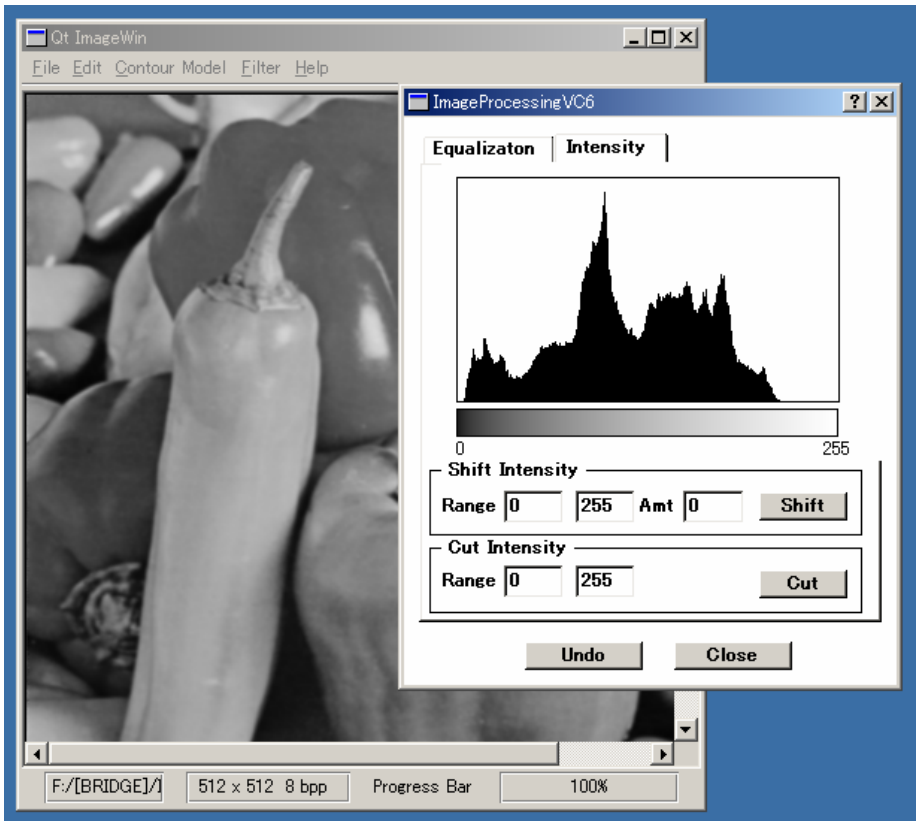


Fig. 16.3-2 Screenshot of operations

16.4 Image Properties

1. Information of an image

The Image Information window (Fig.16.4-1) shows the information of an image. It shows the information such as name, location, color depth and the frequency of intensities of the image. The Image Property window is shown by selecting the menu item “Image Property” under edit on menu bar.

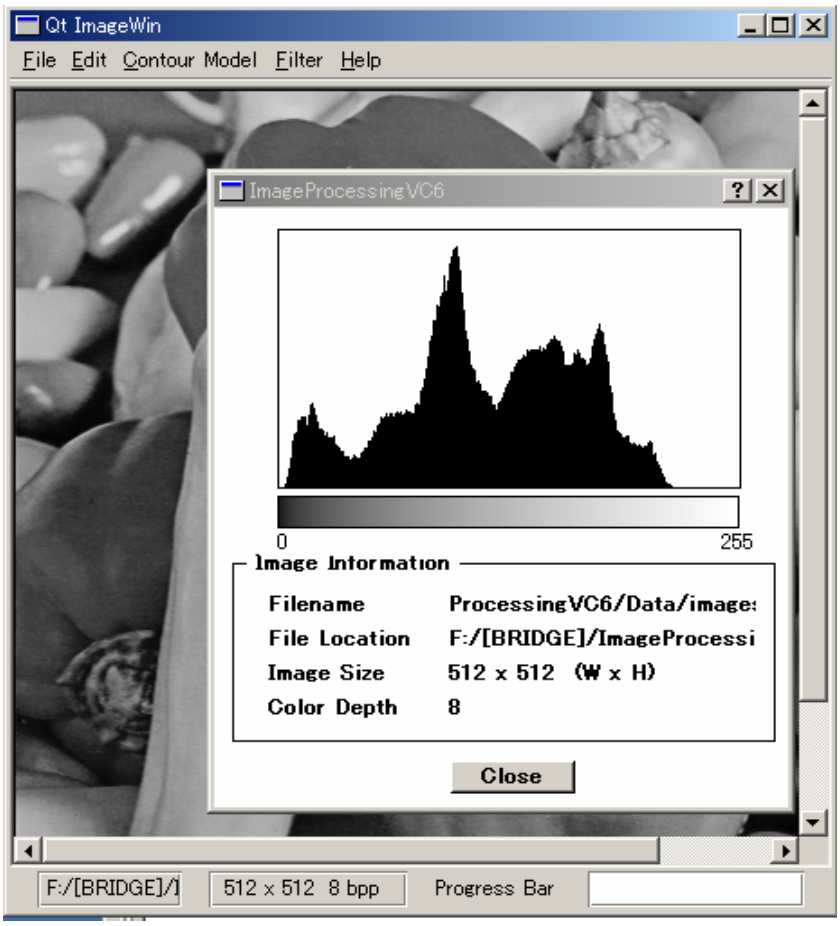


Fig. 16.4-1 Screenshot of the information of an image

2. Gradient and GVF field of an image

Fig.16.4-2a and 16.4-2b show the OpenGL window and the OpenGL Control Panel. The list of check boxes on the control panel controls the viewing layers on the OpenGL window. Each component is shown or hidden by marking or unmarking a check box next to it. The OpenGL window and its control panel for *Gradient* is shown by selecting the menu item, "Show OpenGL dialog" in the *right-click* menu on the main window. Meanwhile, the OpenGL window and its control panel for the *GVF field* is shown by selecting the menu item, "Show OpenGL dialog", in the *right-click* menu on the active contour models window. The center of the OpenGL window is the location of the mouse on the image when the menu item "Show OpenGL dialog" is clicked.

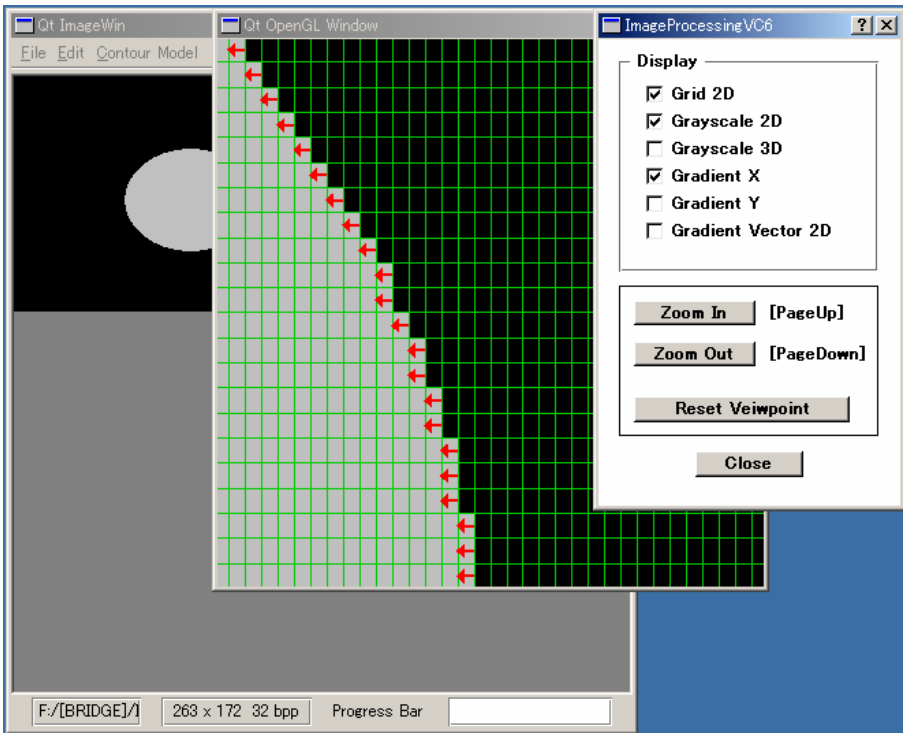


Fig. 16.4-2a Screenshot of Gradient of an image

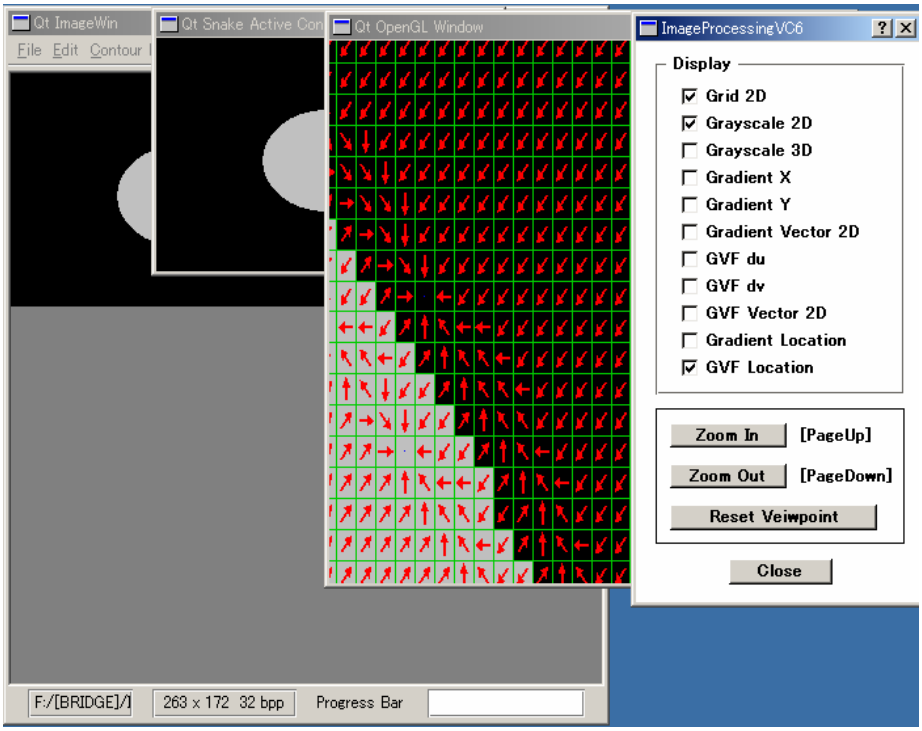


Fig. 16.4-2b Screenshot of GVF field of an image

16.5 Active Contour Models

A user can select a method of the active contour models. Two types of active contour models are provided; the snake active contour models and the GCBAC. The user can select one of them by selecting the menu items under "Active Contour Model" on the menu bar.

1. Snake active contour models

The control panel in Fig. 16.5-1 is used to set parameters for the original snake, the adaptive snake and the GVF snake. Table 16.5-1 lists functionality of each section on the Snake Control Panel.

Sections	Functions
Coefficients	· Adjust weights of the internal and external energies of the snake.
Snaxels	· Set interval of the snake control points on the contour.
Snake	· Select an algorithm of snake · Rollback the previous iteration of the snake · Save the current contour · Retrieve the saved contour
Original snake	· Perform the original snake.
Adaptive snake	· Set the adaptive force in pixel. · Adjust threshold on the image force. · Perform the Adaptive snake.
GVF snake	· Set the value of the ***mu*** · Set the number of iteration to compute GVF field · Perform the GVF snake. · Update GVF field. It updates GVF field only if the number of iteration has been changed.
Termination Criteria	· Select a termination criterion. · Show the current number of iterations. · Show the message how the algorithm of the snake has been terminated.
Other [Right bottom of the	· Trace back an active contour. · Perform the deformation process by single iteration. · Extract multiple objects

Snake Control Panel]	· Close the Snake Control Panel

Table 16.5-1 List of functionalities on the Snake Control Panel

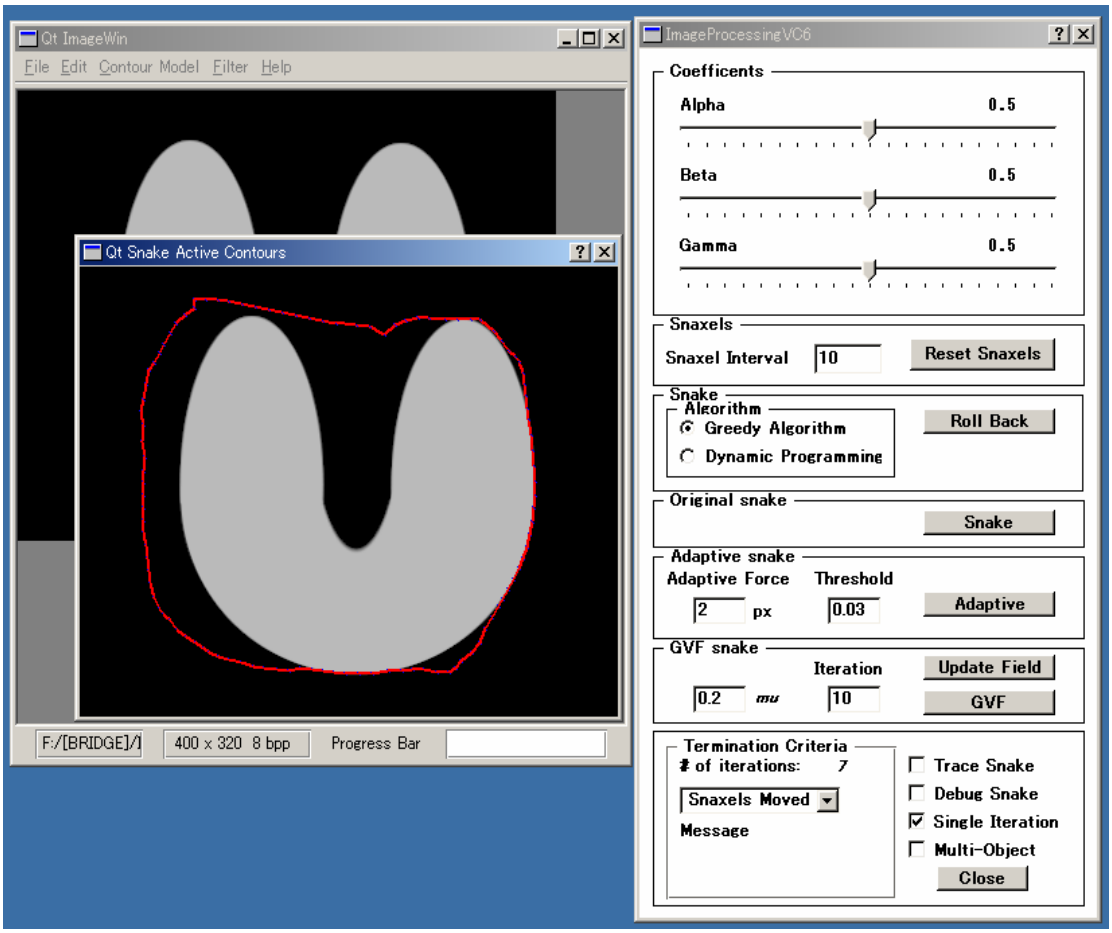


Fig. 16.5-1 Screenshot of Snake Control Panel

2. GCBAC

Fig.16.5-2 shows the GCBAC window. The size of CN (contour neighbor) is specified by a user at the bottom of the GCBAC window.

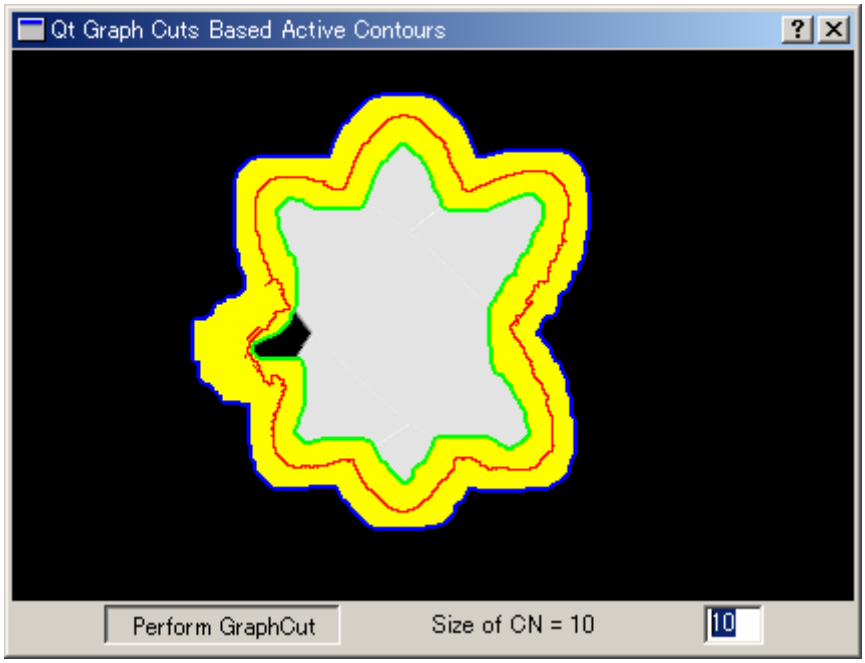


Fig. 16.5-2 Screenshot of GCBAC