EdgeFlow: A Technique for Boundary Detection and Image Segmentation

Wei-Ying Ma and B. S. Manjunath

Department of Electrical and Computer Engineering
University of California,
Santa Barbara, CA 93106-9560
E-mail: wei@hpl.hp.com, manj@ece.ucsb.edu

Abstract

A novel boundary detection scheme based on "edge flow" is proposed in this paper. This scheme utilizes a predictive coding model to identify the direction of change in color and texture at each image location at a given scale, and constructs an edge flow vector. By iteratively propagating the edge flow, the boundaries can be detected at image locations which encounter two opposite directions of flow in the stable state. A user defined image scale is the only significant control parameter that is needed by the algorithm. The scheme facilitates integration of color and texture into a single framework for boundary detection.

1 Introduction

In most computer vision applications, the edge/boundary detection and image segmentation constitute a crucial initial step before performing high-level tasks such as object recognition and scene interpretation. In the image analysis literature, typically segmentation performance was demonstrated on a very small example set of images. Large scale image database annotation demands robustness with very little parameter tuning over a wide range of image data.

While considerable research and progress have been made in the area of image segmentation, the robustness and generality of the algorithms on a large variety of image data have not been established. One of the difficulties arises from the fact that most natural images are usually made of various types of boundaries created by changes in cues such as color, texture, or phase. Thus the algorithm needs to consider all

these different types of image attributes together in order to segment real natural images. Furthermore, image segmentation itself is an ill-posed problem. It often requires additional information from the user in order to select a proper scale for detecting edges and boundaries, and thus, segmenting the regions of interest. For example, Figure 1(a) shows an image which contains five different "beans" regions. One might consider each bean as an individual object and obtain a result similar to Figure 1(b), or might consider each "beans" region as a texture and get a segmentation like in Figure 1(c).

In order to develop a segmentation algorithm which is capable of processing large and diverse collections of images, a general framework of boundary detection and image segmentation called "*edge flow*" is proposed in this paper. This framework utilizes a predictive coding model to identify and integrate the direction of change in various types of image attributes (color, texture, and phase discontinuity) at each image location, and constructs an edge flow vector which points to the closest image boundary. By iteratively propagating the edge flow, the boundaries can be detected at image locations which encounter two opposite directions of flow in the stable state. Furthermore, this whole process including image smoothing and feature extraction is designed in a way that it can be controlled by a scale parameter and, therefore, the algorithm can adapt itself to satisfy the user's preference in segmenting the regions of interest. Figure 1(b) and (c) actually show the results of the proposed algorithm by selecting different scale parameters to obtain appropriate segmentations.

2 Previous Work

In this section we briefly review the previous work on edge/boundary detection and image segmentation. Previous work on segmentation can be broadly classified into two main categories: the first one focuses on the detection and localization of intensity/color discontinuities, and the second one considers the partition of an image into homogeneous regions.

2.1 Edge Detection

Much of the research on edge detection has been devoted to the development of optimal edge detectors which provide the best trade-off between the detection and localization performance [4, 6, 18, 21]. A

common strategy in designing such edge operators is to find the filter which optimizes the performance with respect to the three criteria: good detection, good localization, and a unique response to a single edge. In [4] Canny showed that the optimal detector can be approximated by the first derivative of a Gaussian. By convolving the image with this filter, the edge detection is equivalent to finding the maxima in gradient magnitude of a Gaussian-smoothed image in the appropriate direction.

Detecting and combining edges at multiple resolutions and scales is another important issue in edge detection [4, 18, 25]. The scale-space technique introduced by Witkin [25] involves generating coarser resolution images by convolving the original images with a Gaussian smoothing kernel.

The regularization theory has also been frequently used in helping the design of edge detection algorithms [24]. Gökmen and Jain [9] recently proposed an image representation called the λ_τ-space representation using this theory. Based on this model, they develop a generalized edge detector which encompasses most of the well-known edge detectors under a common framework.

2.2 Texture Segmentation

The goal of texture segmentation is to partition an image into homogeneous regions and identify the boundaries which separate regions of different textures. Segmentation is obtained either by considering a gradient in the texture feature space [8, 12, 15], or by unsupervised clustering [1, 7, 10], or by texture classification [17]. Segmentation by labelling often suffers from a poor localization performance because of the conflicting requirements of region labeling and boundary localization in terms of the observation neighborhood (window size) [26]. Unsupervised clustering/segmentation requires an initial estimate of the number of the regions in the image, which is obtained mostly by setting a threshold in the feature clustering algorithm. However, estimating the number of regions is a difficult problem and the results are usually not reliable.

The idea of anisotropic diffusion has also been recently utilized to detect texture boundaries by coalescing texture features in image feature space [20]. The strategy that is used in [20] is to extend the notion of edge-preserving smoothing and anisotropic diffusion from image intensities to feature vectors that

describe the textural content of each image patch. The diffusion encourages intraregion feature smoothing in preference to interregion feature smoothing. Thus, the feature difference across two texture regions will remain “sharp” during the smoothing process.

2.3 Edge Flow and Anisotropic Diffusion

The edge flow scheme proposed in the following has certain similarities to anisotropic diffusion methods discussed in [19, 20]. We summarize the basic differences between the two approaches in below.

First, the scale parameter in edge flow is mainly controlled by users to decide what scale should be used to distinguish object and texture, and therefore, appropriate texture features can be computed. Although a single parameter is used in edge flow, it effectively controls a range of scales for texture. Even though the scale affects smoothing, no specific noise model or blur estimation is used in scale selection.

Secondly, the local edge energy is iteratively propagated to one of its neighbors in the edge flow model. The direction of local edge flow is determined to point to the closest boundaries based on the prediction errors, and the goal of this flow propagation is to accumulate the local edge energies toward their closest image boundaries. In contrast, in anisotropic diffusion the local image intensity is diffused to all its neighbors when the variance of Gaussian smoothing kernel increases. A spatially varying diffusion coefficient controls the rate at which diffusion occurs near the edges.

Finally, to the best of our knowledge, the performance of diffusion type algorithms has not been demonstrated on any large image data sets.

3 “Edge Flow”

In this section the general concept of edge flow is first outlined and a detailed implementation of it is illustrated and explained. Traditionally edges are located at the local maxima of the gradient in intensity/ image feature space. In contrast, in our approach the detection and localization of edges (or image boundaries in a more general sense) are performed indirectly: first by identifying a flow direction at each pixel location that points to the closest boundary; then followed by the detection of locations that encounter two opposite directions of edge flow. Since any of the image attributes such as color, texture, or their combina-

tion can be used to define the edge flow, this scheme provides a general framework for integrating different types of image information for boundary detection.

3.1 Definition of the Edge Flow

Let us define the general form of edge flow vector F at image location s with an orientation θ as:

$$F(s, \theta) = \boldsymbol{F}[E(s, \theta), P(s, \theta), P(s, \theta + \pi)] \quad (1)$$

where

- $E(s, \theta)$ is the edge energy at location s along the orientation θ.
- $P(s, \theta)$ represents the probability of finding the image boundary if the corresponding flow at location s "flows" in the direction θ.
- $P(s, \theta + \pi)$ represents the probability of finding the image boundary if the corresponding flow at location s flows backwards, i.e., in the direction $\theta + \pi$.

The first component $E(s, \theta)$ of edge flow is used to measure the energy of local image information change (such as intensity/color, texture, and phase difference), and the remaining two components $P(s, \theta)$ and $P(s, \theta + \pi)$ are used to represent the probability of flow direction. The basic steps for detecting image boundaries is summarized as follows:

- At each image location, we first compute its local edge energy and estimate the corresponding flow direction.
- The local edge energy is iteratively propagated to its neighbor if the edge flow of the corresponding neighbor points in a similar direction.
- The edge energy stops propagating to its neighbor if the corresponding neighbor has an opposite direction of edge flow. In this case, these two image locations have both their edge flows pointing at each other indicating the presence of a boundary between the two pixels.

- After the flow propagation reaches a stable state, all the local edge energies will be accumulated at the nearest image boundaries. The boundary energy is then defined as the sum of the flow energies from either side of the boundary.

Some Definitions

A two dimensional isotropic Gaussian function is defined as

$$G_\sigma(x, y) = (1/(\sqrt{2\pi}\sigma))\exp[-(x^2 + y^2)/2\sigma^2]. \tag{2}$$

The first derivative of Gaussian (GD) along the x-axis is given by

$$GD_\sigma(x, y) = \frac{\partial G_\sigma(x, y)}{\partial x} = -\frac{x}{\sigma^2}G_\sigma(x, y), \tag{3}$$

and the difference of offset Gaussian (DOOG) along the x-axis is defined as:

$$DOOG_\sigma(x, y) = G_\sigma(x, y) - G_\sigma(x + d, y) \tag{4}$$

where d is the offset between centers of two Gaussian kernel and is chosen proportional to σ. By rotating these two functions, we generate a family of the Gaussian derivative and the difference of offset Gaussian functions along different orientations θ:

$$GD_{\sigma,\theta}(x, y) = GD_\sigma(x', y'), \tag{5}$$

$$DOOG_{\sigma,\theta}(x, y) = DOOG_\sigma(x', y'),$$

$$x' = x\cos\theta + y\sin\theta,\ y' = -x\sin\theta + y\cos\theta.$$

Note that the parameter σ is clearly denoted for all the previous functions. As can be seen later, this parameter will correspond to the scale (or resolution) level at which the boundary detection and image segmentation are conducted.

3.2 Intensity Edge Flow

Computing $E(s, \theta)$:

Now consider an image at a given scale σ as $I_\sigma(x, y)$, which is obtained by smoothing the original image $I(x, y)$ with a Gaussian kernel $G_\sigma(x, y)$. The scale parameter will control both the edge energy computation and the local flow direction estimation, so that only edges larger than the specified scale are

detected. The edge flow energy $E(s, \theta)$ at scale σ is defined to be the magnitude of the gradient of the smoothed image $I_\sigma(x, y)$ along the orientation θ:

$$E(s, \theta) = \left| \frac{\partial}{\partial \boldsymbol{n}} I_\sigma(x, y) \right| = \left| \frac{\partial}{\partial \boldsymbol{n}} [I(x, y) * G_\sigma(x, y)] \right| = \left| I(x, y) * \frac{\partial}{\partial \boldsymbol{n}} G_\sigma(x, y) \right| \tag{6}$$

where $s = (x, y)$ and $\boldsymbol{n}$ represents the unit vector in the gradient direction. We can rewrite (6) as

$$E(s, \theta) = \left| I(x, y) * GD_{\sigma, \theta}(x, y) \right|. \tag{7}$$

This edge energy indicates the strength of the intensity change. Many existing edge detectors actually use similar operations to identify the local maxima of intensity changes as edges. The distinguishing part of the edge flow model is that the edge energy is represented as a flow vector by assigning probabilities to its flow directions. Boundary detection itself is formulated as a dynamic process wherein the local edge energies *flow* in the direction of most probable image boundaries closest to the corresponding locations.

Computing $P(s, \theta)$:

For each of the edge energy along the orientation θ at location s, we now consider two possible flow directions; the forward (θ) and the backward $(\theta + \pi)$, and estimate the probability of finding the nearest boundary in each of these directions. These probabilities can be obtained by looking into the prediction errors toward the surrounding neighbors in the two directions. Consider the use of image information at location s to predict its neighbor in the direction θ. Ideally they should have similar intensity if they belong to the same object and the prediction error can thus be computed as

$$\begin{aligned} Error(s, \theta) &= \left| I_\sigma(x + d\cos\theta, y + d\sin\theta) - I_\sigma(x, y) \right| \\ &= \left| I(x, y) * DOOG_{\sigma, \theta}(x, y) \right| \end{aligned} \tag{8}$$

where d is the distance of the prediction, and which should be proportional to the scale at which the image is being analyzed. In the experiments we choose $d = 4\sigma$. A large prediction error in a certain direction implies a higher probability of finding a boundary in that direction. Therefore, the probabilities of edge flow direction are assigned in proportion to their corresponding prediction errors:

$$P(s, \theta) = \frac{Error(s, \theta)}{Error(s, \theta) + Error(s, \theta + \pi)} \tag{9}$$

The idea of this approach to computing the flow probabilities comes from [5, 23]. It has been suggested that the human visual system uses a predictive coding model to process image information. This model (although the details vary) has been successfully used in interpreting many vision phenomena such as the retinal inhibitory interactions [23], and the coding of textured patterns [5].

Figure 2 shows the computation of $E(s,\theta)$ and $P(s,\theta)$ using the GD and the DOOG functions. Notice the relative positioning of the two DOOG filters with respect to a given pixel location (x,y). As mentioned earlier, this offset depends on the scale parameter σ.

Figure 3 shows a comparison of the edge flow model with the conventional approaches to detecting edges. Instead of seeking the local maxima of the intensity gradient magnitude (or finding the zero-crossings of the second derivative of image intensity), we construct the flow vectors whose energy is equivalent to the magnitude of the intensity gradient and whose direction is estimated by the prediction errors. As can be seen from Figure 3(b), the edge flows on the right side of boundary all have their directions pointing to the left because $P(left) > P(right)$ in that region, and the edge flows on the left side all point to the right because of $P(right) > P(left)$. After the flow is propagated (see Section 5) and reaches a stable state, the edge locations are identified as those places where two opposite edge flows meet each other, and the boundary energy is equal to the integration of the gradient magnitude (shaded area). This example illustrates that the edge flow model gives identical results as a zero crossing for noise-free step edges (this result can also be easily derived analytically using (7)-(9)). However, real images usually do not contain such ideal edges.

3.3 Texture Edge Flow

Much of the same formulation of Section 3.2 for intensity edges carries over to image attributes such as color and texture. In this section we consider textured images and compute the texture edge flow using the directional gradient in the texture feature maps.

The texture features are extracted based on a Gabor wavelet decomposition scheme proposed in [16]. However, in contrast with the use of a fixed set of Gabor filters for computing the texture features, the bank

of Gabor filters used here are generated according to the scale parameter σ specified by the user. This parameter defines the resolution at which the image boundaries are considered. Therefore, only the patterns with sizes smaller than that scale are considered as elements of a texture (i.e., texels), and anything larger than that scale will be treated as an object.

The strategy for Gabor filter bank design proposed in [16] is particularly useful for this purpose. Given the scale parameter σ, define the lowest center frequency U_l of the Gabor filters to be $1/(4\sigma)$ cycles/pixel. This value is based on the consideration of the Gaussian smoothing window and the distance $d = 4\sigma$ used in computing the prediction error, so that the window size covers at least one cycle of the lowest spatial frequency. Furthermore, the highest center frequency U_h is set to 0.45 cycles/pixel. The number of scales S in the filters is determined according to the lowest center frequency U_l so that the filters cover the spectrum appropriately. S ranges from 1 for the small scale to 5 for the large scale. The number of orientations K is fixed to 6 in the experiments. Figure 4 shows the Fourier transforms of the Gabor filter banks which are generated for different value of σ. See Appendix A for a more detailed explanation of the Gabor filter bank design.

The complex Gabor filtered images can be written as:

$$O_i(x, y) = I(x, y) * g_i(x, y) = m_i(x, y) \exp[\phi_i(x, y)] \tag{10}$$

where $1 \le i \le N$, $N = S \cdot K$ is the total number of filters, $m(x, y)$ is the amplitude, and $\phi(x, y)$ is the phase. By taking the amplitude of the filtered output across different filters at the same location (x, y), we form a texture feature vector

$$\Psi(x, y) = [m_1(x, y), m_2(x, y), \ldots, m_N(x, y)] \tag{11}$$

which characterizes the local spectral energies in different spatial frequency bands. For most of the textured regions, this feature vector is good enough for distinguishing their underlying pattern structure. Some exceptions are illusory boundaries such as the ones in Figure 8(c). In this case, the phase information $\{\phi(x, y)\}$ has to be incorporated in order to detect the discontinuity. We will discuss this in Section 3.4. In

the following, let us first consider the formulation of edge flow $F(s, \theta)$ using the texture features Ψ. The texture edge energy, which is used to measure the change in local texture information, is given by

$$E(s, \theta) = \sum_{1 \le i \le N} \left| m_i(x, y) * GD_{\sigma, \theta}(x, y) \right| \cdot w_i \tag{12}$$

where $w_i = 1 / \left\| \alpha_i \right\|$ and $\left\| \alpha_i \right\|$ is the total energy of the subband i. The weighting coefficients w_i normalize the contribution of edge energy from the various frequency bands.

Similar to the intensity edge flow, the direction of texture edge flow can be estimated based on the texture prediction error at a given location:

$$Error(s, \theta) = \sum_{1 \le i \le N} \left| m_i(x, y) * DOOG_{\sigma, \theta}(x, y) \right| \cdot w_i \tag{13}$$

which is the weighted sum of prediction errors from each texture feature map. Thus, the probabilities $P(s, \theta)$ and $P(s, \theta + \pi)$ of the flow direction can be estimated using (9).

3.4 Edge Flow Based on Gabor Phase

In this section, the phase information of Gabor filter output is used to construct an edge flow field for detecting boundaries. We have not found much use of phase information at this time on real images, but the scheme does detect very accurately the illusory boundaries shown in Figure 8 and Figure 9.

From (10), the complex Gabor filtered image can be written as

$$O(x, y) = Re(x, y) + j \cdot Im(x, y) \tag{14}$$

where $Re(x, y)$ and $Im(x, y)$ represent the real and imaginary parts of Gabor filtered output, respectively. The phase of the filtered image can be expressed as:

$$\phi(x, y) = \operatorname{atan}[Im(x, y) / Re(x, y)]. \tag{15}$$

This phase information will contain discontinuities at $\pm\pi$ because the operation of inverse tangent only provides the principal value of the phase. In order to compute $\phi(x, y)$ without discontinuity, phase unwrapping is required. A general strategy for solving the unwrapping problem is to add or subtract 2π from the part of phase function that lies after a discontinuity. However, this phase unwrapping problem can become very difficult if too many zero points (both the real and imaginary parts are zero here, and therefore, the phase is undefined) are in the image [23].

The unwrapped phase can be decomposed into a global linear phase component and a local phase component. The local phase contains information about the locations where the texture property changes. In other words, within a uniform textured region, the phase $\phi(x, y)$ will vary linearly, and it changes its varying rate when a boundary between different texture regions is crossed. As a result, the local phase has been used in many texture segmentation schemes [1, 2, 3].

In order to compute the edge flow field using the phase information, there are two problems that we have to consider here. First, we have to compute the phase derivatives without unwrapping the phase. Second, instead of just using the DOOG functions to compute the prediction error, we have to include a first-order predictor to compensate for the global linear phase component.

Consider the formula

$$\frac{d}{dx}\text{atan}(x) = 1/(1+x^2). \tag{16}$$

Assuming the derivative of the phase exists everywhere, we can compute the phase derivative using the following equation without going through the phase unwrapping procedure:

$$\frac{\partial}{\partial x}\phi(x, y) = \text{imag}\left[O^*(x, y)\cdot\frac{\partial}{\partial x}O(x, y)\right]/m(x, y)^2 \tag{17}$$

where $*$ is complex conjugate. The phase derivative with respect to any arbitrary orientation can be computed in a similar manner.

Without loss of the generality, we first consider the design of a linear phase predictor along the x axis

$$\hat{\phi}(x+a, y) = \phi(x, y) + a\cdot\frac{\partial}{\partial x}\phi(x, y), \tag{18}$$

and therefore, the prediction error is equal to

$$Error = \phi(x+a, y) - \phi(x, y) - a\cdot\frac{\partial}{\partial x}\phi(x, y). \tag{19}$$

However, because the first two terms in equation (19) are wrapped phases, the prediction error has to be further corrected by adding or subtracting 2π such that it always lies between $-\pi$ and π. Because the linear component of the phase has been removed by the first-order predictor, the magnitude of the prediction

error is usually much smaller than π. As a result, the prediction error contributed by the 2π phase wrapping can be easily identified and corrected. The general form of computing the phase prediction error can be written as

$$Error(s,\theta) = \left|\phi(x + a\cdot\cos\theta, y + a\cdot\sin\theta) - \phi(x,y) - a\cdot\frac{\partial}{\partial n}\phi(x,y) + 2\pi k(x,y)\right| \tag{20}$$

where $n = (\cos\theta, \sin\theta)$ and $k(x,y)$ is an integer which ensures that the prediction error is always between $-\pi$ and π. One can use the second derivative of the phase to compute the corresponding phase edge energy. However, for simplicity in implementation, we directly use the prediction error to represent the phase "edge" energy.

4 Edge Flow Integration

4.1 Combining Different Types of Edge Flows

The edge flows obtained from different types of image attributes can be combined together to form a single edge flow field for boundary detection. Consider

$$\boldsymbol{E}(s,\theta) = \sum_{a\in A} E_a(s,\theta)\cdot w(a), \text{ and } \sum_{a\in A} w(a) = 1 \tag{21}$$

$$\boldsymbol{P}(s,\theta) = \sum_{a\in A} P_a(s,\theta)\cdot w(a) \tag{22}$$

where $E_a(s,\theta)$ and $P_a(s,\theta)$ represent the energy and probability of the edge flow computed from image attribute a, $a \in \{\text{intesity/color, texture, and phase}\}$. $w(a)$ is the weighting coefficient associated with image attribute a.

Now let us consider the use of combined color and texture information for boundary detection. For a given color image, the intensity edge flow can be computed in each of three color bands (R, G, B) using (7), (8), and (9), and the texture edge flow can be calculated from the illuminance $I = (R+G+B)/3$. Then the overall edge flow can be obtained by combining them as in (21) and (22) with $A = \{\text{red, green, blue, texture}\}$. In the following experiments $w(\text{texture}) = 0.4$ and $w(\text{red}) = w(\text{green}) = w(\text{blue}) = 0.2$.

4.2 Combining Edge Flows from Different Directions

In the example of Figure 3, the final direction of edge flow at each location is simply determined by selecting the direction with larger probability because there are only two possible directions to be considered in the 1-D case. However, for a given image, the computed edge flows can range from 0 to π. In order to identify the best direction for searching for the nearest boundary, the following scheme is used:

Suppose we have edge flows $\{\boldsymbol{F}[E(s,\theta), P(s,\theta), P(s,\theta+\pi)]\big|_{0 \le \theta < \pi}\}$, we first identify a continuous range of flow directions which maximizes the sum of probabilities in that half plane:

$$\Theta(s) = \arg\max_{\theta} \left\{ \sum_{\theta \le \theta' < \theta + \pi} P(s,\theta') \right\} \tag{23}$$

Then, the final resulting edge flow is defined to be the vector sum of the edge flows with their directions in the identified range, and is given by

$$\vec{\boldsymbol{F}}(s) = \sum_{\Theta(s) \le \theta < \Theta(s) + \pi} E(s,\theta) \cdot \exp(j\theta), \tag{24}$$

where $\vec{\boldsymbol{F}}(s)$ is a complex number with its magnitude representing the resulting edge energy and angle representing the flow direction. Figure 5(a)-(b) show an example of the final edge flows after combining different directions of edge flows (scale $\sigma = 2$ pixels). As can be seen, the direction of each local edge flow points to its nearest boundary.

5 Edge Flow Propagation and Boundary Detection

After the edge flow $\vec{\boldsymbol{F}}(s)$ of an image is computed, boundary detection can be performed by iteratively propagating the edge flow and identifying the locations where two opposite direction of flows encounter each other. At each location, the local edge flow is transmitted to its neighbor in the direction of flow if the neighbor also has a similar flow direction (the angle between them is less than 90 degrees). The steps are:

1. Set $n = 0$ and $\vec{\boldsymbol{F}}_0(s) = \vec{\boldsymbol{F}}(s)$.

2. Set the initial edge flow $\vec{\boldsymbol{F}}_{n+1}(s)$ at time $n+1$ to zero.

3. At each image location s, identify the neighbor $s' = (x', y')$ which is in the direction of edge flow $\vec{\boldsymbol{F}}_n(s)$, i.e., $\angle\vec{\boldsymbol{F}}_n(s) = \text{atan}\left(\frac{y'-y}{x'-x}\right)$.

4. Propagate the edge flow if $\vec{\boldsymbol{F}}_n(s') \cdot \vec{\boldsymbol{F}}_n(s) > 0$: $\vec{\boldsymbol{F}}_{n+1}(s') = \vec{\boldsymbol{F}}_{n+1}(s') + \vec{\boldsymbol{F}}_n(s)$; otherwise the edge flow stay at its original location, $\vec{\boldsymbol{F}}_{n+1}(s) = \vec{\boldsymbol{F}}_{n+1}(s) + \vec{\boldsymbol{F}}_n(s)$.

5. If nothing has been changed, stop the iteration. Otherwise, set $n = n + 1$ and go to the step 2 and repeat the process.

Once the edge flow propagation reaches a stable state, we can detect the image boundaries by identifying the locations which have non-zero edge flows coming from two opposing directions. Let us first define the edge signals $V(x, y)$ and $H(x, y)$ as the vertical and horizontal edge maps between image pixels as shown in Figure 6(a), and let

$$\boldsymbol{F} = (h(x, y), v(x, y)) = (\text{real}(\vec{\boldsymbol{F}}(s)), \text{imag}(\vec{\boldsymbol{F}}(s))) \tag{25}$$

be the final stable edge flow (see Figure 6(b)). Then, the edge signals $V(x, y)$ and $H(x, y)$ will be turned on if and only if the two neighboring edge flows point at each other. Once the edge signal is on, its energy is defined to be the summation of the projections of those two edge flows towards it. Summarizing:

- Turn on the edge $V(x, y)$ if and only if $h(x-1, y) > 0$ and $h(x, y) < 0$; then

$$V(x, y) = h(x-1, y) - h(x, y).$$

- Turn on the edge $H(x, y)$ if and only if $v(x, y-1) > 0$ and $v(x, y) < 0$; then

$$H(x, y) = v(x, y-1) - v(x, y).$$

Figure 6(c) shows an example of boundary detection. Note that only the edge signals with two opposite directions of flow from their neighboring pixels are turned on.

After the edge signals are detected, the connected edges are used to form a boundary, whose energy is defined to be the average of its edge signals $V(x, y)$ and $H(x, y)$. A certain threshold for the energy is used to remove weak boundaries.

Figure 5(c) shows an example of the edge flow propagation using the flower image. As can be seen, the edge flows are concentrated only along the two sides of the image boundaries with their flow directions

pointing at each other. Figure 5(d) illustrates the result of boundary detection after turning on the edge signals between two opposite directions of edge flows.

6 Boundary Connection and Region Merging

After boundary detection, disjoint boundaries are connected to form closed contours and result in a number of image regions. The basic strategy for connecting the boundaries are summarized as follows.

- For each open contour, we associate a neighborhood search size proportional to the length of the contour. This neighborhood is defined as a half circle with its center located at the unconnected end of the contour.
- The nearest boundary element which is within the half circle is identified.
- If such a boundary element is found, a smooth boundary segment is generated to connect the open contour to the nearest boundary element.
- This process is repeated few times (typically 2-3 times) till all salient open contours are closed.

At the end, a region merging algorithm is used to merge similar regions based on a measurement that evaluates the distances of region color and texture features, the sizes of regions, and the percentage of original boundary between the two neighboring regions. This algorithm sequentially reduces the total number of regions each time by checking if the user's preferred number has been approached to the best extent. Figure 7(a)-(b) shows an example of boundary detection after the edge flow propagation. The disjointed boundaries are connected to form closed contours and result in an initial image segmentation as shown in Figure 7(c). This initial segmentation is further processed by the region merging algorithm to group similar regions, and the final segmentation result is illustrated in Figure 7(d).

7 Experimental Results

Figure 8 shows three images which contain intensity, texture, and illusory boundaries respectively. The scheme described in Section 3 is used to construct the edge flow field for these different image attributes. The final segmentation results after the post-processing (edge flow propagation, boundary detection, boundary connection, and region merging) are illustrated. As can be seen, these three images, which

traditionally require different algorithms to segment, can now be processed under the same framework using the edge flow model. Figure 9 shows few more segmentation results on some typical images published in the literature.

7.1 Segmenting Natural Color Photographs

We have applied this segmentation algorithm to segment about 2,500 real natural images from Corel color photo CDs (volume 7, *Nature*). To the best of our knowledge, this is first time that a general-purpose segmentation algorithm has been demonstrated on such a large and diverse collection of real natural images. The usefulness of the proposed scheme lies in the fact that very little parameter tuning or selection is needed. The three parameters controlling segmentation are

1. Image attributes to be used to detect boundaries: color (gray intensity), texture, or combination of color and texture.
2. The preferred scale to localize the desirable image boundaries.
3. The approximate number of regions (for the region merging algorithm).

Each photo CD contains 100 images, and the same parameters were used for the entire image set on a give CD. The experimental results indicate that the proposed algorithm resulted in visually acceptable performance on this diverse image collection. Figure 10 shows some of the image segmentation results.

The average computational time for segmenting a 128×192 color image on a SUN Sparc20 workstation is about 4-10 minutes, depending on the types of image attributes used for constructing the edge flow. The texture edge flow is computationally more expensive because it requires the Gabor filtering as a preprocessing to extract features and the prediction has to be performed in each of the feature planes.

An image retrieval system which utilizes the proposed edge flow based segmentation algorithm for automatically analyzing images has been demonstrated in [14]. A web demonstration is available at *http://vivaldi.ece.ucsb.edu/Netra.*

7.2 Segmenting Large Aerial Photographs

Here we consider the use of edge flow model for segmenting large aerial photographs. The technique which we developed here has been used to process the images for geographical information retrieval in the UCSB Alexandria Digital Library project [13, 22].

Because the typical size of an airphoto is large (usually contain more than $5K \times 5K$ pixels), an accurate pixel-level segmentation could be computationally expensive. For most of the pattern classification and image retrieval applications, however, such a precise segmentation is often not necessary. To this end, we extend the edge flow model to perform a coarse image segmentation based on the texture features extracted from the equally partitioned image blocks. The prediction is performed between the neighboring blocks and a flow vector which points to the closest boundary at each block location is constructed.

In the experiment, each aerial photograph is first partitioned into 64×64 blocks of pixels. From each block, a texture feature vector is computed. Now consider a set of texture features $\left\{ \vec{f}(h, w)|_{1 \le h \le N_h,\, 1 \le w \le N_w} \right\}$ which are extracted from an airphoto with $N_h \times N_w$ blocks, where N_h and N_w represent the numbers of partitions in height and width, respectively. Given the texture feature at image block (h, w), we can predict its surrounding eight neighbors to have the same texture feature if they belong to the same homogeneous region. Thus, the prediction errors can be computed as

$$\begin{aligned}
Error(s, 0) &= \left\| \vec{f}(h, w+1) - \vec{f}(h, w) \right\| \\
Error(s, \pi/4) &= \left\| \vec{f}(h-1, w+1) - \vec{f}(h, w) \right\| \\
Error(s, \pi/2) &= \left\| \vec{f}(h-1, w) - \vec{f}(h, w) \right\| \\
Error(s, 3\pi/4) &= \left\| \vec{f}(h-1, w-1) - \vec{f}(h, w) \right\| \\
Error(s, \pi) &= \left\| \vec{f}(h, w-1) - \vec{f}(h, w) \right\| \\
Error(s, 5\pi/4) &= \left\| \vec{f}(h+1, w-1) - \vec{f}(h, w) \right\| \\
Error(s, 3\pi/2) &= \left\| \vec{f}(h+1, w) - \vec{f}(h, w) \right\| \\
Error(s, 7\pi/4) &= \left\| \vec{f}(h+1, w+1) - \vec{f}(h, w) \right\|
\end{aligned} \tag{26}$$

where $s = (h, w)$. Figure 11 shows the spatial relationship between the neighboring feature vectors. For simplicity, the texture edge energy $E(s, \theta)$ is set to be the same as $Error(s, \theta)$. Note that there are two edge energies associated with each orientation now in contrast with the previous cases. The probabilities of

edge flow direction (forward and backward) along each of the four orientations (0, $\pi/4$, $\pi/2$, and $3\pi/4$) are assigned based on (9). By using the strategy described in Section 4.2, the different directions of edge flows can be combined together to identify the best direction for searching for the closest boundary.

Following the edge flow computation, the resulting local edge flow is propagated to its neighbors if they have the same directional preference. The flow continues till it encounters an opposite flow. After the propagation reaches a stable state, the final edge flow energy is used for boundary detection. The detected boundary are then connected to form an initial set of image regions. At the end, a conservative region merging algorithm is used to group similar neighboring regions. Figure 12 shows one of such image segmentation result. This segmentation scheme help to represent and organize the image information in a more efficient way. Description of an image retrieval system for searching similar regions from an airphoto collection can be found in [13].

8 Discussions

In this paper we have presented a novel framework for detecting image boundaries and demonstrated its use in segmenting a large variety of natural images. In contrast to the traditional approaches, the edge flow model utilizes a predictive coding scheme to detect the direction of change in various image attributes and construct an edge flow field. By iteratively propagating the edge flow, the boundaries can be detected at image locations which encounter two opposite directions of flow in the stable state. The only significant control parameter (not including the region merging post-processing) is the image scale, which can be adjusted to the user's requirements.

For simplicity, a single scale parameter has been used for the entire image segmentation in our current implementation. However, this might not be appropriate for some images which contain multiple scale information. There is a need to locally adjust the scale parameter depending on the local texture/color properties such that meaningful boundaries at each image location can be detected. This local scale control remains as a future research problem.

As indicated in the experiments, the use of texture information increases the processing time significantly in performing image segmentation. One can use local image statistics to determine if the image is textured or not [11], and thus determine if texture segmentation is required.

A note regarding performance evaluation: Since no ground truth is available for the color images from the stock photo galleries, no quantitative performance evaluation can be provided at this time. However, our experiments with some of the synthetic texture mosaics have given results better than most of the algorithms that we are currently aware of in the segmentation literature. A visual inspection of the results indicate that the segmentation is of acceptable quality and well suited for applications such as image browsing wherein the automatic segmentation is critical. The segmentation results of 2,500 natural color images from a Corel photo gallery can be found at *http://vivaldi.ece.ucsb.edu/Netra*.

Appendix A:

Gabor Functions and Wavelets

Gabor functions are Gaussians modulated by complex sinusoids. In two dimensions they take the form:

$$g(x, y) = \left(\frac{1}{2\pi\sigma_x\sigma_y}\right)\exp\left[-\frac{1}{2}\left(\frac{x^2}{\sigma_x^2}+\frac{y^2}{\sigma_y^2}\right)\right]\cdot\exp[2\pi jWx] \tag{27}$$

The 2-D Fourier transform of $G(x, y)$ is

$$H(u, v) = \exp\left\{-\frac{1}{2}\left[\frac{(u-W)^2}{\sigma_u^2}+\frac{v^2}{\sigma_v^2}\right]\right\}, \tag{28}$$

where $\sigma_u = 1/2\pi\sigma_x$ and $\sigma_v = 1/2\pi\sigma_y$. Let $g(x, y)$ be the mother Gabor wavelet, then this self-similar filter dictionary can be obtained by appropriate dilations and rotations of the mother wavelet through the generating function:

$$g_{mn}(x, y) = a^{-m}G(x', y'), \qquad a > 1, \quad m, n = \text{integer} \tag{29}$$

$$x' = a^{-m}(x\cos\theta + y\sin\theta)\,,$$

$$y' = a^{-m}(-x\sin\theta + y\cos\theta)\,,$$

where $\theta = n\pi/K$ and K is the total number of orientations.

Gabor Filter Dictionary Design

The non-orthogonality of the Gabor wavelets implies that there is redundant information in the filtered images, and the following strategy is used to reduce this redundancy. Let U_l and U_h denote the lower and upper center frequencies of interest. Let K be the number of orientations and S be the number of scales in the multi-resolution decomposition. Then the design strategy is to ensure that the half-peak magnitude cross-sections of the filter responses in the frequency spectrum touch each other as shown in Figure 4. This results in the following formulas for computing the filter parameters σ_u and σ_v (and thus σ_x and σ_y).

$$a = (U_h/U_l)^{\frac{1}{S-1}}, \tag{30}$$

$$\sigma_u = \frac{(a-1)U_h}{(a+1)\sqrt{2\ln 2}} \tag{31}$$

$$\sigma_v = \tan\left(\frac{\pi}{2k}\right)\left[U_h - 2\ln 2\left(\frac{\sigma_u^2}{U_h}\right)\right]\left[2\ln 2 - \frac{(2\ln 2)^2\sigma_u^2}{U_h^2}\right]^{-\frac{1}{2}} \tag{32}$$

where $W = U_h$, $\theta = \pi/K$ and $m = 0, 1, \ldots, S-1$.

References

[1] A. C. Bovic, M. Clark, W. S. Geisler, “Multichannel texture analysis using localized spatial filters,” IEEE Trans. Pattern Anal. and Machine Intell., Vol. 12, pp. 55-73, January 1990.

[2] J. M. H. Du Buf, “Gabor phase in texture discrimination,” Signal Processing, Vol. 21, pp. 221-240, 1990.

[3] J. M. H. Du Buf and P. Heitkämper, “Texture features based on Gabor phase,” Signal Processing, Vol. 23, pp. 225-244, 1991.

[4] J. Canny, “Computational approach to edge detection,” IEEE Trans. on Pattern Analysis and Machine Intelligence, Vol. 8, No. 6, pp. 679-698, Nov. 1986.

[5] J. G. Daugman and C. J. Downing, “Demodulation, predictive coding, and spatial vision,” Journal of the Optical Society of America A, Vol. 12, No. 4, pp. 641-660, April 1995.

[6] R. Deriche, “Optimal edge detection using recursive filtering,” Proc. IEEE ICCV, pp. 501-505, 1987.

[7] D. Dunn, W.E. Higgins, and J. Wakeley, “Texture segmentation using 2-D Gabor elementary functions,” IEEE Trans. Pattern Anal. and Machine Intell., Vol. 16, pp. 130-149, Feb. 1994.

[8] I. Fogel and D. Sagi, “Gabor filters as texture discriminator,” Biological Cybernetics, 61, pp 103-113, 1989.

[9] M. Gökmen and A. K. Jain, “λ_τ-space representation of images and generalized edge detector,” IEEE Int. Conf. on Computer Vision and Pattern Recognition, pp. 764-769, San Francisco, CA, June 18-20, 1996.

[10] A. K. Jain and F. Farroknia, “Unsupervised texture segmentation using Gabor filters,” Pattern Recognition, 24(12), 1167-1186, 1991.

[11] K. Karu, A. K. Jain, and R. M. Bolle, “Is there any texture in the image?” Pattern Recognition, Vol. 29, No. 9, pp. 1437-1446, 1996.

[12] J. Malik and P. Perona, “Preattentive texture discrimination with early vision mechanisms,” J. Opt. Soc. Am. A, Vol. 7, pp. 923-932, May 1990.

[13] W. Y. Ma and B. S. Manjunath, “A texture thesaurus for browsing large aerial photographs,” to appear in Journal of the American Society for Information Science, 1997.

[14] W. Y. Ma and B. S. Manjunath, “NeTra: a toolbox for navigating large image databases,” in IEEE Int. Conf. on Image Processing, 1997.

[15] B. S. Manjunath and R. Chellappa, “A Unified approach to boundary detection,” IEEE Trans. Neural Networks, Vol. 4, No. 1, pp. 96-108, January 1993.

[16] B. S. Manjunath and W. Y. Ma, “Texture features for browsing and retrieval of image data,” IEEE Trans. on Pattern Analysis and Machine Intelligence, Vol. 18, No. 8, pp. 837-842, Aug. 1996.

[17] J. Mao and A. K. Jain, “Texture classification and segmentation using multiresolution simultaneous autoregressive models,” Pattern Recognition, Vol. 25, No. 2, pp. 173-188, 1992.

[18] D. Marr and E. Hildreth, “Theory of edge detection,” in Proc. of Roy. Soc., (Sec B, 207), pp. 187-217, 1980.

[19] P. Perona and J. Malik, “Scale-space and edge detection using anisotropic diffusion,” IEEE Trans. on Pattern Analysis and Machine Intelligence, Vol. 12, No. 7, pp. 629-639, July 1990.

[20] Y. Rubner and Carlo Tomasi, “Coalescing texture descriptors,” Proc. of the ARPA Image Understanding Workshop, pp. 927-935, Feb. 1996.

[21] J. Shen and S. Castan, “An optimal linear operator for edge detection,” Proc. IEEE Int. Conf. on Computer Vision and Pattern Recognition, pp. 109-114, 1986.

[22] T. R. Smith, “A digital library for geograpgically referenced materials,” IEEE Computer, pp.54-60, May 1996.

[23] M. V. Srinivasan, S. B. Laughlin, and A. Dubs, “Predictive coding: a fresh view of inhibition in the retina,” Proc. R. Soc. London Ser. B 216, 427-459, 1982.

[24] V. Torre and T. Poggio, “On edge detection,” IEEE Trans. Pattern Anal. Machine Intell., Vol. 8, No. 4, pp. 147-163, 1986.

[25] A. P. Witkin, “Scale-space filtering,” Proc. 8th Int. Joint Conf. on AI.(Karlsruhe, West Germany), pp. 1019-1022, 1983.

[26] S. R. Yhann and T. Y. Young, “Boundary localization in texture segmentation,” IEEE Trans. Pattern Anal. and Machine Intell., Vol. 4, pp. 849-855, June 1995.

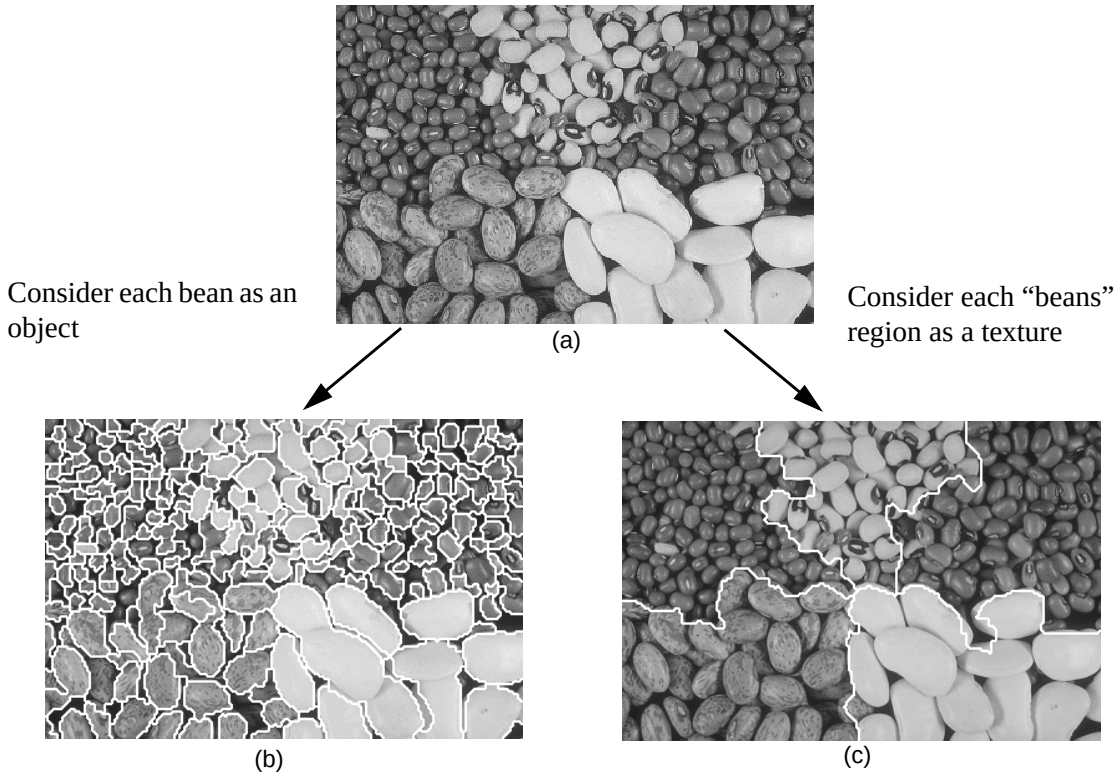


FIGURE 1. Image segmentation often require additional information from the user in order to select a proper scale to segment the objects or regions of interest, (a) shows an image with five different “beans” regions, (b) is the segmentation result using a smaller scale, and (c) is the segmentation result using a larger scale.

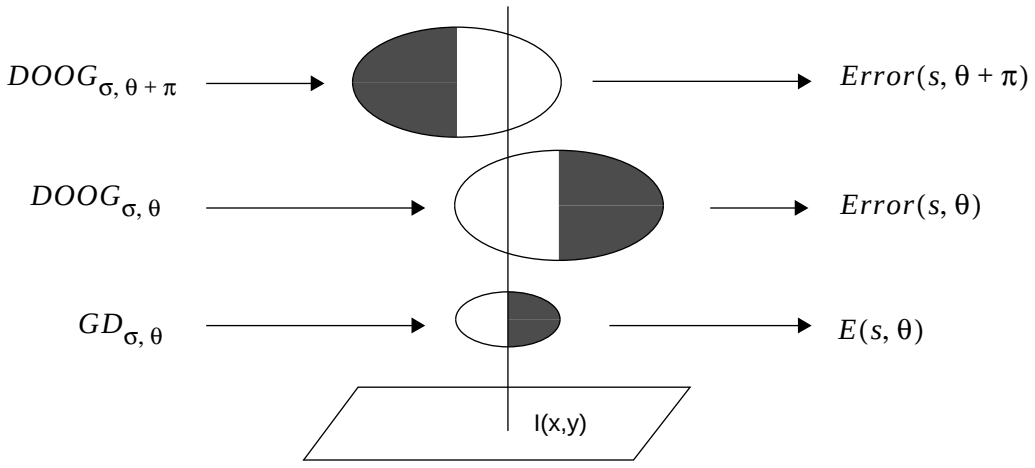


FIGURE 2. The computation of $E(s,\theta)$ and $P(s,\theta)$ using the GD and DOOG functions along a orientation θ. The shaded regions indicate the negative regions in the filter responses.

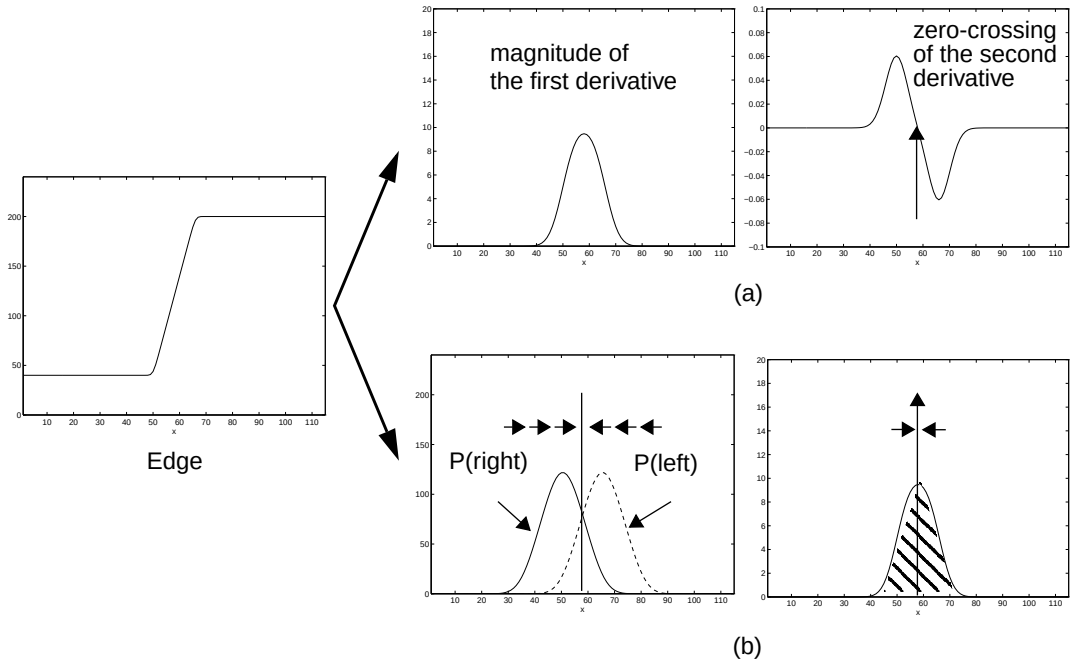


FIGURE 3. A comparison of the edge flow model with the conventional approach to detecting edges. (a) Traditional method of edge detection. (b) Edge flow model.

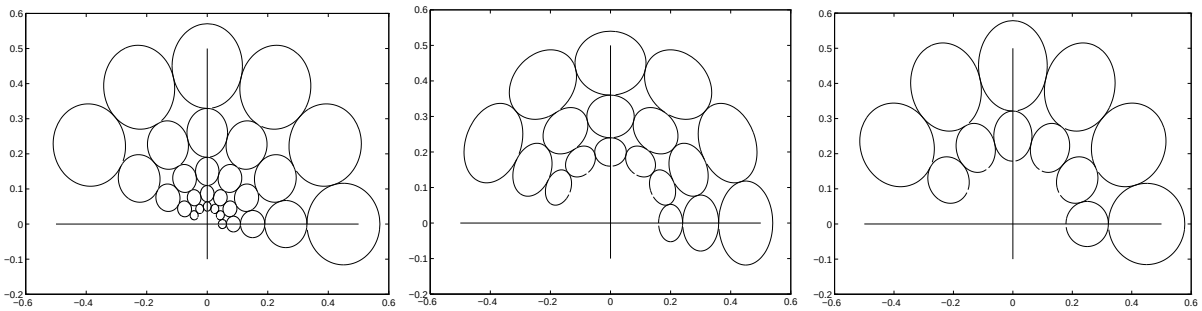


FIGURE 4. Fourier transforms of the Gabor filters, which are used to extract texture features for performing the image segmentation at different scales. (a) $\sigma = 5.0$ and $S = 5$, (b) $\sigma = 1.25$ and $S = 3$, and (c) $\sigma = 1.0$ and $S = 2$. The contour indicate the half-peak magnitude of the filter response.

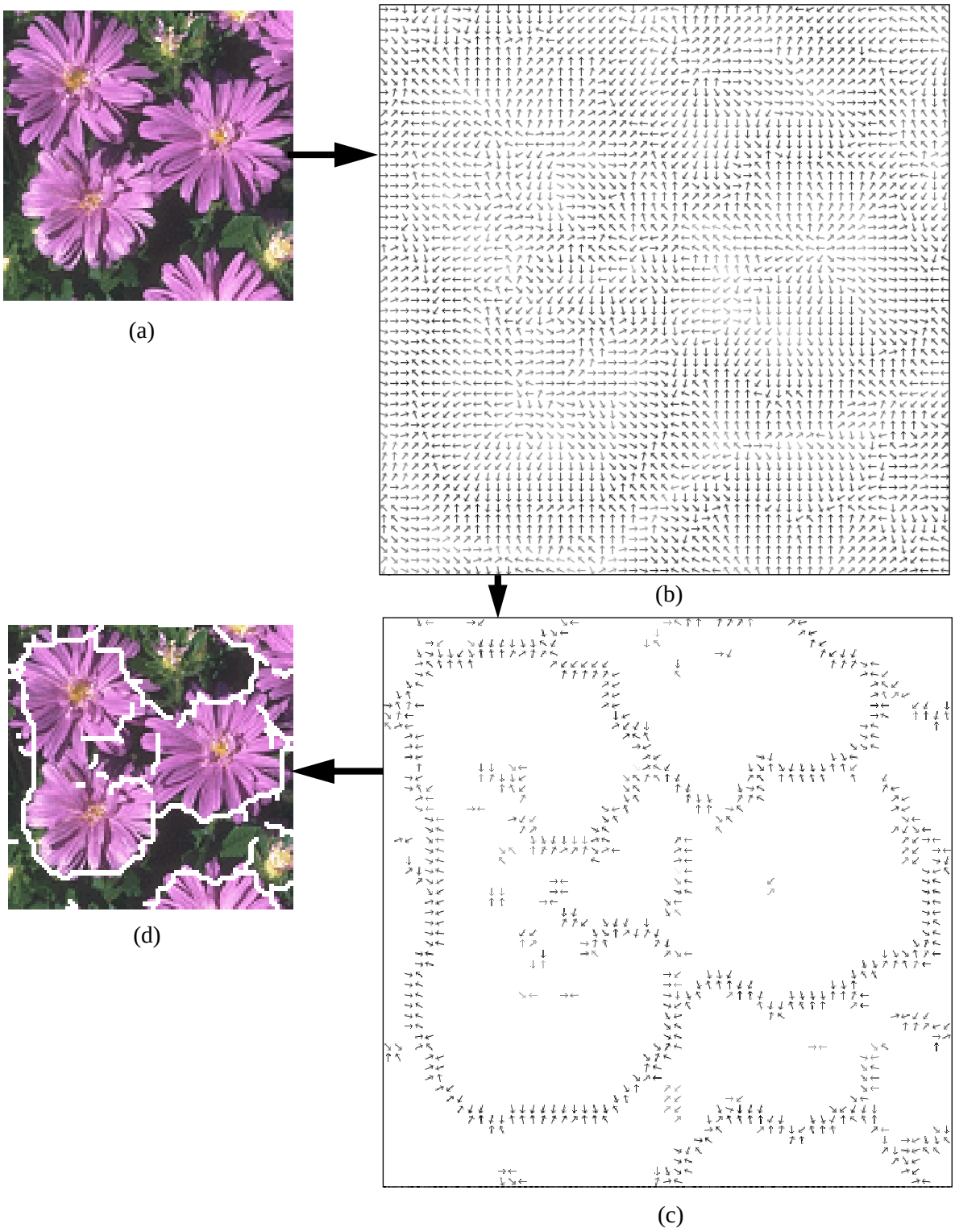


FIGURE 5. (a) A flower image. (b) The final edge flow field. (c) The result after edge flow propagation. (d) The result of boundary detection.

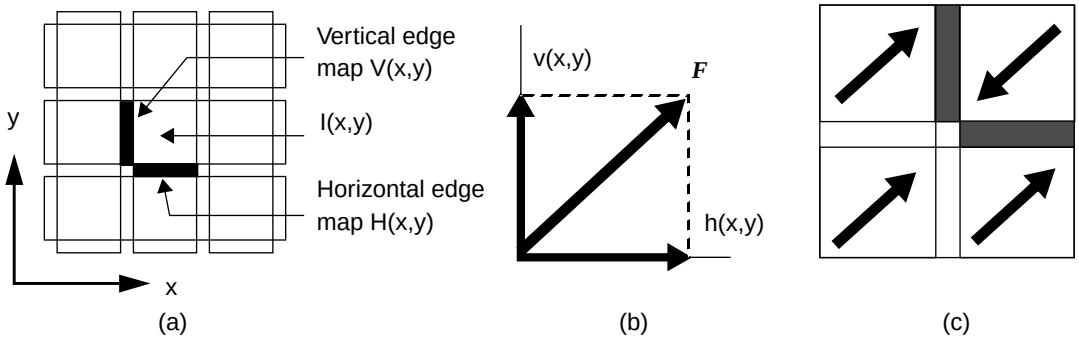


FIGURE 6. (a) Edge signals and image pixels, (b) The stable flow field vector ***F***, and (c) Boundary detection based on the edge flow.

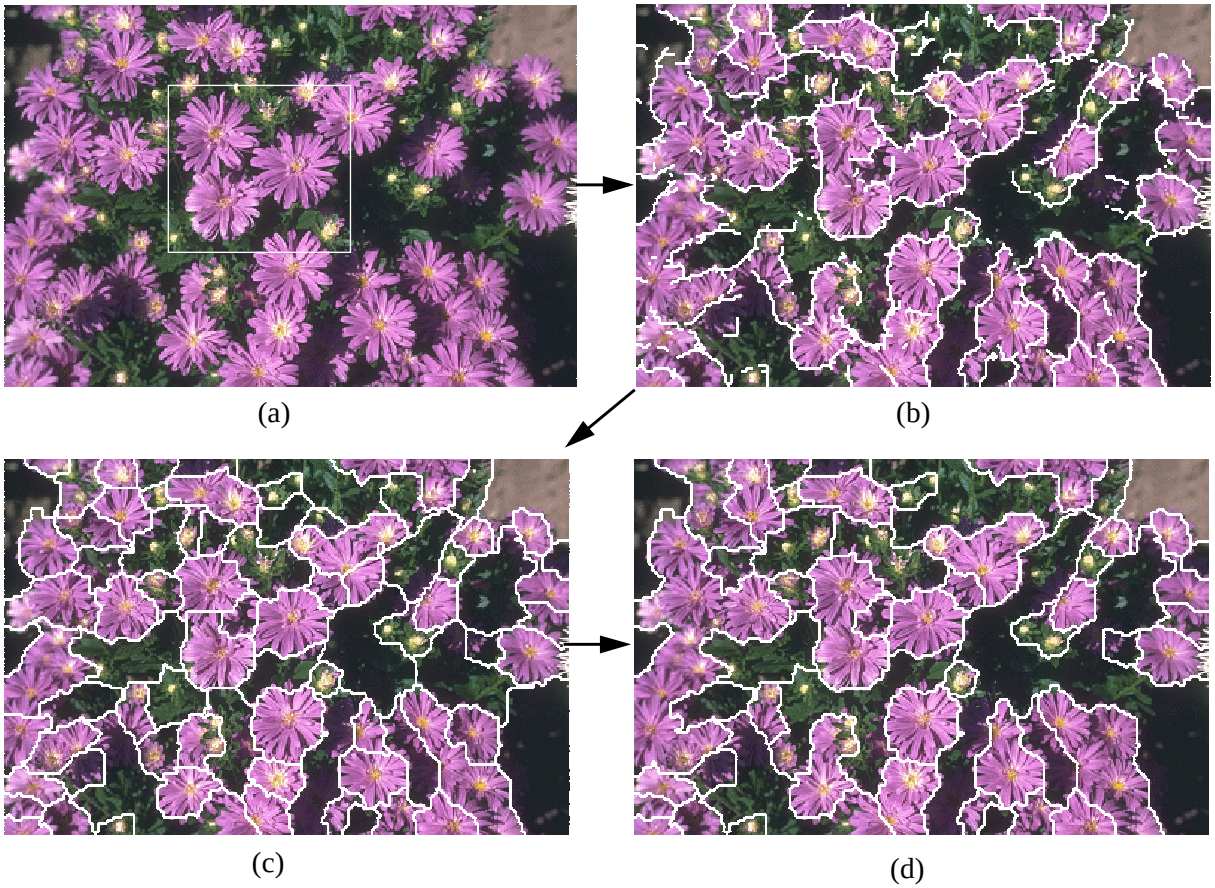


FIGURE 7. (a) A color flower image. Note that the detailed edge flow of image within the small window has been shown in Figure 5, (b) boundary detection using the edge flow model, (c) result after the boundary connection, and (d) result after the region merging.

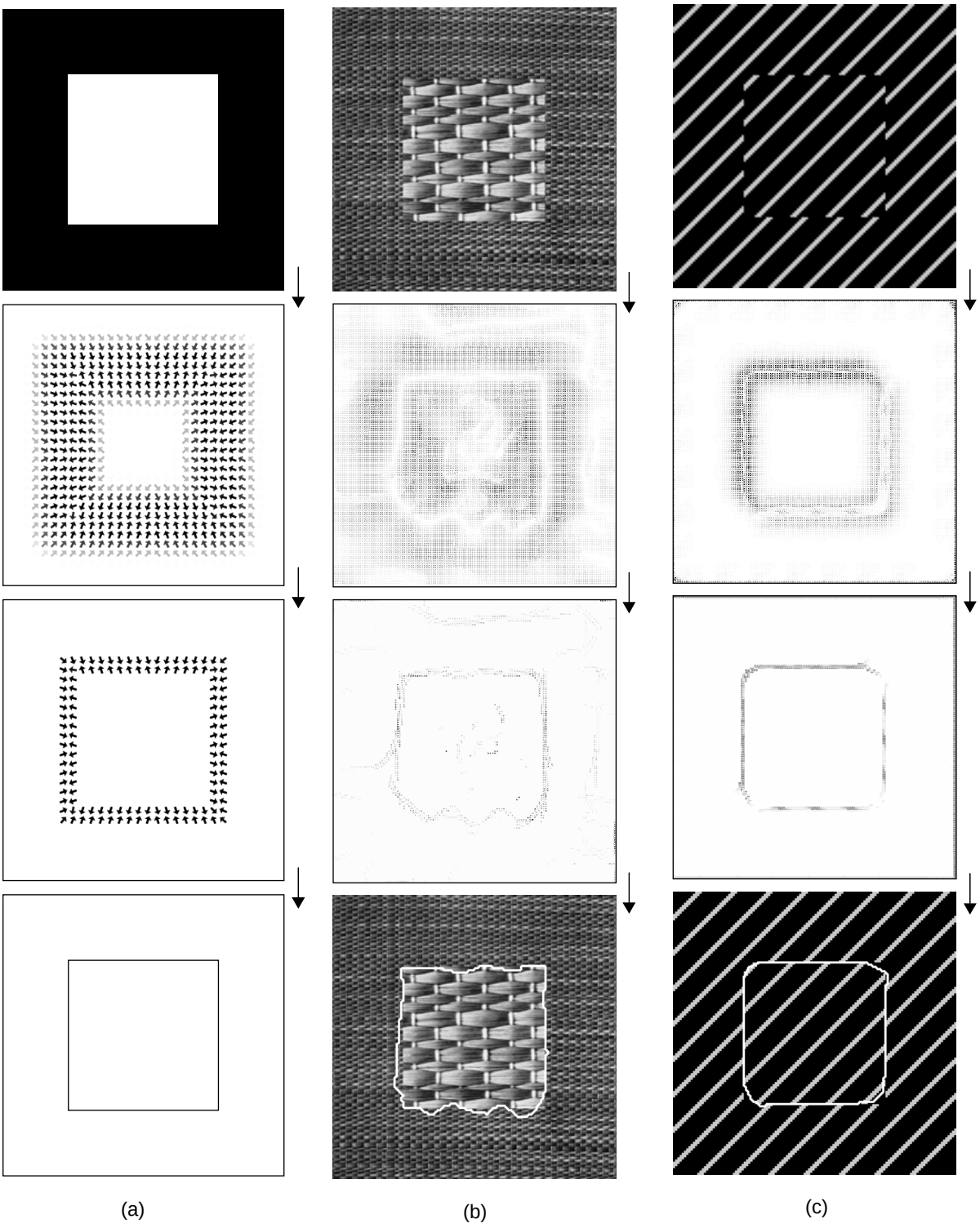
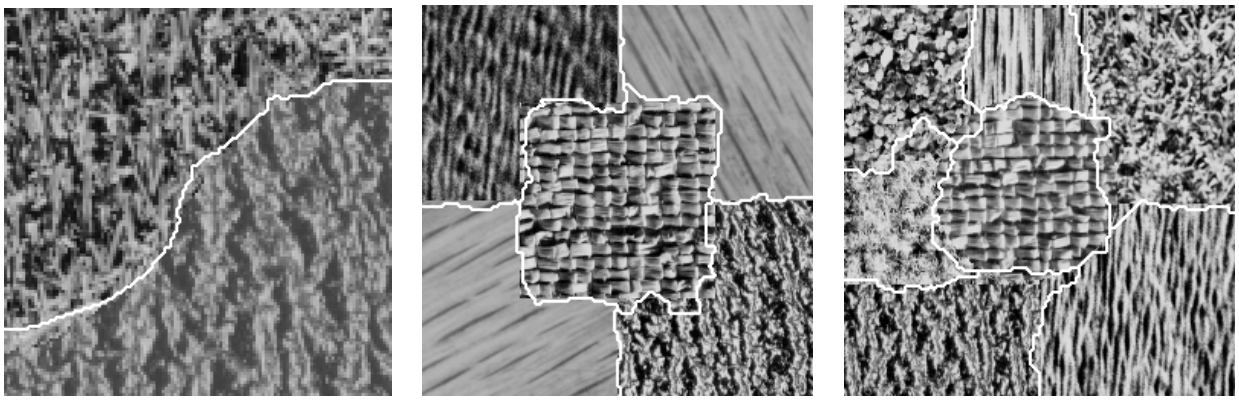
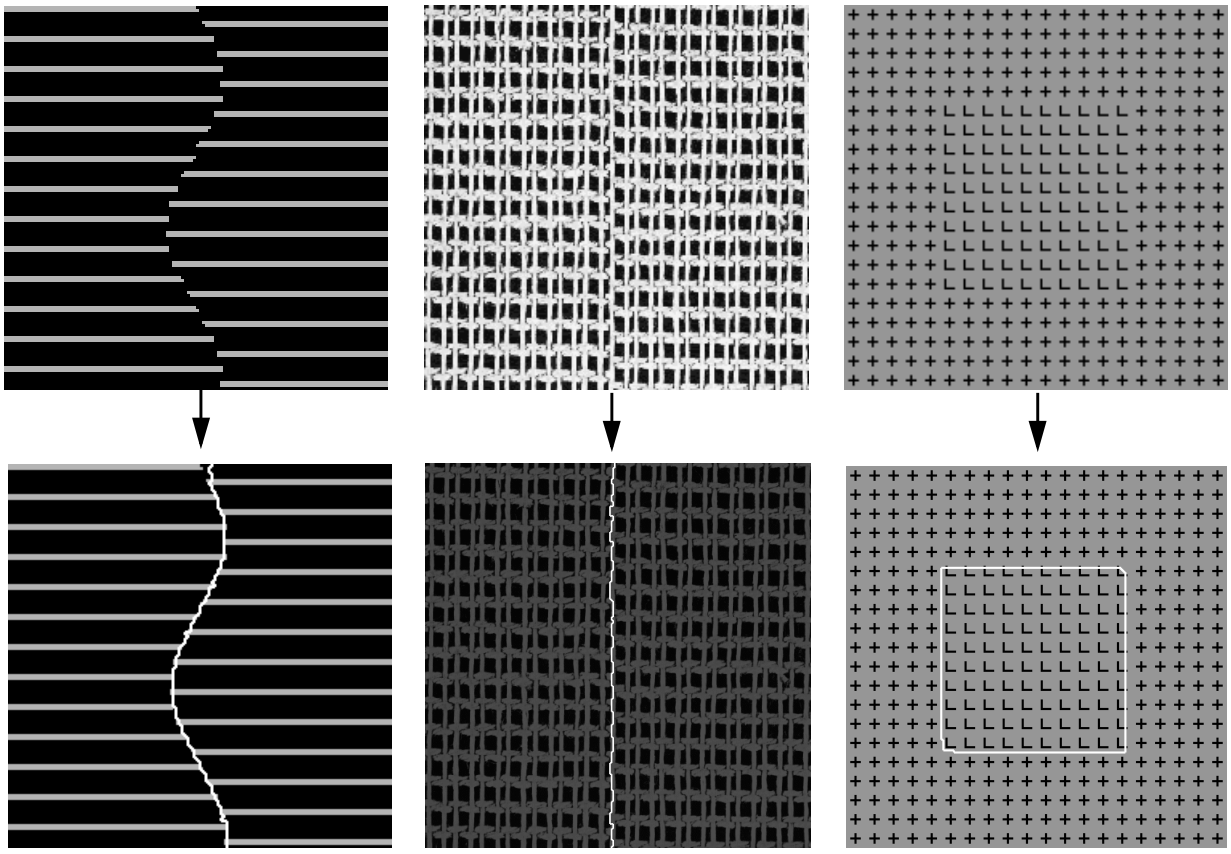


FIGURE 8. The use of edge flow model for detecting different type of image boundaries. From top to bottom are original image, edge flow computation, edge flow propagation, and boundary detection. (a) Intensity edges. (b) Texture boundaries. (c) Illusory boundaries.



(a)



(b)

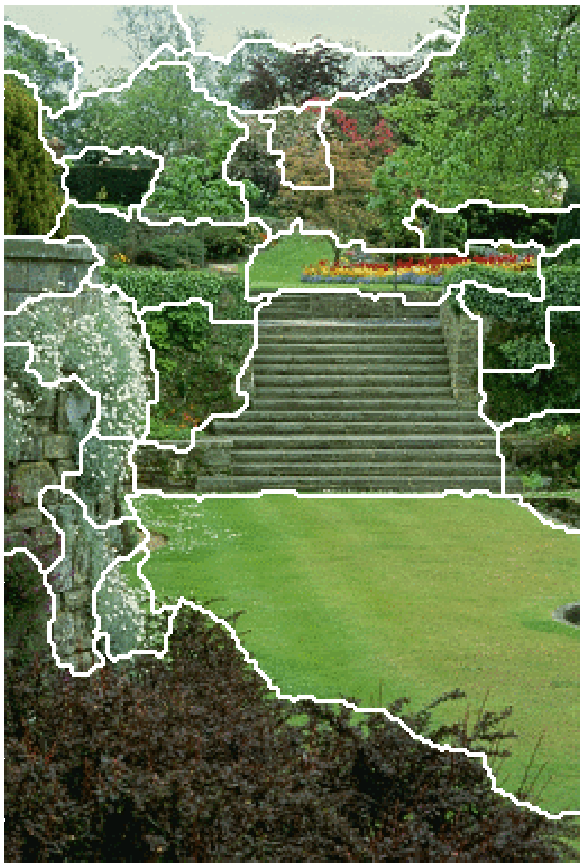
FIGURE 9. Image segmentation using edge flow model. (a) Using texture edge flow, (b) Using edge flow based on the Gabor phase.

(a)

(b)

(c)

(d)



(e)

FIGURE 10. Segmentation results of the real natural images from the Corel photo CDs. (a) scale parameter $\sigma = 3$ pixels, (b) $\sigma = 4$ pixels, (c) $\sigma = 3$ pixels, (d) $\sigma = 6$ pixels, and (e) $\sigma = 2$ pixels.

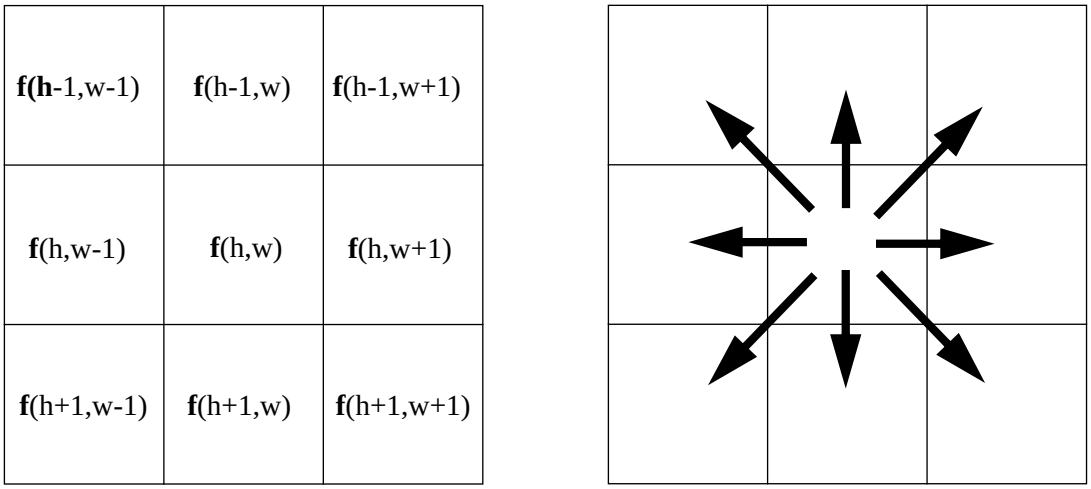
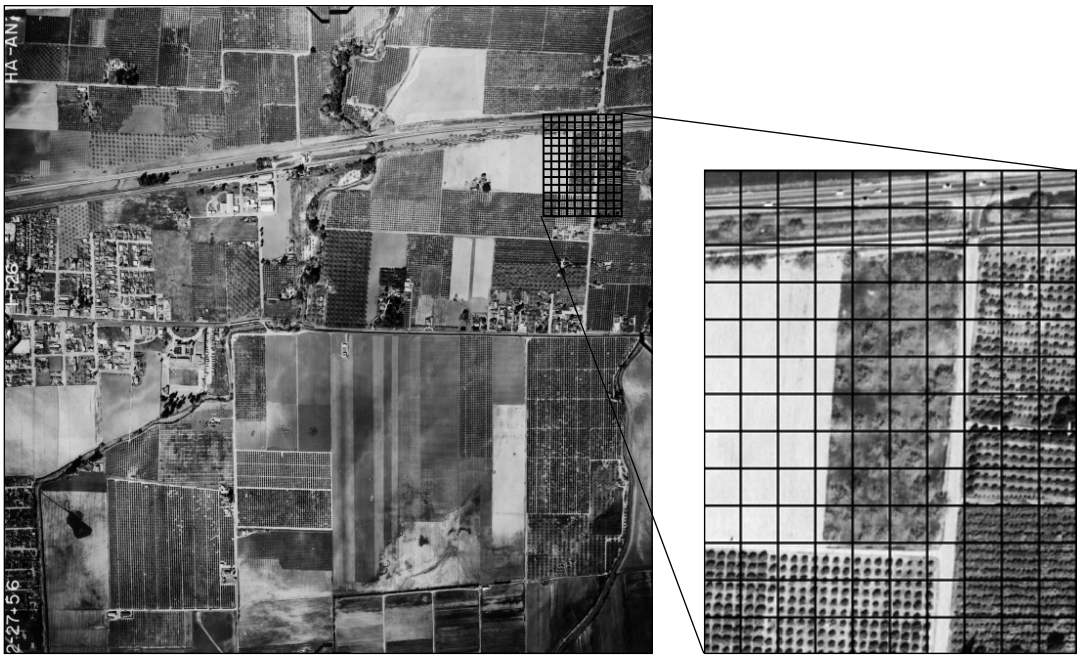
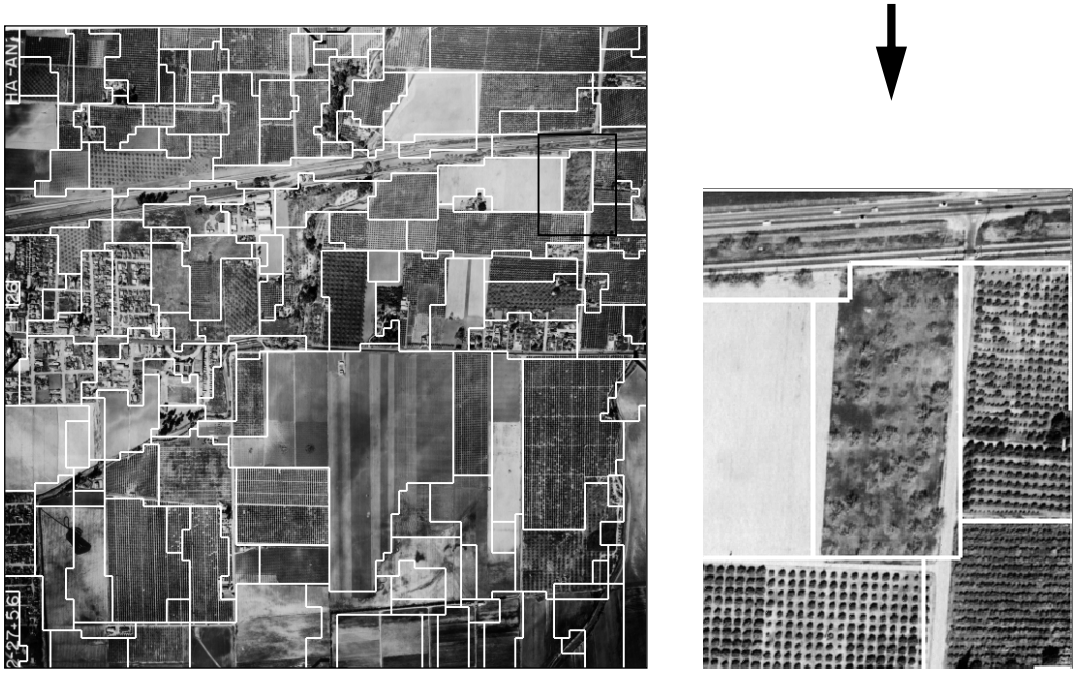


FIGURE 11. (a) The spatial relationship between neighboring feature vectors of image blocks, and (b) probabilities of edge flow toward different directions.



(a)



(b)

FIGURE 12. Segmenting large aerial photographs using the edge flow model.