Edge Flow: A Framework of Boundary Detection and Image Segmentation

W. Y. Ma and B. S. Manjunath
Department of Electrical and Computer Engineering
University of California, Santa Barbara, CA 93106-9560
WWW: http://vivaldi.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. However, despite considerable research and progress made in this area, the robustness and generality of the algorithms on large image datasets have not been established. Detecting discontinuities caused by color and texture continues to be a challenging problem.

Our objective is to develop a framework for detecting and integrating intensity/color and texture discontinuities as well as illusory boundaries in images. Towards this we propose an *edge flow* model for boundary detection and image segmentation. Salient features of this approach include (1) use of a predictive coding model for identifying and integrating the different types of image boundaries, (2) boundary detection based on a flow field propagation, and (3) very few "free" parameters that control the segmentation.

In the next section we briefly review the previous work on edge and boundary detection. The proposed edge flow model is explained in Section 3. The edge flow integration and some post-processing issues are discussed in Section 4, Section 5, and Section 6. Experimental results are demonstrated in Section 7. We conclude with some discussions in Section 8.

2 Previous Work

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, 13]. 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, 13, 16]. The regularization theory has also been frequently used in helping the design of edge detection algorithms [7, 15].

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 [6, 9, 10], or by unsupervised clustering [1, 5, 8], or by texture classification [12]. 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 window. 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.

3 "Edge Flow"

In this section, the general concept of edge flow is first outlined and a detailed implementation of it is illustrated. This approach facilitates the integration of different image attributes such as intensity/color, texture, and illusory discontinuities into a single framework 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:

$$F(s, \theta) = F[E(s, \theta), P(s, \theta), P(s, \theta + \pi)] \qquad (1)$$

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 edge flow "flows" in the direction θ, and $P(s, \theta + \pi)$ represents the probability of finding the image boundary if the edge flow "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 in the filtered outputs), 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.

3.1.1 Some Definitions

Consider the Gaussian function $G_\sigma(x, y)$ (σ denotes its standard deviation). The first derivative of Gaussian (GD) along the x-axis is given by

$$GD_\sigma(x, y) = -(x/\sigma^2)\, G_\sigma(x, y)\ , \qquad (2)$$

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

$$DOOG_\sigma(x, y) = G_\sigma(x, y) - G_\sigma(x + d, y) \qquad (3)$$

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 previous functions along different orientations θ:

$$GD_{\sigma,\theta}(x, y) = GD_\sigma(x', y')\ , \qquad (4)$$

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

$$x' = x\cos\theta + y\sin\theta\text{, and } y' = -x\sin\theta + y\cos\theta\ .$$

3.2 Intensity Edge Flow

3.2.1 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

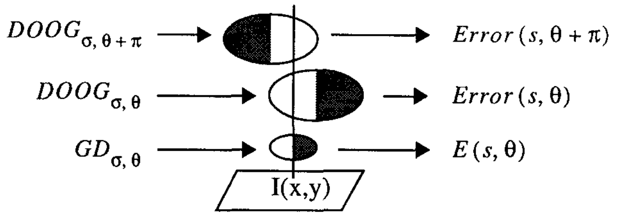


Figure 1: The computation of $E(s, \theta)$ and $P(s, \theta)$ using GD and DOOG along a orientation θ. The shaded regions indicate the negative regions in the filter responses.

scale σ is defined to be the magnitude of the gradient of the smoothed image $I_\sigma(x, y)$ along the orientation θ, which can be computed as

$$E(s, \theta) = |I(x, y) * GD_{\sigma,\theta}| \qquad (5)$$

where $s = (x, y)$. This edge energy indicates the strength of the intensity change.

3.2.2 Computing $P(s, \theta)$

For each of the edge energy along the orientation θ at location s, we 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 s to predict its neighbor in the direction θ. Ideally they should have similar intensities if they belong to the same object, and thus the prediction error can be computed as

$$Error(s, \theta) = |I_\sigma(x + d\cos\theta, y + d\sin\theta) - I_\sigma(x, y)| \qquad (6)$$
$$= |I(x, y) * DOOG_{\sigma,\theta}(x, y)|$$

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

$$P(s, \theta) = \frac{Error(s, \theta)}{Error(s, \theta) + Error(s, \theta + \pi)}\ . \qquad (7)$$

The computations of $E(s, \theta)$ and $P(s, \theta)$ using the GD and the DOOG are shown in Figure 1.

Figure 2 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 2(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

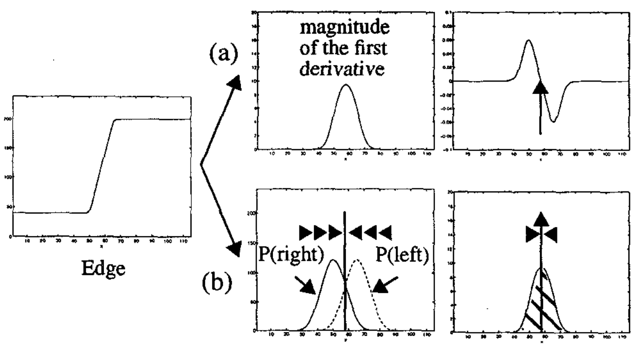


Figure 2: A comparison of the edge flow model with the conventional approach to detecting edges. (a) Traditional method of edge detection. (b) Edge flow model.

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 (5)-(7)). However, real images usually do not contain such ideal edges.

3.3 Texture Edge Flow

The previous section illustrated the basic concepts of edge flow computation using intensity edges. Much of the same formulation carries over to image attributes such as color and texture.

The texture features are extracted based on a Gabor decomposition. These features have been widely used in many texture analysis/segmentation applications [1, 6, 8, 11]. The basic idea is to decompose images into multiple oriented spatial frequency channels, and the channel envelopes (amplitude and phase) are used to form the feature maps. We use the strategy proposed in [11] for adaptively designing the Gabor filter bank.

Given the scale parameter σ, we 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. Based on the range of these two center frequency, an appropriate number of Gabor filters $g_i(x, y)$ is generated to cover the spectrum.

The complex Gabor filtered images are:

$$O_i(x, y) = I * g_i(x, y) = m_i(x, y) \exp[\phi_i(x, y)] \qquad (8)$$

where $1 \le i \le N$, N is the total number of filters, $m_i(x, y)$ is the magnitude, and $\phi_i(x, y)$ 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

$$\Psi(x, y) = [m_1(x, y), m_2(x, y), \ldots, m_N(x, y)] \,. \qquad (9)$$

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 one in Figure 5(c). In this case, the phase information $\{\phi(x, y)\}$ have 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 using the texture features Ψ. The texture edge energy, which measures the change in local texture information, is given by

$$E(s, \theta) = \sum_{1 \le i \le N} |m_i(x, y) * GD_{\sigma,\theta}(x, y)| \cdot w_i \qquad (10)$$

where $w_i = 1/\|\alpha_i\|$ and $\|\alpha_i\|$ 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 using the prediction error:

$$Error(s, \theta) = \sum_{1 \le i \le N} |m_i(x, y) * DOOG_{\sigma,\theta}(x, y)| \cdot w_i (11)$$

which is the weighted sum of prediction errors from each texture feature map. Thus, the probabilities $P(s, \theta)$ of the flow direction can be estimated using (7).

3.4 Edge Flow Based on Phase

From (8), the complex Gabor filtered image can be written as $O(x, y) = Re(x, y) + j \cdot Im(x, y)$, 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:

$$\phi(x, y) = \text{atan}[Im(x, y)/Re(x, y)] \,. \qquad (12)$$

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 [14].

The unwrapped phase can be decomposed into a global linear phase component and a local phase component. The local phase contains important information about the locations where the texture property changes. In other words, within a uniformly 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 formulate the edge flow field using the phase information, there are two problems that we have to overcome. 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 $\frac{d}{dx}\mathrm{atan}(x) = 1/(1+x^2)$. 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:

$$\frac{\partial}{\partial x}\phi(x, y) = \mathrm{imag}\left[O^*(x, y) \cdot \frac{\partial}{\partial x}O(x, y)\right] / m(x, y)^2 \quad (13)$$

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

$$\hat{\phi}(x+d, y) = \phi(x, y) + d \cdot \frac{\partial}{\partial x}\phi(x, y), \quad (14)$$

and therefore, the prediction error is equal to

$$Error = \phi(x+d, y) - \phi(x, y) - d \cdot \frac{\partial}{\partial x}\phi(x, y). \quad (15)$$

However, because the first two terms in equation (15) 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

$$Error(s, \theta) = \Big|\phi(x + d \cdot \cos\theta, y + d \cdot \sin\theta) - \phi(x, y) - d \cdot \frac{\partial}{\partial n}\phi(x, y) + 2\pi k(x, y)\Big| \quad (16)$$

where $n = (\cos\theta, \sin\theta)$ and $k(x, y)$ is an integer.

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 for general-purpose boundary detection:

$$\boldsymbol{E}(s, \theta) = \sum_{a \in A} E_a(s, \theta) \cdot w(a), \quad \sum_{a \in A} w(a) = 1 \quad (17)$$

$$\boldsymbol{P}(s, \theta) = \sum_{a \in A} P_a(s, \theta) \cdot w(a) \quad (18)$$

where $E_a(s, \theta)$ and $P_a(s, \theta)$ represent the energy and probability of the edge flow computed from image attribute a where $a \in$ {intensity/color, texture, and phase}. $w(a)$ is the weighting coefficient among various types of image attributes.

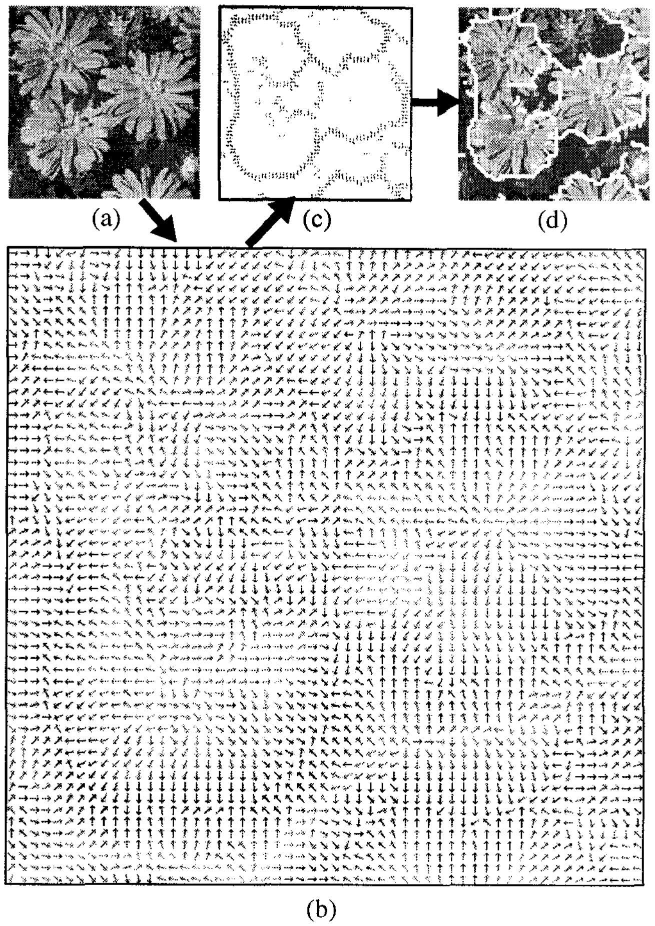


Figure 3: (a) A flower image. (b) The computed edge flow field. (c) The result after edge flow propagation. (d) The result of boundary detection.

4.2 Combining Different Directions of Edge Flows

In the example of Figure 2, the final direction of local edge flow 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 location, 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 $\{F(s, \theta)|_{0 \le \theta < \pi}\}$, we first identify a continuous range of flow directions which maximizes the sum of probabilities in that half plane:

$$\Theta(s) = \underset{\theta}{\mathrm{argmax}} \left\{ \sum_{\theta \le \theta' < \theta + \pi} P(s, \theta') \right\} \quad (19)$$

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

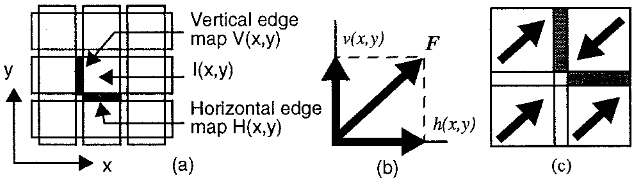


Figure 4: (a) Edge signals and image pixels, (b) The stable flow field vector F, and (c) Boundary detection based on the edge flow.

range, and is given by

$$\vec{F}(s) = \sum_{\Theta(s) \le \theta < \Theta(s) + \pi} E(s, \theta) \cdot \exp(j\theta) , \qquad (20)$$

where $\vec{F}(s)$ is a complex number with its magnitude representing the resulting edge energy and angle representing the flow direction. Figure 3(a)-(b) show an example of the edge flows on a flower image with scale $\sigma = 2$ pixels.

5 Flow Propagation and Boundary Detection

After the edge flow $\vec{F}(s)$ 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 $\vec{F}_0(s) = \vec{F}(s)$.
2. Set the initial edge flow $\vec{F}_{n+1}(s)$ at time $n+1$ to zero.
3. At each image location $s = (x, y)$, identify the neighbor $s' = (x', y')$ which is in the direction of edge flow $\vec{F}_n(s)$, i.e., $\angle\vec{F}_n(s) = \text{atan}((y'-y)/(x'-x))$.
4. Propagate the edge flow if $\vec{F}_n(s') \cdot \vec{F}_n(s) > 0$
   $\vec{F}_{n+1}(s') = \vec{F}_{n+1}(s') + \vec{F}_n(s)$;
   otherwise, $\vec{F}_{n+1}(s) = \vec{F}_{n+1}(s) + \vec{F}_n(s)$.
5. If nothing has been changed, stop the iteration. Otherwise, set $n = n+1$ and go to step 2.

Once the edge flow propagation reaches a stable state, we can detect the image boundaries by identifying the locations which have non-zero edge flow coming from two opposing directions. The vertical and horizontal edge maps between image pixels (see Figure 4(a)) are used to represent boundary. Let $\vec{F} = (h(x, y), v(x, y))$ be the final stable edge flow (see Figure 4(b)). Then, the edge signals will be turned on if and only if the two neighboring edge flows point at each other. Its energy is defined to be the summation of the projections of those two edge flow toward it. Figure 4(c) shows the detected edges. Figure 3(c)-(d) shows an example of the edge flow propagation and boundary detection using a flower image.

6 Boundary Connection and Region Merging

After boundary detection, disjoint boundaries are connected to form closed contours. The basic strategy is to associate a neighborhood search size proportional to the length of the contour. At the unconnected ends, a search is conducted for the nearest boundary element which is within the specified search neighborhood. If such a boundary element is found, a smooth boundary segment is generated to connect the open contour to another boundary element. This process is repeated few times (typically 2-3 times in our experiments) till all open contours are closed. Finally, a region merging algorithm (based on a normalized distance in the feature space: color histograms and texture features) is used to reduce the number of regions to within a certain user specified value.

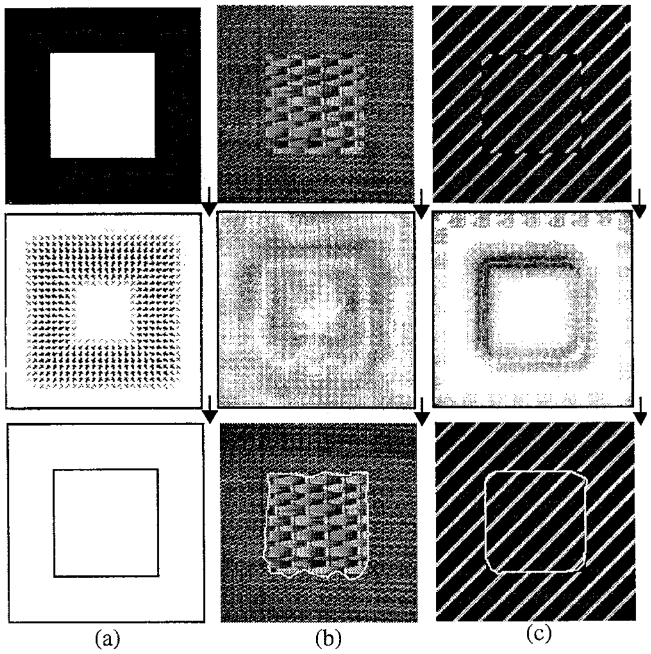
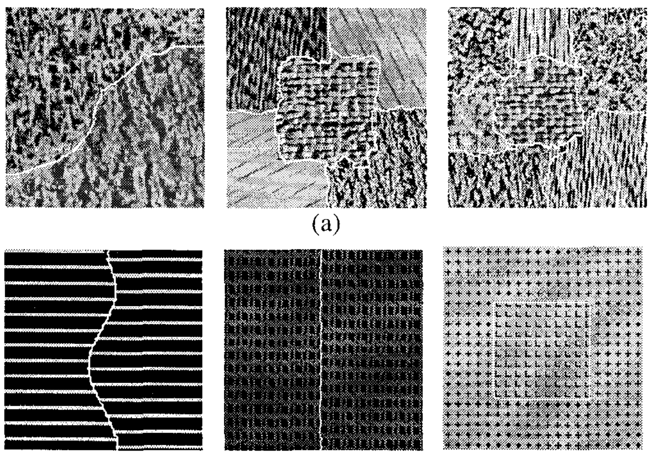


Figure 5: The use of edge flow model for detecting different type of image boundaries. From top to bottom are original image, edge flow computation, and boundary detection. (a) Intensity edges. (b) Texture boundaries. (c) Illusory boundaries.

7 Experimental Results

Figure 5 shows the use of edge flow model in detecting the various types of image boundaries. The segmentation results of some typical texture images are shown in Figure 6.

In addition, we have applied this algorithm to segment about 2,500 real natural images from Corel color photo CDs. The usefulness of the proposed scheme for segmentation lies in the fact that very little parameter tuning or selection is needed. The two parameters controlling segmentation are the preferred image scale and the approximate number of regions (for the region merging algorithm). The experimental results indicate that the proposed approach results in visually acceptable segmentation on this diverse image collection. Unfortunately, we can not provide any quantitative performance measures at this time due to the lack of ground


(a)

(b)

Figure 6: Image segmentation using edge flow model. (a) Using texture edge flow, (b) Using edge flow based on the Gabor phase.

truth. Figure 7 shows some of the examples. More examples of color image segmentation are available on the web (http:/ /vivaldi.ece.ucsb.edu).

8 Discussions and Conclusions

We have presented an edge flow model for detecting various types of image boundaries within a single framework, and demonstrated the use of this model in segmenting a large variety of natural images. In contrast with the traditional approaches, the edge flow model utilizes a predictive coding scheme to detect the direction of change in various image attributes, and perform the boundary detection by propagating this flow field. The only control parameter (not including the region merging post-processing) is the image scale, which can be adjusted to the user's requirements.

We are currently working on using an adaptive local scale instead of a single scale parameter for the entire image. Our preliminary experiments indicate that the results can be improved further by selectively choosing this scale depending on the local texture/color properties. Automatic selection of this parameter is an important research issue. Application to content based image retrieval which combines segmentation and feature extraction are being investigated.

Acknowledgments: This research was supported in part by a grant from NSF IRI-94-11330.

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, Jan. 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 Anal. and Machine Intell., Vol. 8, No. 6, pp. 679-698, Nov. 1986.

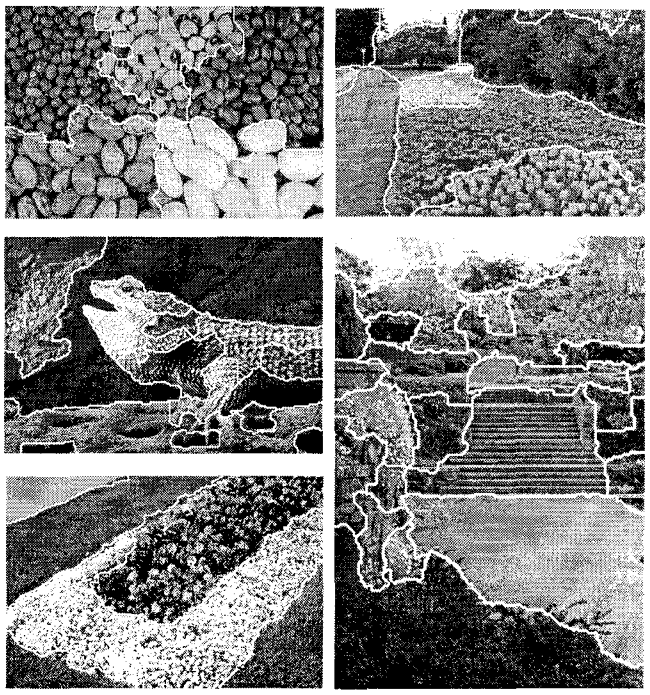


Figure 7: Segmentation results of the real natural images from a Corel photo CD.

[5] 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.
[6] I. Fogel and D. Sagi, "Gabor filters as texture discriminator," Biological Cybernetics, 61, pp 103-113, 1989.
[7] M. Gökmen and A. K. Jain, "λ_τ-space representation of images and generalized edge detector," IEEE CVPR'96, pp. 764-769, San Francisco, CA, Junc 18-20, 1996.
[8] A. K. Jain and F. Farroknia, "Unsupervised texture segmentation using Gabor filters," Pattern Recogn., 24(12), 1167-86, 1991.
[9] J. Malik and P. Perona, "Preattentive texture discrimination with early vision mechanisms," J. Opt. Soc. Am. A, Vol. 7, pp. 923-932, May 1990.
[10] B. S. Manjunath and R. Chellappa, "A Unified approach to boundary detection," IEEE Trans. Neural Networks, Vol. 4, No. 1, pp. 96-108, Jan. 1993.
[11] B. S. Manjunath and W. Y. Ma, "Texture features for browsing and retrieval of image data," IEEE Trans. on Pattern Anal. and Machine Intell., Vol. 18, No. 8, pp. 837-842, Aug. 1996.
[12] 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.
[13] D. Marr and E. Hildreth, "Theory of edge detection," in Proc. of Roy. Soc., (Sec B, 207), pp. 187-217, 1980.
[14] M. S. Scivire and M. A. Fiddy, "Phase ambiguities and the zeros of multidimensional band-limited functions," J. Opt. Soc. Amer., Vol. A2, pp. 693-697, 1985.
[15] V. Torre and T. Poggio, "On edge detection," IEEE Trans. Pattern Anal. Machine Intell., Vol. 8, No. 4, pp. 147-163, 1986.
[16] A. P. Witkin, "Scale-space filtering," Proc. 8th Int. Joint Conf. on AI.(Karlsruhe, West Germany), pp. 1019-1022, 1983.