Two-Level Image Segmentation Based on Region and Edge Integration

Qing Wu and Yizhou Yu

Department of Computer Science
University of Illinois at Urbana-Champaign
Urbana, IL 61801, USA
{qingwu1,yyz}@uiuc.edu

Abstract. This paper introduces a two-level approach for image segmentation based on region and edge integration. Edges are first detected in the original image using a combination of operators for intensity gradient and texture discontinuities. To preserve the spatial coherence of the edges and their surrounding image regions, the detected edges are vectorized into connected line segments which serve as the basis for a constrained Delaunay triangulation. Segmentation is first performed on the triangulation using graph cuts. Our method favors segmentations that pass through more vectorized line segments. Finally, the obtained segmentation on the triangulation is projected onto the original image and region boundaries are refined to achieve pixel accuracy. Experimental results show that the two-level approach can achieve accurate edge localization, better spatial coherence and improved efficiency.

1 Introduction

Two basic image segmentation approaches are region-based and edge-based segmentation. Region-based segmentation offers closed contours automatically while edge-based methods need an extra propagation step to obtain complete region contours. Region-based methods can also be categorized into local region growing and top-down global region partitioning, such as graph-theoretic optimal grouping algorithms [1, 2]. Global region partitioning makes use of global optimality criteria and can produce more semantically meaningful results, but typically suffers from poor localization of the region boundaries. This is because the criteria used are based on the statistics obtained from all the pixels in an entire image region. They do not reflect the local characteristics available at a specific pixel. On the other hand, local region growing or edge detection offer accurate boundary localization, but usually do not have sufficient global knowledge to perform the task well. Researchers have also been trying to integrate region and edge information in image segmentation. A complete survey on such integrations is given in [3]. In this paper, we are interested in integration between global graph-theoretic grouping algorithms and edge detection. There has been limited work along this direction except the embedded integration presented in [4, 5].

The goal of this paper is to perform image segmentation with integrated region and edge information so that both global image statistics and local edge responses can be utilized. The result we get from this would still be segmented regions but with accurate boundary localization at places where edge detection operators produce reasonably strong responses. The rest of the region boundaries can also be viewed as a viable solution to contour completion based on region information.

How can we integrate region and edge information in image segmentation ? A detected edge does not necessarily belong to a region boundary since an object can have internal edges besides the edges on its contour. The decision whether a specific edge should belong to a region boundary should be made within a global context, such as the relative positions of all the edges and the image statistics of the regions surrounding the edges, rather than a local one. A detected edge segment is usually made up of multiple connected edge pixels. These multiple edge pixels should be considered as a whole instead of a set of uncorrelated edge fragments because of their spatial proximity (Fig. 1). For the same reason, the pixels immediately adjacent to the edge should be considered as connected local regions instead of uncorrelated pixels. If we imagine a local image region with an edge passing through its center, the edge splits the region into two halves. Depending on whether this edge belongs to a region boundary or not, these two halves may or may not belong to the same region in the final segmentation. Nevertheless, pixels in the same half should belong to the same region. Note that part of the boundaries of the two halves except for the central edge are indefinite and need to be determined during segmentation. This argument leads to the suggestion that region segmentation should be performed at a coarser granularity than pixels in order to incorporate edges.

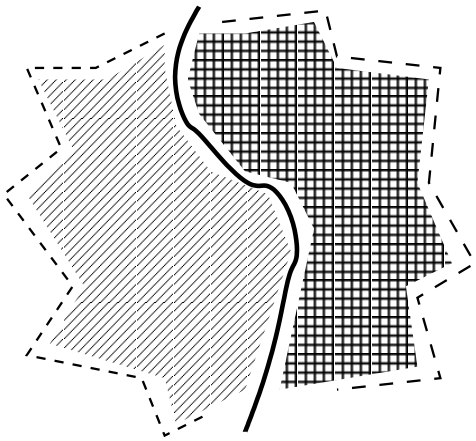

Fig. 1. A connected edge segment divides its neighborhood into two halves. During segmentation, the multiple pixels on the edge segment as well as the pixels in each of the half neighborhoods should be considered as a whole instead of being uncorrelated.

The main contribution of this paper is to propose a two-level approach for image segmentation with integrated region and edge information. At the lower level is the original set of pixels upon which edge detection can be performed. Connected edge pixels are grouped into edge segments. At the upper level is a triangulation which is an abstraction of the original image. Each image edge segment corresponds to an edge in the triangulation. Segmentation is performed on the triangulation. Once the segmentation at the upper level is done, those triangle edges without corresponding edge segments at the lower level will be refined at the pixel level. Segmentation at a granularity coarser than pixels in a rich texture region is also helpful because texture descriptors typically need to collect information in a local region. One additional advantage of having a two-level scheme is significantly improved efficiency.

2 Two-Level Segmentation

2.1 Edge Detection

Our method needs to detect edges first in a preprocessing step although edge detection is not the focus of this paper. Since we only consider gray-scale images, we apply previous approaches for detecting both intensity and texture edges. Intensity edges are detected using the first and second derivatives of Gaussian. A tensor product between a Gaussian and the first derivative of Gaussian detects step edges. It is very similar to the numerical solutions of Canny detectors [6]. A tensor product between a Gaussian and the second derivative of Gaussian is used for detecting ridge edges. These filters based on Gaussian derivatives can produce accurate locations for intensity edges. But they may produce extraneous edges in texture regions. The initial set of texture edges are obtained from the EdgeFlow algorithm [7]. While this algorithm can detect most texture edges, it also produces extraneous edges.

To remove extraneous edges, we adopt a likelihood of texture edges as follows: define a circular image region centered at the currently considered pixel; divide the circle into two halves using one of the potential edge orientations; apply multiscale odd-symmetric and even-symmetric filters to the two halves[5]; compute both intensity histograms and histograms of the filter responses for each of the two half regions [8]; use χ^2 test to calculate the difference between two corresponding histograms, where the χ^2 distance between two histograms is defined as

$$\chi^2(h_L, h_R) = \frac{1}{2} \sum_k \frac{[h_L(k) - h_R(k)]^2}{h_L(k) + h_R(k)};$$

obtain the weighted average of the histogram differences. The likelihood of a texture edge at the pixel is defined as the maximum weighted average among all potential orientations. Six different edge orientations are used in practice. We eliminate those extraneous edges which lie in an image region with low likelihood of texture edges.

Finally, we combine intensity and texture edges. Actually the EdgeFlow algorithm also produces intensity edges which are not as accurate as those obtained from Gaussian derivatives. Therefore, if there is an EdgeFlow edge in the vicinity of an edge obtained from Gaussian derivatives, the EdgeFlow edge is eliminated.

Our segmentation algorithm can incorporate any type of edges including the high quality edges recently obtained from a learning method [9].

2.2 Upper Level Construction

We first vectorize the detected edges from Section 2.1. This process converts every connected edge into a set of connected line segments by tracing the pixels on the edge. All the vertices of the line segments lie on the original edge. To create a new line segment from a vertex v_1, we move along the edge pixel by pixel until the distance between the current pixel p and v_1 reaches a prescribed threshold or we have reached the end of the edge. Note that we do not require that the line segments fit the original edge very well since we still keep the position of the original edge in the lower level and the line segments are only an abstraction in the upper level. The line segments are called *hard edges* since each of them is associated with a corresponding edge segment in the lower level.

The hard edges may be quite sparse in the image plane. Although we need a coarser granularity at the upper level, it is still undesirable to have a large image region overly under-represented since there may be weak boundaries present. Therefore, we generate more line segments and vertices as follows. Set up a uniform sparse grid over the image plane with its spacing the same as the distance threshold used in vectorization. Inside each grid cell, we find locally strongest edges that were missed during edge detection when some global thresholds on filter responses are typically applied. Vectorize these weak edges, too. Line segments thus generated are also collected as hard edges. If a grid cell is completely featureless, an isolated vertex with a randomized location is generated within the cell. This last step has resemblance to the sampling step in [10].

Once we have the set of line segments and isolated vertices, a constrained Delaunay triangulation (CDT) can be constructed. The vertex set of the CDT consists solely of the predefined vertices and all the endpoints of the line segments. It is guaranteed that the predefined line segments are actually edges in the triangulation. Under these constraints, the CDT is the optimal triangulation in the sense that the minimal angle in the triangulation is maximized. We use the software package, TRIANGLE [11], to generate the CDT. The CDT represents the coarse structure we adopt for the upper level. Note that not all edges in the CDT have associated edge segments in the lower level.

There are a few reasons why we choose the above CDT as the coarse structure instead of choosing an over-segmentation of the original image using a local region-growing scheme. First, region-growing needs a stopping criterion which is not trivial to define. Second, region-growing does not work well for texture images with high contrast local variations. Third, region-growing may extend a local region beyond a low contrast gap on a boundary, thus, connect the neighborhoods on both sides of the boundary.

2.3 Triangulation Segmentation by Normalized Cut

Our main segmentation is performed on the coarse structure defined by the obtained CDT. Therefore, the pixels belonging to the same edge segment or triangular region are not going to be separated. Obviously, we would like the final segmentation to pass through as many hard edges as possible. Once a hard edge is part of a boundary, its associated edge segment in the original image immediately provides accurate localization of that part of the boundary.

A weighted graph $G = (V, E)$ can be defined on the CDT. The nodes represent the set of triangles. There is a graph edge between two nearby triangles. Additional graph edges connecting distant triangles may also be defined. The normalized cut algorithm [2] can be applied to this graph to segment the set of triangles into multiple groups with each group having coherent attributes.

We first introduce some details of the normalized cut algorithm where the weight on an edge measures the similarity of the attributes at the two nodes. The idea is to partition the nodes into two subsets, A and B, such that the following disassociation measure, the normalized cut, is minimized.

$$Ncut(A, B) = \frac{cut(A, B)}{asso(A, V)} + \frac{cut(A, B)}{asso(B, V)} \tag{1}$$

where $cut(A, B) = \sum_{u \in A, v \in B} w(u, v)$ is the total connection from nodes in A to nodes in B; $asso(A, V) = \sum_{s \in A, t \in V} w(s, t)$ is the total connection from nodes in A to all nodes in the graph; and $asso(B, V)$ is similarly defined. To compute the optimal partition based on the above measure is NP-hard. The second eigenvector of the following generalized eigenvalue system yields a real-valued solution to the normalized cut.

$$(D - W)y = \lambda Dy \tag{2}$$

where D is a diagonal matrix with $D(i, i) = \sum_j w(i, j)$, W is the weight matrix with $W(i, j) = w(i, j)$. A suboptimal partition can be obtained by searching for the best threshold to partition the real-valued elements of y into two subgroups. The two resulting subregions from this partition can be recursively considered for further subdivision. To improve efficiency, the complete graph defined by the data is usually simplified to only have edges that connect two nearby nodes.

In our current context, the attributes at a graph node are collected from its corresponding triangle which in turn has a set of corresponding pixels in the original image. Because the edge segments in the original image may not be straight, the set of pixels in the lower level corresponding to a triangle most likely form a nontriangular region. The intensity of the node is the average intensity of all the pixels in that region. Because there are much fewer triangles than the number of pixels, we can build a much denser graph over the set of triangles while still maintaining high efficiency. The weight $w(u, v)$ between triangles T_u and T_v is the product of a similarity term $S(u, v)$ and a proximity term $P(u, v)$.

$$S(u, v) = \exp(-d_s^2(u, v)/2\sigma_s^2) \tag{3}$$

where $d_s(u, v)$ measures the difference of the average intensities.

The proximity term over an edge (u, v) is used to model spatial coherence.

$$P(u, v) = c(u, v) \cdot \exp(-dist^2(u, v)/2\sigma_p^2) \tag{4}$$

where $dist(u, v)$ is some distance measure and $c(u, v)$ is the connectivity between the two triangles. We connect the centroids of the two triangles and check whether this connection intersects with any hard edges. $c(u, v) = 0$ if there are at least one intersections and $c(u, v) = 1$ otherwise. We found out through experiments that this binary definition of $c(u, v)$ can better force a segmentation to go through hard edges than a continuous definition. We adopt the Euclidean distance between the centroids for $dist(u, v)$.

The constrained Delaunay triangulation is partitioned into multiple groups of triangles after normalized cut is performed. We take this grouping result as the final segmentation of the triangulation.

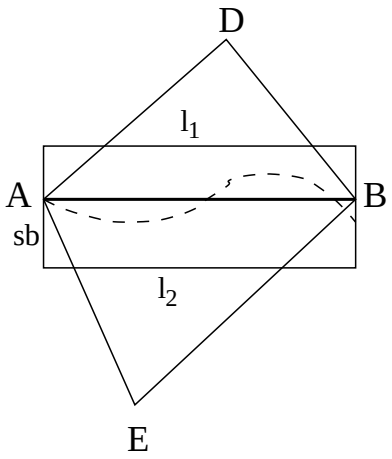

Fig. 2. Boundary refinement at the lower level. A rectangular region surrounding the edge AB to be refined is marked.

2.4 Lower Level Boundary Refinement

Once the segmentation at the upper level is finished, the hard edges on the upper level boundaries already have corresponding edge segments at the lower level while the rest of the upper level boundary edges still need to find out their counterparts at the lower level. We simply project an upper level boundary onto the lower level and search for a new boundary with pixel accuracy in its vicinity. There are potentially many techniques that can be applied for this purpose, such as snakes[12] and dynamic programming[13]. In practice, we choose to adopt the generalized graph minimum $s - t$ cut [14].

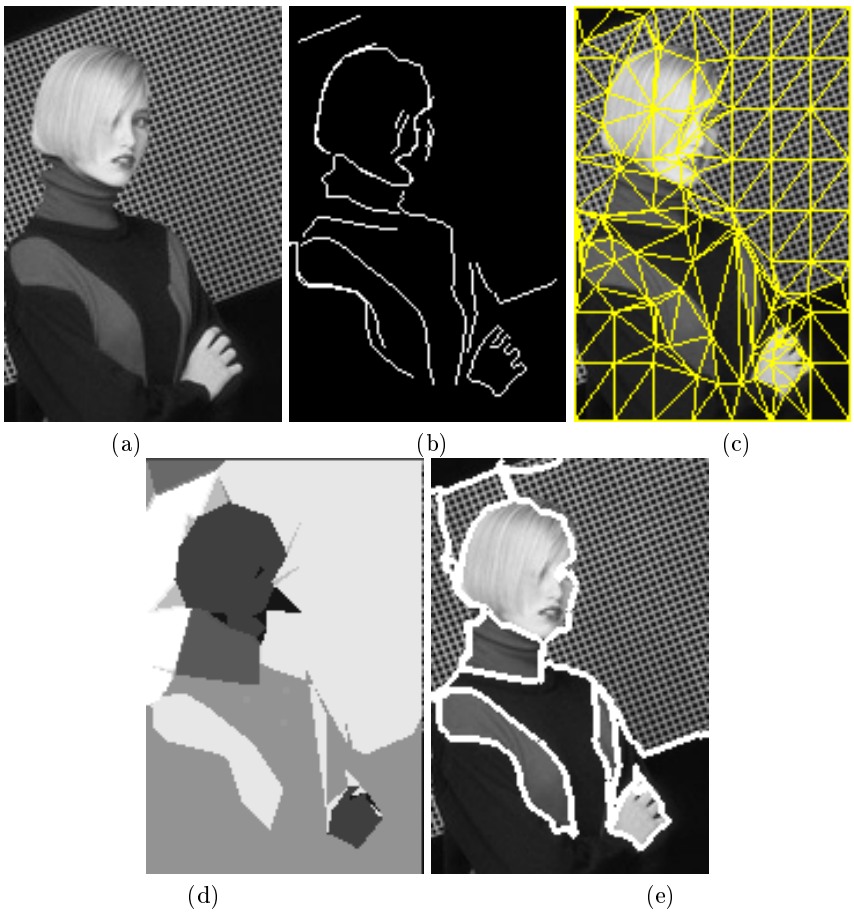

Fig. 3. (a) an original image; (b) final edge detection results; (c) the constrained Delaunay triangulation; (d) a segmentation of the CDT (triangles in the same group have the same intensity); (e) final image segmentation with refined boundaries.

If we partition a weighted graph G into two subgraphs G_1 and G_2 by enforcing minimum cost on the cut, it is well known that we can easily obtain unbalanced results with one of the subgraphs only containing a small number of nodes. One method to remedy this needs some prior knowledge of the two resulting subgraphs. It needs to know a subset of the nodes in G_1 as well as a subset in G_2. Assume the unknown desired partition of $G = (V, E)$ is $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ with $V = V_1 \cup V_2$, and two subsets of nodes, $V_1^s \subseteq V_1$ and $V_2^s \subseteq V_2$, are known. The problem becomes finding an optimal partition of the difference set $V - V_1^s - V_2^s$. If both V_1^s and V_2^s have only one node, this is the standard s-t minimum cut problem and can be solved using network maximum flow algorithms[15] . If at least one of V_1^s and V_2^s has multiple nodes, the nodes need to be merged to create a single new node. Because of the merge, a new

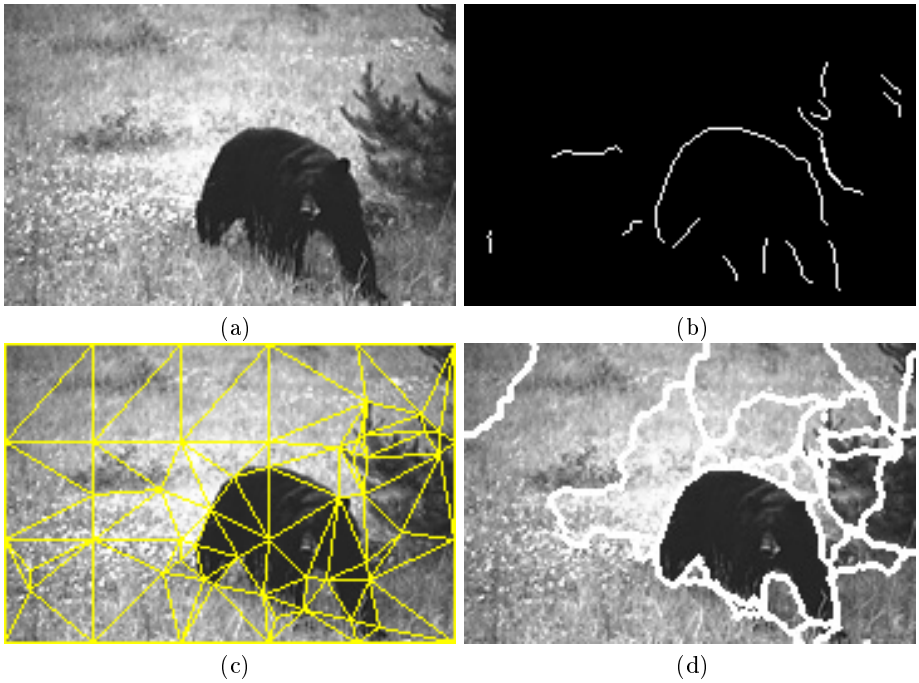

Fig. 4. (a) an original image; (b) final edge detection results; (c) the constrained Delaunay triangulation; (d) final image segmentation with refined boundaries.

graph $G' = (V', E')$ is created based on the original one. In the new graph, $V' = (V - V_1^s - V_2^s) \cup \{s\} \cup \{t\}$ where the node s replaces V_1^s and t replaces V_2^s. If an edge $(u, v) \in E$ with $u, v \in V_1^s$, it is removed. If an edge connects V_1^s and $V - V_1^s$, it becomes incident to s. Anything related to t is also treated similarly. It can be easily proven that the s-t minimum cut of the transformed graph is actually the minimum cut of $V - V_1^s - V_2^s$ in the original graph[14].

Our method to refine a boundary is as follows. Suppose AB is a boundary edge in the triangulation, and it is shared by two triangles $\triangle ABD$ and $\triangle ABE$ (Fig. 2). A rectangular image region surrounding AB and with one of its axes parallel to AB is marked as the potential region containing the accurate boundary. The width of the rectangle should be comparable to the average height of the triangles to ensure sufficient coverage of the pixels in them. A graph can be built for this rectangular region with the set of covered pixels as its nodes. Each node is connected to its eight neighbors. For every pair of adjacent nodes, their edge weight is the similarity between the intensities. Before minimum cut is performed, all the pixels on l_1 are merged into a single node s, and those on l_2 are merged into t. The minimum s-t cut (the dashed line in Fig. 2) thus obtained is guaranteed to enter the region at one of the side borders and exits at the other.

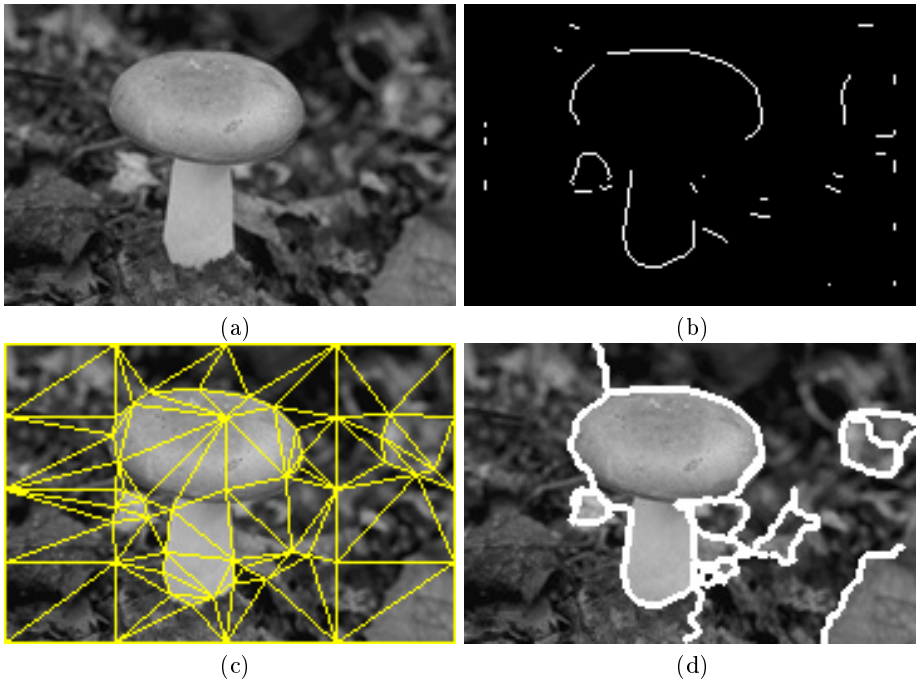

Fig. 5. (a) an original image; (b) final edge detection results; (c) the constrained Delaunay triangulation; (d) final image segmentation with refined boundaries.

If multiple consecutive triangle edges need refinement at the same time, the refined edges should still be connected. The edges are still processed in a sequential order. Suppose two triangle edges e_1 and e_2 join at vertex v. During the refinement of e_1, the new position of v, v', is going to be determined. When e_2 is up for refinement, v' should be fixed. Since v' should be close to one of the side borders, sb, of the surrounding rectangle of e_2, this fixed endpoint can be enforced by merging pixels on the side border sb into either node s or t depending on the part of sb they lie on.

3 Results and Conclusions

We have successfully run our algorithm on a variety of natural images. Figure 3 to 5 show typical segmentation results. σ_s in (3) is set to 25 if there are 256 grey scales for the image intensity, and σ_p in (4) is set to 50 pixels. In all the cases, the foreground object is cleanly separated from the background. However, there are oversegmented regions in the background. Color information has not been used in these examples.

In summary, we have introduced a two-level approach for image segmentation based on region and edge integration. Edges are first detected in the original

image. To preserve the spatial coherence of the edges and their surrounding image regions, the detected edges are vectorized into connected line segments which serve as the basis for a constrained Delaunay triangulation where grouping is actually carried out. Our method favors segmentations that pass through more detected edges in the original image.

Acknowledgment

This work was supported by National Science Foundation CCR-0132970.

References

1. Wu, Z., Leahy, R.: An optimal graph theoretic approach to data clustering: Theory and its application to image segmentation. IEEE Trans. Pat. Anal. Mach. Intell. **11** (1993) 1101–1113
2. Shi, J., Malik, J.: Normalized cuts and image segmentation. In: IEEE Conf. on Computer Vision and Pattern Recognition. (1997) 731–737
3. Freixenet, J., Munoz, X., Raba, D., Marti, J., Cufi, X.: Yet another survey on image segmentation: Region and boundary information integration. In: European Conf. Computer Vision. (2002) 408–422
4. Leung, T., Malik, J.: Contour continuity in region based image segmentation. In: Fifth European Conf. on Computer Vision. (1998)
5. Malik, J., Belongie, S., Leung, T., Shi, J.: Contour and texture analysis for image segmentation. Int'l Journal of Computer Vision **43** (2001) 7–27
6. Canny, J.: A computational approach to edge detection. IEEE Trans. Pat. Anal. Mach. Intell. **8** (1986) 679–698
7. Ma, W., Manjunath, B.: Edgeflow: a technique for boundary detection and image segmentation. IEEE Transactions on Image Processing **9** (2000) 1375–88
8. Heeger, D., Bergen, J.: Pyramid-based texture analysis/synthesis. In: Proc. of SIGGRAPH. (1995) 229–238
9. Martin, D., Fowlkes, C., Malik, J.: Learning to detect natural image boundaries using brightness and texture. In: Neural Information Processing Systems(NIPS). (2002)
10. Belongie, S., Fowlkes, C., Chung, F., Malik, J.: Spectral partitioning with indefinite kernels using the nystrom extension. In: European Conf. Computer Vision. (2002) 531–542
11. Shewchuk, J.: Triangle: Engineering a 2d quality mesh generator and delaunay triangulator. In: First Workshop on Applied Computational Geometry. (1996) 124–133
12. Kass, M., Witkin, A., Terzopoulos, D.: Snakes: Active contour models. Int'l Journal of Computer Vision (1988) 321–331
13. Mortensen, E., Barrett, W.: Intelligent scissors for image composition. In: SIGGRAPH 95 Proceedings. (1995) 191–198
14. Xu, N., Ahuja, N.: Object contour tracking using graph cuts based active contours. In: IEEE Conf. on Image Processing. (2002) 277–280
15. Ahuja, R., Magnanti, T., Orlin, J.: Network Flows. Prentice Hall, Inc. (1993)