Segmenting "Simple" Objects Using RGB-D

Ajay K. Mishra
http://www.vision.inst.ac.uk/~ss
Yiannis Aloimonos
http://www.vision.inst.ac.uk/~pp

Computer Vision Laboratory
University of Maryland
College Park, MD 20742

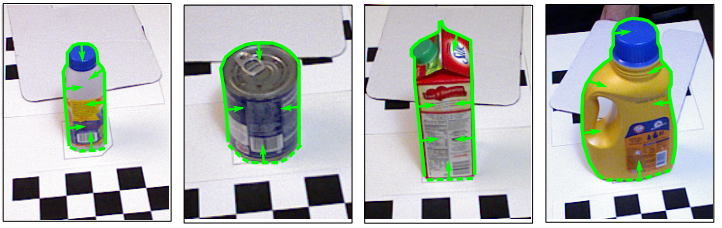
Abstract

Segmenting "simple" objects using low-level visual cues is an important capability for the unsupervised learning of a vision system. We define a "simple" object as a compact region enclosed by depth and/or contact boundary in the scene and propose a segmentation process to extracts all the "simple" objects in the scene using both color and depth information. The proposed segmentation process builds on the fixation-based segmentation framework that segments a region given a point anywhere inside that region. In this work, we augment that framework with a fixation strategy to automatically select points inside the "simple" objects and a post-segmentation process to select only the regions corresponding to the "simple" objects in the scene. The proposed segmentation process has been evaluated on more than 400 test cases and the results show the promise of the method.

1 Introduction

Over the last four decades of vision research, a number of segmentation algorithms have been proposedShi and Malik [11], Rother et al. [10]. Despite this tremendous progress, we still lack an approach that takes an image/scene as its input and outputs only the regions corresponding to the objects in that scene using only the low-level visual cues such as color and depth. In fact, the most common application of segmentation in recent computer vision algorithms is to oversegment a scene into small regions which are then used for high-level semantic analysisAyvaci [2], Hoiem et al. [6], B. Micusik [3]. While these oversegmented regions are more informative than pixels, there is no certainty about the number of these regions that have to be combined to form any object in the scene. Recognizing this as a major problem of segmentation algorithms, we propose a process to segment each object in a scene as one region in a purely bottom-up fashion.

Since we are using only the low-level cues, we have to, first, come up with a definition of an object without using any high-level semantics. Let us consider the four objects in Figure 1a. While the objects are perceptually distinct, they all share one common property: depth varies smoothly inside the objects and abruptly across their boundary (except across the portion of the boundary touching the surface). This observation motivated us to define a "simple" object as a compact region enclosed by the depth and/or contact boundary pixels in the scene (Contact boundary is detected as the location with abrupt change in surface orientation. More about detecting both types of boundaries is in section 2). Since a "simple"



(a) "Simple" objects

(b) A complex object

Figure 1

object is defined in a bottom-up manner, a complex object such as a chair will constitute of mutiple "simple" objects as its parts (see Figure 1b).

Our process to segment a "simple" object is built on the fixation-based segmentation approach proposed by Mishra *et al*. [8] which takes a point (called "fixation" in an anthromorphic sense) anywhere inside a "simple" object as its input and outputs the closed boundary around that point without being affected by the scale of that object. But[8] does not propose a strategy to automatically select the points inside different "simple" objects in the scene. We propose a strategy to automatically select the points inside different "simple" objects without knowing their actual size or number. Since the selection process is based on local processing only, multiple points are usually selected inside the same "simple" object. So we also propose a method to pick only those closed contours that uniquely correspond to the "simple" objects in the scene.

An essential component of the proposed process is using the concept of "border-ownership" which is the knowledge of the side a boundary edge pixel belongs to. In neuroscience literature[4, 12], the evidence that a neuron at a boundary edge location records the information about the "object" side has been reported. The object-side information not only helps us select points inside objects but also helps us identify the closed contours correspoding to the "simple" objects in the scene. This concept, also known as figure/ground cue, has been studied in [4, 5], but they use only monocular cues to make that decision. [12] reports that depth cues are stronger than monocular cues in making unambiguous decision about the border-ownership of a boundary edge pixel.

The rest of the paper is organized as follows: in section 2, the method to localize depth and contact boundaries using color, texture and depth information is explained. Following this, we describe our strategy to select the points inside the probable objects in section 3 and an overview of [8] that finds the closed contour for each point in section 4. From the set of closed contours segmented for all of the selected points, we extract the subset of contours corresponding to the objects using a min-cover formulation explained in section 5. Finally, we present the experiments with more 400 scenes with objects in section 6. The results demonstrate the reliability of the proposed method in extracting objects from a real scene.

2 Generating Probabilistic Boundary Edge Map

We start by generating a probabilistic boundary edge map of a scene using color, texture and depth cues. In the probabilistic boundary edge map of a scene, the intensity of a pixel is the probability of that pixel to be at the boundary of any "simple" object in the scene. The

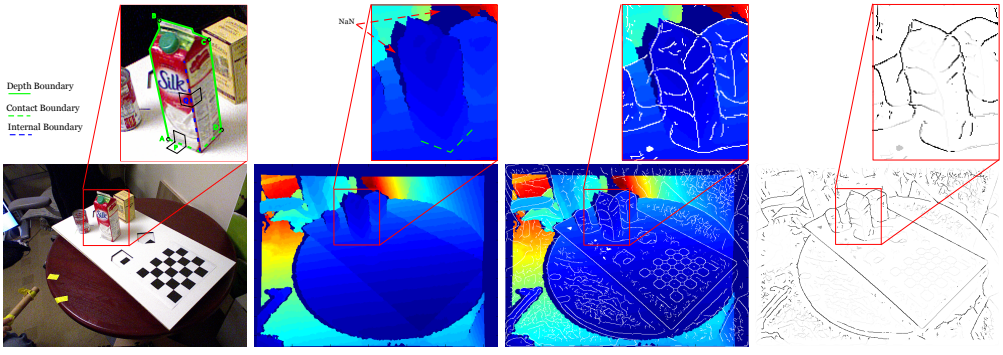


Figure 2: Generating Probabilistic Boundary Edge Map. (a) The depth and contact boundary of the milk bag in the color image is shown on the top. (b) Color coded depth map. NaNs are where the values are not known. (c)The edge pixels found using color and texture cues are overlaid on to the depth map. (d) probabilistic boundary edge map of the scene.

pixels at the object boundary are of two types: depth and contact boundary. The pixels at a depth boundary could be found by checking for discontinuities in the depth map. But the exact location of discontinuities in the depth maps often do not match with the true object boundaries in the RGB image. This is why we are going to use color and texture cues to faithfully find all possible boundary locations which are essentially the edge pixels with positive color and/or texture gradient. Then, the probability of these edge pixels to be at depth or contact boundary is obtained using depth information.

2.1 Boundary Localization

We find all possible boundary locations as the pixels in the image with positive color and texture information using [7]. We form a binary edge map using these locations. At each of these edge pixels, the dominant tangential direction, which is one of eight possible quantized values between 0 and π, is calculated as well. The localization process has selected only a subset of all pixels which also means the pixels from inside smooth areas in the scene have already been assigned zero probability to be on the boundary of an object. However, the probability of the possible boundary edge pixels can not be estimating using color or texture gradient at these locations because an edge pixel with a high color and/or texture gradient can be both inside and at the boundary of a object. So we use depth information to determine the boundary probability of the edge pixels. In Figure 2c, the white pixels are the possible boundary locations overlaid on the depth map of the scene.

2.2 Boundary Probability Estimation

There are two kinds of pixels likely to be at an object boundary: the ones with a significant depth discontinuity across them, and the other at the contact between the object and the surface. We estimate the probability of each edge pixel to both types of object boundaries and assign it the maximum of the two probabilities. The final probabilistic boundary edge map is represented by P_B such that $0 \leq P_B(x,y) \leq 1$. See Figure 2d for an example.

Depth boundary

The pixels at a depth boundary can be found in two ways: first, by measuring the difference in the depth values from its two sides, and the other is by occlusion. Occlusion sometimes create region with unknown depth estimate on the one side of the edge pixel, which can be used to detect a boundary edge pixel. But it is only a small fraction of all boundary pixels. Most edge pixels at the boundary has significant depth change across it.

To compute the depth gradient across an edge pixel, we place the appropriately oriented disc filter on that edge pixel and calculate the absolute difference Δd of the mean depth values of the pixels on the $+$ and $-$ sides of the filter (see Figure for the filters oriented along 8 possible edge orientations). We then use a logistic function $L : \mathbb{R}_+ \mapsto [0,1]$ to then convert Δd into the probability value:

$$L(\Delta d) = \frac{1}{1 + e^{-\beta_1(\Delta d - \beta_2)}}$$

where β_1 and β_2 are learned using training examples.

Since we basically have to differentiate between the internal edges from the boundary edge pixels, we collect the depth gradient values from both the pixels along the boundary of a manually segmented object and those from inside the object. Specifically, we select 200 pixels from inside and at the boundary of an object. We collect the data from 10 different objects at different distances from the camera. These gradient values form a training set by associating the gradient scores for the boundary pixels with ouput 1 and those for the internal edges with 0. We then use logistic regression to find the values of the parameter β_1 and β_2 which are found to be 74.0 and 0.02 respectively in our case.

Contact boundary

A pixel at a contact boundary does not have a significant depth change across it because the object touches the surface and depth changes gradually across it. But the contact boundary makes up sizeable portion of the total object boundary. So, how can we detect it?

Although the depth does not change significantly across a contact boundary, the pixels on the two sides generally form orthogonal planes. Additionally, one of these planes have to be parallel to the horizonal plane in the camera reference frame. For a robotic system, estimating the normal of the horizontal plane is an easy task. Besides, we only need a rough estimate of the normal. Just like when we estimated the probability of an edge pixel to be depth boundary, we place the oriented filters on an edge pixel and estimate the planes using (x,y,z) co-ordinates of the pixels on $+$ and $-$ sides of the filter. We check whether the two planes are orthogonal and one of them is parallel to the horizontal plane. If both conditions hold, we declare those edge pixels contact boundary and are assigned probability 1.

3 Fixation Strategy

The objective is to select points inside objects so that the fixation-based segmentation method[8] finds the closed boundaries enclosing those points. But selecting the points inside objects even before segmenting the objects appears as a chicken and egg problem. The solution is in the logic we used previously to generate the probabilistic boundary edge map P_B.

An edge pixel in P_B is more likely to be on a depth boundary if there is a significant difference in depth on its two sides. Additonally, we can identify the side closer to the

camera or have relatively low Z value as the "object" side. So we first pick the edge pixels with the boundary probability greater than 0.5 and assume that they lie at the boundary of some object in the image. We call this binary edge map I_B. We then assign the "object" side information for the edge pixels in I_B by comparing the mean depth on the two sides. The object information is stored in O_B defined as:

$$O_B(x,y) = \begin{cases} +1 & \text{if } I_B(x,y) \neq 0 \,\&\, Z_+ > Z_- \\ -1 & \text{if } I_B(x,y) \neq 0 \,\&\, Z_- > Z_+ \\ 0 & \text{if } I_B(x,y) = 0 \end{cases}$$

where Z_+ and Z_- are mean depth values on the positive and negative side of the edge pixel.

With O_B, we can select the points inside objects by moving a fixed distance in the normal direction from the edge pixels in I_B towards its "object" side. But selecting a point for each pixel in I_B will cause enormous redundancy in the segmentation output as many fixations inside the same object will result in the similar segmentation. To reduce redundancy, we break I_B into edge fragments as using end points and junctions. Now, instead of edge pixels, we now select a point for each fragment by moving towards the "object side" in the normal direction at the middle of the fragment. This point is selected at a fixed distance of 20 px from the edge pixel.

4 Fixation based Segmentation

For each selected point, the fixation-based segmentation approach [8] finds the closed boundary around the given point by combining the edge pixels in the probabilistic boundary edge. The segmentation for each point has two intermediate steps: first, the probabilistic boundary edge map P_B is converted from the Cartesian to the Polar space using the given point as the pole for the conversion, in order to achieve scale invariance. Following this, a binary segmentation of the polar edge map generates the optimal path through the polar edge map such that when the curve is mapped back to the original Cartesian space, it encloses the point. The two-step segmentation process is repeated for all fixation points found in section 3 using the same P_B. The source code for this segmentation algorithm was obtained from the author's website.

5 Selecting "Simple" objects

We have as many closed contours as the number points selected by the fixation strategy. Since the points are select using edge fragments in I_B, multiple edge fragments that are part of the same object boundary select multiple points inside the same object and segmentation for these points result in duplicate closed contours. Some points are selected outside of any object due to the error in "object" side estimation for some edge fragment. Segmenting for these points however result in closed contours that do not correspond to any object in the scene. We need a process that sifts through all of these closed contours to pick only the ones that uniquely correspond to the objects in the scene. To do that should be able to compare two closed contours and decide which one is more likely to be an object compared to the other. We compare contours on the basis of its coverage as defined in section 5.1. Using coverage as our basis, we describe our minimum set cover formulation to select the subset of closed contours that correspond to the "simple" objects in the scene (section 5.2.)

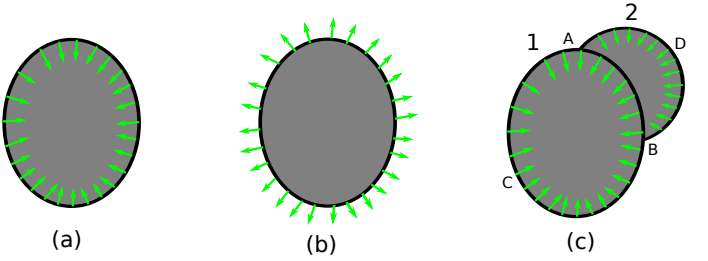


Figure 3: (a) and (b) are the closed contours likely to a "simple" object and a hole respectively. (b) A closed contour hole. (c) Two occluding closed contours. Note that the arrows indicate the "object" side of the boundary pixels.

Notation: I_C^i is the binary mask representing the i^{th} closed contour whose interior is represented by another binary mask I_R^i. If I is a 2D matrix with binary entries, X_I is the set of 2D coordinates of the non-zero pixels in I.

5.1 Coverage of a Closed Contour

We define the coverage of a closed contour such that a high coverage means the contour is more likely to correspond to the boundary of a "simple" object. A closed boundary of a "simple" object is composed of those edge pixels in I_B whose "object" side is inside the contour (see Figure 3(a) for an example). The opposite is true for the closed boundary of a hole. The "object" side of their boundary edge pixels lie outside of the contour (see Figure.3(b) for an example of a hole). The coverage of a closed contour is defined as:

$$Coverage(I_C, I_R) = \frac{1}{|X_{I_C}|} \sum_{x \in X_{I_C}} I_B(x)\Pi(x) \quad (1)$$

$$\Pi(x) = \begin{cases} +1 & \text{if } I_R(x + \lambda O_B(x) u_\perp) \neq 0 \\ -1 & \text{otherwise} \end{cases}$$

$$u_\perp = \begin{bmatrix} cos(\theta - \frac{\pi}{2}) \\ sin(\theta - \frac{\pi}{2}) \end{bmatrix}$$

where x is a 2D coordinate, θ is the orientation of the tangential direction at the edge pixel x and λ is the distance from the selected point on the "object" side to the edge pixel. In our experiment, we keep λ to be 5 px.

5.2 Selection Closed Contours of "Simple" Objects

The characterstic of a closed contour outside of any "simple" object is that most of its boundary edge pixels has their "object" side outside of the contour and thus its coverage is either negative or close to zero if positive. Duplicate closed contours traces almost the same boundary pixels, so we can pick the one among them with highest coverage. Keeping these in mind, we formulate the process of selecting the closed boundaries of "simple" objects from the set of all closed contours as a type of a min-cover problem, whose standard definition is:

Given an Universal set U and a set of subsets S, find $S' \subseteq S$ such that S' contains all the elements of U and $|S'|$ is minimum.

In our case, X_{I_B} is the universal set U. $\{X_{I_C^i}\}_{i=1}^{n}$ is the set of subsets since $X_{I_C^i}$, the pixels along each closed contour, contains a subset of pixels in X_{I_B}. n is the total number of closed contours. Our objective is to find the minimum number of closed contours that together trace almost all the pixels in I_B. Since the minimum set cover problem is NP-Complete, we propose a greedy solution. The pseudo-code is given in Algorithm 1. The selected closed contours at the end of the process are the boundaries of the different "simple" objects in the scene.

The greedy solution works iteratively. It computes the coverage of all closed contours and then selects the contour with best coverage in each iteration. At the end of the iteration, the universal set is updated by eliminating all the pixels traced by the current best contour. After updating, the coverage of the remaining contours are recomputed for the next iteration. The selection process repeats until the "best" closed contour in the current iteration has coverage score below a certain threshold.

Algorithm 1 Selecting "good" regions, $t_{coverage} = 0.5$, $t_o = 0.05$

Input:
- I_B ▷ edge pixels predicted to be on object boundaries
- O_B ▷ object side information
- $S_{in} = \{I_R^i, I_C^i\}_{i=1}^{n}$ ▷ closed contours for all n fixations

Output:
- $S_{out} = \{I_R^j, I_C^j\}_{j=1}^{m}$ ▷ final closed contours

Intermediate variables:
- I_T^k ▷ all closed contours traced until iteration (k-1)
- I_M^k ▷ accumulated region mask until iteration (k-1)

begin
Initialize $k \leftarrow 0$; $I_B^0 \leftarrow 0$; $I_A^0 \leftarrow 0$
while $|S_{in}| > 0$ **do**
  Compute coverage of all closed contours $\in S_{in}$;
  Let b be the index of the closed contour with maximum coverage;
  if $Coverage(I_R^b, I_C^b) < t_{coverage}$ **then**
    Exit the loop;
  else
    Move the closed contour from S_{in} to S_{out};
    $I_A^k \leftarrow I_A^k + I_R^b$;
    $X_{I_B} \leftarrow X_{I_B} - (X_{I_B} \cap X_{I_C^b})$;
    $X_{I_T^k} \leftarrow (X_{I_T^k} \cup X_{I_C^b})$;
    for $I_C^i \in S_{in}$ **do**
      overlap = $|X_{I_R^i} \cap X_{I_A}| / |X_{I_R^i}|$;
      if overlap $< t_o$ **then**
        $X_{I_C^i} \leftarrow X_{I_C^i} - (X_{I_C^i} \cap X_{I_T^k})$ ▷ to handle occlusion
      end if
    end for
    Increment k;
  end if
end while

An important step in the proposed greedy solution is the the update of the remaining closed contours at the end of each iteration. To handle occlusion between contours: when one of them is selected as the best contour in an iteration, the remaining closed contours are updated such that the pixels on the shared boundary do not affect their coverage as those pixels have already been traced by the current best contour. Consider, for instance, the two occluding closed contours *ABCA* and *ADBA* in Figure 3(c). They share the pixels along the segment *AB* and the "object" side indicates it will contribute positively to the coverage of *ABCA* and negatively to that of contour *ADBA*. After the selection of *ABCA*, in the next

iteration, we would like to make *ADBA* prominent because the boundary portion *AB* that was contributing negatively has already been traced by region *ABCA*. So, the new coverage of *ADBA* will include the contribution from the remaining pixels in the segment *ADB*. But, this could have unintended consequence for overlapping duplicate contours where once the best contour for the "simple" object is selected, all the duplicate contours with a slight change will become important just as the occluding contour case. To avoid this situation, we check for the overlap of the inside of the remaining closed contours with that of already selected closed contours. Only the closed contours with no overlap are updated.

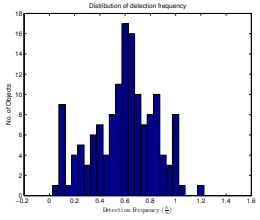
6 Experiments

To evaluate the performance of the proposed segmentation process, we use the publicaly available dataset of RGB images (along with 3D information) at the solutionsInPerception-Challenge[1]. This dataset has 24 test scenarios. In each case, a table with 6 objects placed on it is viewed from an average of ≈ 15 different viewpoints around the table using a depth sensor named Kinect. For each viewpoint, the sensor output is a 640×480 matrix where each element represented by (r,g,b,x,y,z) contains both color and 3D information at each pixel in the scene. In the dataset, the viewpoint is changed by rotating the table holding the objects. Also, the orientation of the depth sensor remains fixed and the vertical direction in the sensor's reference frame is $\hat{n} = \{0.8026, -0.0155, -0.5964\}^T$ which is used to contact boundary (as explained in section 2).

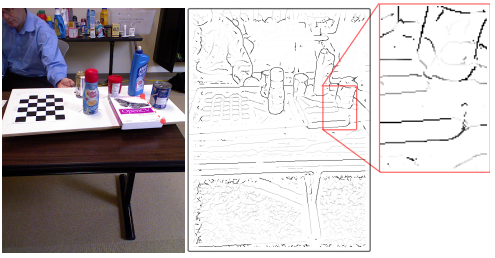
In each viewpoint, only a subset of objects on the table are visible. The top row (3^{rd}col) of Figure 5 shows how the "Clorex" bottle is completely blocking the object behind it which means that object can not be segmented. So, for each object i in a test case, we count the number of viewpoint n_i in which more than 50% the object area is visible. We count the number of viewpoints k_i in which our segmentation method detects the object. Using these two numbers, we compute the detection frequency k_i/n_i. The detection frequency ≥ 1 means the object is not only segmented exactly in all the viewpoints it is sufficiently visible but also in the cases when it is heavily occluded. An object is considered detected if the overlap $\frac{|R_c \cap O_i|}{|R_c \cup O_i|} > 0.95$ where O_i is the manually segmented object mask, and R_c is the interior of the closed contour $c \in C$ and C is the set of all closed contours found by the proposed method for that viewpoint.

We consider each object instance as a distinct object. So there are total of $144 = 24 \times 6$ objects in the dataset. We create a histogram of their detection frequency as given in Figure 4a. The average relative frequency for the entire dataset is 0.59. So, statistically speaking, if a vision system looks at an object on a table from 2 different viewpoints and the object is visible in both of them, our segmentation method will segment that object as one region in at least one of those viewponts. As shown in Figure 4a, there are some objects with low relative frequency. These objects are the repeated instance of an "OpenCV" book in different test cases. See the left image in Figure 4b for an example. The primary reason for the poor detection of this object is missing color-based edge pixels at its boundary. A significant portion of the boundary, marked by dotted red in the left image, is missing completely in the edge map as shown in the rightmost image of Figure 4b. This makes it difficult for the segmentation step to form the correct closed boundary around the "OpenCV" object repeatedly. An example of when this object is detected is given in the second column of Figure 5 in a different test case.

Figure 5 shows a number of examples when almost all objects are visible in these views,



(a) Detection Frequency



(b) The "OpenCV" book in test #15, Viewpoint #13

Figure 4

Figure 5: In the bottom row, the output of our segmentation process, closed contours around the "simple" objects, for a scene shown right above the results.

the segmentation process extracted the closed contours around those objects. The results show that the algorithm is able to tolerate occlusion. Finally, the algorithm also returns closed contours corresponding to other "simple" objects such as hand of the person in Figure 4b which are not on the table. A few examples of these types of closed contours are given in the supplementary material. Our method is purely driven by low-level cues, so it is upto recognition module to pick those that are important for the job at hand. Our algorithm returns on average ≈ 8 closed contours for each viewpoint.

7 Conclusion

We presented a segmentation process that takes an RGB image along 3D information about a scene and returns a set of closed contours corresponding to "simple" objects in the scene. These regions can be used a mid-level visual cue in recognition, tracking, object manupulation and many other complicated visual tasks. In particular, object recognition algorithms can now make us of boundary information explicitly something that is not available when

image patches are used.

References

[1] Solutionsinperceptionchallenge. http:/opencv.willowgarage.com/wiki/solutionsinperceptioncl

[2] S. Ayvaci, A.and Soatto. Motion segmentation with occlusions on the superpixel graph. In *Proc. of the Workshop on Dynamical Vision, Kyoto, Japan*, October 2009.

[3] J. Kosecka B. Micusik. Semantic segmentation of street scenes by superpixel co-occurrence and 3d geometry. In *IEEE Workshop on Video-Oriented Object and Event Classification, ICCV*, 2009.

[4] Edward Craft, Hartmut Schütze, Ernst Niebur, and Rüdiger von der Heydt. A neural model of figure-ground organization. *Journal of Neurophysiology*, 6(97):4310–4326, 2007. ISSN 0162-8828.

[5] C.C. Fowlkes, David R. M., and J. Malik. Local figure/ground cues are valid for natural images. *JV*, 7(8):1–9, 2007.

[6] Derek Hoiem, Alexei A. Efros, and Martial Hebert. Automatic photo pop-up. *ACM Trans. Graph.*, 24(3):577–584, 2005. ISSN 0730-0301. doi: http://doi.acm.org/10.1145/1073204.1073232.

[7] D. Martin, C. Fowlkes, and J. Malik. Learning to detect natural image boundaries using local brightness, color and texture cues. *T-PAMI*, 26(5):530–549, May 2004.

[8] A. Mishra, Y. Aloimonos, and L. F. Cheong. Active segmentation with fixation. In *ICCV*, 2009.

[9] X.F. Ren, C.C. Fowlkes, and J. Malik. Figure/ground assignment in natural images. pages II: 614–627, 2006.

[10] Carsten Rother, Vladimir Kolmogorov, and Andrew Blake. "GrabCut": interactive foreground extraction using iterated graph cuts. *ACM Trans. Graph.*, 23(3):309–314, 2004.

[11] Jianbo Shi and Jitendra Malik. Normalized cuts and image segmentation. *T-PAMI*, 22 (8):888–905, 2000. ISSN 0920-5691.

[12] H. Zhou, H.S. Friedman, and R. von der Heydt. Coding of border ownership in monkey visual cortex. *The Journal of Neuroscience*, 20:6594–6611, September, 2000.