Implementing GrabCut

Justin F. Talbot*
Brigham Young University

Xiaoqian Xu†
Brigham Young University

Abstract

GrabCut is an innovative 2D image segmentation technique developed by Rother et al. [2004]. This paper provides implementation details omitted from the original paper. Details covered in background papers are summarized here so that future implementors can refer to a single paper. Our implementation of GrabCut is described and results are included. Our main contribution is correcting errors in equation (9) and in equation (11) of the original paper. We also discuss weaknesses of the algorithm that were not discussed in the original paper. We present possible research directions to address these problems.

CR Categories: I.4.6 [Image Processing and Computer Vision]: Segmentation—Pixel classification

Keywords: GrabCut, Graph Cut, Object Selection, Alpha Matting

1 Introduction

GrabCut [Rother et al. 2004] is an iterative image segmentation technique based upon the Graph Cut algorithm [Boykov and Jolly 2001]. GrabCut extends Graph Cut to color images and to incomplete trimaps. These developments greatly increase the usefulness of Graph Cut. User interaction is simplified to drawing a rectangle around the desired foreground, followed by a small amount of corrective editing. The inclusion of color information in the Graph Cut algorithm and the iterative learning approach increases its robustness. Thus, GrabCut is a very promising image editing tool for foreground extraction.

This paper presents an accessible description of a GrabCut implementation. Only background papers directly used in the implementation are cited. Rother et al. [2004] and Blake et al. [2004] contain more background information.

2 GrabCut Algorithm

The GrabCut algorithm is briefly outlined below. Each portion of the algorithm is then described in detail. This discussion of the GrabCut algorithm is based around our implementation of the algorithm. We only implemented the hard segmentation portion of the algorithm, so we do not discuss the border matting portion of the paper. Rother et al. [2004] and to Mortensen and Barrett [1999] contain more information on the border matting technique used.

*e-mail: jft2@email.byu.edu
†e-mail: xiaoqian@et.byu.edu

GrabCut Summary

1. User creates an initial trimap by selecting a rectangle. Pixels inside the rectangle are marked as unknown. Pixels outside of rectangle are marked as known background.
2. Computer creates an initial image segmentation, where all unknown pixels are tentatively placed in the foreground class and all known background pixels are placed in the background class.
3. Gaussian Mixture Models (GMMs) are created for initial foreground and background classes.
4. Each pixel in the foreground class is assigned to the most likely Gaussian component in the foreground GMM. Similarly, each pixel in the background is assigned to the most likely background Gaussian component.
5. The GMMs are thrown away and new GMMs are learned from the pixel sets created in the previous set.
6. A graph is built and Graph Cut is run to find a new tentative foreground and background classification of pixels.
7. Steps 4-6 are repeated until the classification converges.

2.1 Data structures

GrabCut requires four different pieces of information for each pixel. In our implementation, each piece is stored in its own array. Each array is the same size as the original image. Variable names that occur in the original paper appear in parenthesis after the description.

- Color – an RGB value (z)
- Trimap – either TrimapUnknown, TrimapBackground, or TrimapForeground
- Matte – in the initial hard segmentation step, either MatteBackground or MatteForeground (α)
- Component Index – a number between 1 and K, where K is the number of Gaussian components in a GMM (k)

In addition to these, GrabCut also requires K Gaussian components for each of the foreground and background GMMs. For each component, we store:

- μ – the mean (an RGB triple)
- Σ^{-1} – the inverse of the covariance matrix (a 3x3 matrix)
- $\det\Sigma$ – the determinant of the covariance matrix (a real)
- π – a component weight (a real)

Section 3.3 covers details on how these values are computed in our implementation.

2.2 Initialization

The initialization process includes steps 1-3. In step 1, the user initializes the trimap by selecting a rectangular region around the

object of interest. Pixels inside the rectangle are marked as TrimapUnknown. Pixels outside are marked as TrimapBackground. This is the initial information given to the algorithm. In step 2, the matte is initialized to MatteBackground in the TrimapBackground set and to MatteForeground in the TrimapUnknown set. This distinction between the trimap and the matte formalizes the separation between the user input, which is regarded as correct, and the segmentation derived by the GrabCut algorithm, which may be incorrect.

In step 3, given the initialized matte, we create the K components of the Gaussian Mixture Models (GMM) for the MatteForeground and MatteBackground regions. To clarify, this means that we must create a total of $2K$ components. We first divide both regions into K pixel clusters. The Gaussian components are then initialized from the colors in each cluster. For good separation between foreground and background, it is necessary that we generate low variance Gaussian components. Thus, we seek to find tight, well-separated clusters. There are a wide variety of clustering algorithms that could be used for this step. Guided by Ruzon and Tomasi [2000] and Chuang et al. [2001] we use the color quantization technique described by Orchard and Bouman [1991]. This technique uses the eigenvector of the color covariance matrix to determine good cluster splits. Details of our implementation of this algorithm are found in Section 3.2. Using these initial clusters, we build the Gaussian components as described in Section 3.3.

2.3 Learning GMM components

As we iterate through the learning portion of the algorithm (steps 4-6), the matte will change. This moves some pixels from the MatteForeground to the MatteBackground, and vice versa. When this happens we want to update the GMMs to reflect the new foreground and background color distributions. An obvious way to approach this is to rerun the clustering algorithm described in Section 2.2. While this would work, most clustering algorithms are too slow to run to completion during each iteration. Instead, the GrabCut paper interleaves a simple Gaussian mixture clustering algorithm with the matte update step to speed up the algorithm.

The Gaussian clustering algorithm consists of two steps (4 and 5). First, each pixel in the MatteForeground set is assigned to the foreground GMM component which has the highest likelihood of producing the pixel's color. This is found by simply evaluating the Gaussian equation with pixel's color as input. Similarly, we assign pixels in the MatteBackground set to the highest likelihood component of the background GMM. The Component Index data structure stores the component to which the pixel is assigned. Note that foreground pixels are only assigned to components of the foreground GMM and vice versa. Thus, the pair of values in the Matte (MatteForeground or MatteBackground) and in the Component Index (1 – K) uniquely identifies one of the $2K$ components. We could have just as easily allowed Component Index to hold values $1-2K$. This may be a clearer representation, but we follow the notation of the original paper.

Second, once the pixels have been clustered, we throw away the current Gaussian components and create new ones–one for each Matte/Component Index pair.

2.4 Performing Graph Cut

We next build a graph to use in the Graph Cut algorithm. In Graph Cut, as described by Boykov and Jolly [2001], there are two types of links. N-links connect pixels in the 8-neighborhood. These links describe the penalty for placing a segmentation boundary between the neighboring pixels. We want this penalty to be very high in regions of low gradient and low in regions of high gradient (edges). The N-link weights are constant throughout the execution of the algorithm. Thus, they can be computed once and reused. T-links connect each pixel to the foreground and background nodes. These describe the probability that each pixel belongs to the foreground or to the background. In GrabCut, this probability is contained in the GMMs. As we iterate through steps 4–6, the GMMs update and the probabilities change accordingly. This means that the T-link weights must be updated during each iteration.

For N-links the appropriate link weight between pixel m and n is:

$$N(m,n) = \frac{50}{dist(m,n)} e^{-\beta \|z_m - z_n\|^2} \quad (1)$$

where z_m is the color of pixel m.

This is a restatement of equation (11) in the original paper. We drop the summation since it is implicitly included in the Graph Cut computation. The inverse distance term has been added here as suggested by equation (4) of the original paper. Without the distance term, GrabCut displays a tendancy towards diagonal cuts. We believe that this was mistakenly omitted in equation (11).

There are two T-links for each pixel. The Background T-link connects the pixel to the Background node. The Foreground T-link connects the pixel to the Foreground node. The weights of these links depend on the state of the trimap. If the user has indicated that a particular pixel is definitely foreground or definitely background, we reflect this fact by weighting the links such that the pixel is forced into the appropriate group. For unknown pixels we use the probabilities obtained from the GMMs to set the weights.

The T-link weights for pixel m are:

Pixel Type	Back. T-link	Fore. T-link
$m \in$ TrimapForeground	0	K
$m \in$ TrimapBackground	K	0
$m \in$ TrimapUnknown	$D_{Fore}(m)$	$D_{Back}(m)$

D_{Fore} and D_{Back} are the likelihoods that the pixel belongs to the foreground and background GMMs respectively. These likelihoods are computed as follows for pixel m:

$$D(m) = -\log \sum_{i=1}^{K} \left[\pi(\alpha_m, i) \frac{1}{\sqrt{\det \Sigma(\alpha_m, i)}} \times \exp\left(\frac{1}{2} [z_m - \mu(\alpha_m, i)]^\top \Sigma(\alpha_m, i)^{-1} [z_m - \mu(\alpha_m, i)] \right) \right] \quad (2)$$

where the summation is over the appropriate set of Gaussian components. This equation is a correction to equation (9) in the original paper which left out the summation. The summation is necessary since we are using a Gaussian Mixture Model. Without the sum, GrabCut does not work.

K is a large constant value calculated as follows to ensure that it is the largest weight in the graph:

$$K = \max_m \sum_{n:(m,n) \in E} N(m,n)$$

where E is the set of all edges joining neighboring pixels.

2.5 User touchup

The user can correct the segmentation by painting with either a TrimapForeground or a TrimapBackground brush. These brushes change the trimap and thus affect the T-links. After each stroke, we update the trimap and recompute the minimum Graph Cut (step 6).

If the user desires, we can also restart the iterative learning process. This may be necessary if the initial color GMMs are very poor. Note that when we restart the iterative process we must initialize again (Section 2.2). However, now the Trimap may contain TrimapForeground in addition to TrimapUnknown and TrimapBackground. This is easy to handle. We simply initialize the Matte to MatteBackground in the TrimapBackground region. Everywhere else is set to MatteForeground.

3 Our implementation

Here we describe the implementation details of the more difficult portions of the algorithm. Our implementation has not been optimized and may not be particularly good. However, this section should be a good starting point for other implementors.

3.1 Use of the Graph Cut code

Graph Cut is a complicated algorithm. Implementing it can be quite time consuming. Instead, we make use of the publicly available implementation by Vladimir Kolmogorov. It is available from http://www.cs.cornell.edu/people/vnk/software.html. This software is very easy to use and compiles without problems in Visual C++. The documentation is sufficient so we do not discuss more details here.

3.2 Color clustering details

The GMM initialization depends upon a scheme to cluster pixels based upon their color. We use a binary tree quantization algorithm described by Orchard and Bouman [1991]. The algorithm starts with all the pixels in a single cluster. The cluster is then split in two using a function of eigenvector of the covariance matrix as the split point. It then uses the eigenvalues of the covariance matrices to choose which of the resulting clusters to split next. This procedure is repeated until the desired number of clusters is achieved. It is an optimal solution for large clusters with Gaussian distributions.

For example, we do the following to initialize K clusters for the foreground GMM:

1. Initialize the set $C_1 =$ TrimapUnknown
2. Calculate μ_1, the mean of C_1 and Σ_1, the covariance matrix of C_1
3. For $i = 2$ to K do
    - Find the set C_n which has the largest eigenvalue and store the associated eigenvector e_n
    - Split C_n into two sets, $C_i = \{x \; \varepsilon \; C_n : e_n^\top z_n \leq e_n^\top \mu_n\}$ and $C_n = C_n - C_i$.
    - Compute μ_n, Σ_n, μ_i, and Σ_i

This results in K pixel clusters. The GrabCut paper suggests using $K = 5$ for both the foreground and background GMMs.

The eigenvalues and eignevectors can be easily calculated using the OpenCV library. The library can be downloaded from http://www.intel.com/research/mrl/research/opencv/.

3.3 Computation of the Gaussian components

For each of the Gaussian components we have to compute the mean, the inverse of the covariance matrix, the determinant of the covariance matrix, and the weighting value π.

The mean and the covariance matrix are easy to compute from the pixel colors. The determinant can easily be computed from the covariance matrix. The inverse of the covariance matrix can be computed by finding the cofactor matrix and dividing it by the determinant.

The weight, π, is simply the fraction of foreground (or background) pixels that were assigned to this Gaussian component by step 4.

4 Results

Our implementation achieves very similar results to those shown in the original paper. Figure 1 shows the results of the iterative segmentation algorithm on two natural images. The flower image is successfully segmented due to the large color difference between the foreground and the background. The fish image is more difficult since the fish and the background coral share many of the same colors. The initial segmentation is nearly correct, but part of the fish is mistakenly excluded and some coral is included. This can be easily fixed by the user in the touchup step (Section 2.5).

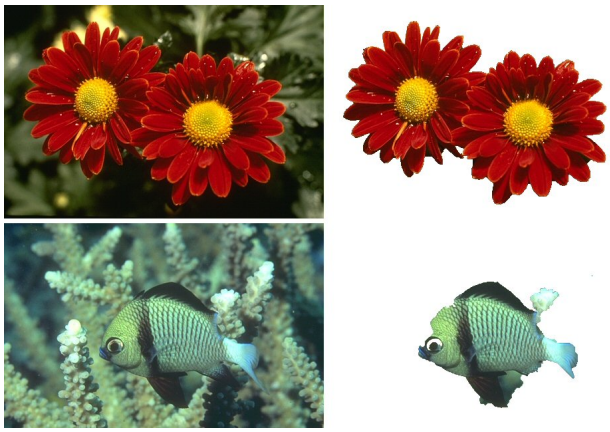

Figure 1: Results without user touchup

Figure 2 shows an image where significant user touchup is required to achieve a good segmentation. The color of the soldier's helmet nearly matches the rocks behind him. Manual editing is required to force a good edge between the helmet and the rock. The boots are not included in the initial segmentation. This problem could be fixed by using more components in the Gaussian mixtures.

The final example uses GrabCut to extract the text and illustrations from a degraded document. Figure 3 shows the poor results from the iterative learning step. Further user touchup is not helpful in eliminating the background. This may be due to a weakness in the GrabCut algorithm.

Figure 2: Middle image is the result of GrabCut, right image is after user touchup

5 Conclusion

GrabCut is a very user-friendly image segmentation tool. For images where the foreground and background are cleanly separated, GrabCut can robustly segment the image given only a rectangular region as input. However, for for some types of images, GrabCut can fail completely. The original paper contained a couple of significant errors that have been corrected here.

6 Future Work

The border matting discussed in the paper needs to be implemented and compared to other matting techniques, such as Bayesian or Poisson Matting. Also, it uses a dynamic programming approach to finding a good matte edge. This requires additional implementation overhead. GrabCut would be greatly simplified if the border matting step could be computed in the same Graph Cut-oriented data structure as the rest of the algorithm.

Although GrabCut is much faster than previous methods, the delay due to the Graph Cut step is still noticeable. Methods for incrementally updating the graph to reduce delay could be explored.

GrabCut could be extended to general N-D images. Perhaps, most interesting would be movies or medical data. Since GrabCut only requires a rough initial Trimap, user interaction costs would be significantly decreased over other N-D segmentation algorithms.

The number of components for the Gaussian mixture models was arbitrarily chosen. A better approach would determine the number of components based upon the color complexity of the image.

Finally, the minimization algorithm presented in the GrabCut paper only guarantees convergence to a local minimum. A nice theoretical result would be the formulation of a GrabCut variant that can guarantee convergence to the global minimum.

Figure 3: Example of poor results

References

BLAKE, A., ROTHER, C., BROWN, M., PEREZ, P., AND TORR, P. 2004. Interactive image segmentation using an adaptive gmmrf model. In *Proc. European Conf. Computer Vision*.

BOYKOV, Y., AND JOLLY, M.-P. 2001. Interactive graph cuts for optimal boundary and region segmentation of objects in n-d images. In *ICCV*, 105–112.

CHUANG, Y.-Y., CURLESS, B., SALESIN, D. H., AND SZELISKI, R. 2001. A bayesian approach to digital matting. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition*, IEEE Computer Society, vol. 2, 264–271.

MORTENSEN, E. N., AND BARRETT, W. A. 1999. Toboggan-based intelligent scissors with a four-parameter edge model. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition*, 2452–2458.

ORCHARD, M. T., AND BOUMAN, C. A. 1991. Color Quantization of Images. *IEEE Transactions on Signal Processing 39*, 12, 2677–2690.

ROTHER, C., KOLMOGOROV, V., AND BLAKE, A. 2004. Grabcut - interactive foreground extraction using iterated graph cuts. *Proc. ACM Siggraph*.

RUZON, M., AND TOMASI, C. 2000. Alpha estimation in natural images. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition*, 597–604.