Implementation and Improvements of Interactive Image Segmentation Techniques

Akshay Uttamani
University of
Wisconsin-Madison
auttamani@cs.wisc.edu

Ankit Aggarwal
University of
Wisconsin-Madison
ankit@cs.wisc.edu

Vaibhav D Patel
University of
Wisconsin-Madison
vpatel26@wisc.edu

ABSTRACT

The problem of efficient, interactive foreground/background segmentation in still images is of great practical importance in image editing. In this paper, we re-implement the two famous interactive image segmentation methods - GrabCut [1] and Lazy Snapping [3]. GrabCut is an approach based on optimization by graph-cut which combines both types of information i.e. color and contrast. It is an iterative version of the optimization that is used to simplify substantially the user interaction needed for a given quality. Lazy Snapping, is another interactive image cutout tool that separates coarse and fine scale processing, making object specification and detailed adjustment easy. Moreover, it provides instant visual feedback, snapping the cutout contour to the true object boundary efficiently despite the presence of ambiguous or low contrast edges. Also, some improvements are suggested to these algorithms and results are compared and explained at the end.

INTRODUCTION

Segmentation is generally the first stage in any attempt to analyse or interpret an image automatically. Segmentation bridges the gap between low-level image processing and high-level image processing. Some kinds of segmentation techniques are found in any application involving the detection, recognition, and measurement of objects in images. The role of segmentation is crucial in most tasks requiring image analysis. The success or failure of the task is often a direct consequence of the success or failure of segmentation.

Applications of Segmentation

- Optical character recognition (OCR)
- Tracking of objects in a sequence of images
- Classification of terrains visible in satellite images
- Detection and measurement of bone, tissue, etc. in medical images

WHY THIS PROBLEM?

The previous image segmentation tools used either texture (color) information or edge (contrast) information. Here we address the problem of efficient, interactive extraction of a foreground object in a complex environment whose background can not be trivially subtracted. The resulting foreground object reflects the proportion of foreground and background. The aim is to achieve high performance at the cost of only modest interactive effort on the part of the user. High performance in this task includes: accurate segmentation of object from background; clean foreground color, free of color bleeding from the source background. In general, degrees of interactive effort range from editing individual pixels, at the labour-intensive extreme, to merely touching foreground and/or background in a few locations.

RELATED WORK

After reviewing the literature we found that there are two main methods that improve on standard pixel-level selection tools: boundary-based and region based. Each of these methods takes features of the image that the computer can detect and uses them to help automate or guide the foreground specification process. There were two main disadvantages in using them individually:

- They demanded a large amount of attention from the user i.e. the user must control the curve carefully for accurate results.
- There are often cases where the features used by the region detection algorithms do not match up with the desired foreground or background. For eg, areas in shadow, low-contrast edges, and other ambiguous areas can be extremely tedious to hint.

Some of the modern approaches to image segmentation include Fast Marching Method which is a numerical method for solving boundary value problems; other graphical partitioning methods like Normalized Cuts, Min Cuts which model the impact of neighbourhood pixels on a given cluster of pixels. Trainable segmentation using Neural Networks have also been used recently for image segmentation which relies on processing small areas of an image using an artificial neural networks.

OVERVIEW

The first approach, Lazy Snapping consists of two steps: a quick object marking step and a simple boundary editing step. The first step, object marking, works at a coarse scale, which

specifies the object of interest by a few marking lines. The second step, boundary editing, works at a finer scale or on the zoomed-in image, which allows the user to edit the object boundary by simply clicking and dragging polygon vertices. This system inherits the advantages of region-based and boundary-based methods in two steps. The first step is intuitive and quick for object context specification, while the second step is easy and efficient for accurate boundary control. Furthermore, at the object marking step, an efficient graph cut algorithm can be implemented by employing pre-computed over-segmentation so that the marking UI can provide instant visual feedback to the user. At the boundary editing step, a simple polygon editing UI is introduced, and the polygon locations are used as soft constraints to improve snapping results around ambiguous or low contrast edges.

The second approach is segmentation using GrabCut. The image segmentation using GrabCut can be done by first dividing the image into foreground and background classes by selecting rectangle around the object of interest and then modeling it as Gaussian Mixture Models (GMMs) using Orchard-Bouman clustering algorithm, and lastly applying graph-cut on the built graph.

TECHNICAL APPROACH

Object Segmentation using Graph-Cut

In order to explain how the GrabCut and Lazy Snapping algorithm works, we need to briefly describe the graph cuts energy minimization. The graph cuts interactive algorithm addresses the segmentation given an initial trimap T that consists of:

- T_B (Set of Background pixels)
- T_F (Set of Foreground pixels)
- T_U (Unlabeled pixels)

Figure 1. an example of a trimap overlaid on a photo, where background pixels are in blue, foreground pixels in red and unlabelled pixels in green

Briefly there are three steps in graph-cut:

- The image is defined as an array of grey values $z=(z_1,...,z_n,...,z_N)$ and the segmentation of the image is expressed as an array $\alpha=(\alpha_1,...,\alpha_n,...,\alpha_N)$, which gives to every pixel a value in a range from 0 to 1 (where 0 means background and 1 means foreground)
- The histograms are built from labelled pixels from the respective trimap regions T_B, T_F and are normalized to sum to 1 over the grey level range.
- The segmentation is estimated as a global minimum of an energy function and is computed by a standard minimum cut algorithm.

Lazy Snapping

Lazy Snapping is an interactive image segmentation technique which separates coarse and fine scale processing, making object specification and detailed adjustments easy. Moreover, Lazy Snapping provides instant visual feedback, snapping the cutout contour to the true object boundary efficiently despite the presence of ambiguous or low contrast edges. Instant feedback is made possible by a image segmentation algorithm which combines graph cut with pre-computed over-segmentation. In this project, we implemented this algorithm and made some modifications to improve its efficiency.

Details of the two steps of Lazy snapping:

Step 1 - Object Marking: The previous approaches required the user to trace the object boundary which incurred human errors and high user effort. Our system allows users to use lines and curves to specify the extent of the object of interest. To specify an object, a user marks few lines on the image by dragging the mouse cursor while holding a button. We used two different colors to display the foreground marker or background marker. The segmentation process is triggered once the user releases the mouse button after each marking line is drawn. The user inspects the segmentation result and decides if he wants to mark more lines or not. To keep this interaction smooth we tried to keep the response time of our tool to be as low as possible.

One of the ways of making it faster is to use the concept of Graph Cut Image Segmentation. An image cutout problem can be posed as a binary labelling problem. Suppose that the image is a graph G(V,E) , where V is the set of all nodes and E is the set of all arcs connecting adjacent nodes. Usually, the nodes are pixels on the image and the arcs are adjacency relationships with four or eight connections between neighboring pixels. The labelling problem is to assign a unique label x_i for each node i V, i.e. x_i foreground(= 1), background(=0). The solution X = x_i can be obtained by minimizing a Gibbs energy:

$$E(X) = \sum_{i \in V} E_1(x_i) + \lambda \sum_{i,j \in E} E_2(x_i, x_j)$$

where $E_1(x_i)$ is the likelihood energy, encoding the cost when the label of node i is x_i, and $E_2(x_i, x_j)$ is the prior energy, denoting the cost when the labels of adjacent nodes i and j are x_i and x_j respectively. Once the user marks the image, two sets of pixels intersecting with the foreground and background markers are defined as foreground seeds F and background seeds B respectively. To compute E_1, first the colors in seeds F and B are clustered by the K-means method. E_2 to is used to represent the energy due to the gradient along the object boundary(calculated mathematically using L2-Norm of the RGB color difference of two pixels). To minimize the

energy E(X) in the main equation, we use the max-flow algorithm.

Step 2 - Boundary Editing: The object marking step preserves the object boundary as accurately as possible but there still exist some errors. To overcome those, we use a polygon editing UI for the user to refine the object boundary. The rough object boundary produced in the previous step is converted into an editable polygons. The polygon is constructed in an iterative way: the initial polygon has only one vertex, which is the point with the highest curvature on the boundary. At each step, the distance of each point on the boundary to the polygon in the previous step is computed. The farthest point is inserted to generate a new polygon. The iteration stops when the largest distance is less than a pre-defined threshold. Once the user releases the mouse button after each polygon editing operation, the system will optimize the object boundary using the graph cut segmentation algorithm again. The optimized boundary automatically snaps to the object boundary even though the polygon vertices may not be on it.

Improvements: To improve efficiency, we use a graph cut formulation which is built on a pre-computed image over-segmentation, instead of image pixels. As proposed in the paper we use watershed algorithm, which locates boundaries well, and preserves small differences inside each small region. Watershed segmentation provides a good super set of object boundaries and hence this approximation produces reasonable results and improves the speed significantly. On top of that we tried using the concept of Gaussian Mixture Model (GMM). We observed that using Gaussian model to fit every cluster improved the time complexity of our algorithm and produced more accurate results. Most importantly, our algorithm is able to feedback the cut out results almost instantly.

GrabCut

The proposed approach includes two main steps: 1) Hard Segmentation using iterative graph cut 2) Border Matting in which alpha values are computed in a narrow strip around the hard segmentation boundary

Step 1 - Image Segmentation: This step addresses the segmentation of an image, given an initial trimap T. The image is an array z = (z_1,..., z_n,..., z_N) of grey values, indexed by the (single) index n. The segmentation of the image is expressed as an array of opacity values α= (α_1,..., α_N) at each pixel. Generally 0 <α_n <1, but for hard segmentation α_n = {0,1}, with 0 for background and 1 for foreground. Also, Gaussian Mixture Model is used to model the pixel color probability distribution function. An energy function E is defined so that its minimum should correspond to a good segmentation, in the sense that it is guided both by the observed foreground and background grey-level histograms and that the opacity is coherent, reflecting a tendency to solidity of objects. This is captured by a Gibbs energy of the form
$E(\alpha, \theta, z) = U(\alpha, \theta, z) + V(\alpha, z)$

The data term U evaluates the fit of the opacity distribution α to the data z, given the histogram model θ , and is defined to be:

$U(\alpha, \theta, z)$= $\sum_n -logh(z_n;\alpha_n)$

The smoothness term can be written as

$V(\alpha, z)$=$\gamma \sum_{m,n \in C} dis(m,n)^{-1}[\alpha_n$!= $\alpha_m]$ exp -$\beta(z_m z_n)^2$

where [ϕ] denotes the indicator function taking values 0,1 for a predicate ϕ , C is the set of pairs of neighboring pixels, and where dis() is the Euclidean distance of neighbouring pixels. This energy encourages coherence in regions of similar grey-level.

Step 2(a) - Border Matting: Border matting begins with a closed contour C, obtained by fitting a polyline to the segmentation boundary from the iterative hard segmentation of the previous section. A new trimap T_B,T_U,T_F is computed, in which T_U is the set of pixels in a ribbon of width w pixels either side of C (we use w = 6). The goal is to compute the map αn, n belongs to T_U, and in order to do this robustly, a strong model is assumed for the shape of the α -profile within T_U with two important additions: regularisation to enhance the quality of the estimated α -map; and a dynamic programming (DP) algorithm for estimating α throughout T_U.

Step 2(b) - Foreground estimation: The aim here is to estimate foreground pixel colors without colors bleeding in from the background of the source image. Such bleeding can occur with Bayes matting because of the probabilistic algorithm used which aims to strip the background component from mixed pixels but cannot do so precisely. The residue of the stripping process can show up as color bleeding. Here we avoid this by stealing pixels from the foreground T_F itself. First the Bayes matte algorithm is applied to obtain an estimate of foreground color f_n on a pixel n belongs to T_U. Then, from the neighbourhood F_t(n), the pixel color that is most similar to f_n is stolen to form the foreground color f_n.

SHORTCOMINGS OF LAZY SNAPPING AND GRABCUT

GrabCut and LazySnapping apply different clustering algorithms to distinguish the color histograms of the user-selected foreground and background. Earlier tools aimed at grey scale images, while these two focus on colored images which has wider application in real life. Thus they have a lot of advantages over any of the existing tools. To list a few:

- They are much easier to learn than any other related tools(Photo-shop etc).
- Since both of them use optimization techniques, they provide better quality cutouts in less time.
- Because of the instant feedback mechanism, they provide a better user experience.

Although there are lots of advantages of using these techniques, they still have certain shortcomings. After experimenting with different types of images we observed that they both require subtle user refinements for most cases, unless images to be handled have significant appearance difference between the foreground and background. This was also pointed out by Kolmogorov and Minka [5] in their paper. Some other shortcomings include:

- They may introduce unacceptable results in the cases of low contrast between foreground and background colors, since the algorithms heavily rely on color intensities.
- Every pixel that is marked by users is considered to be the ground truth for segmentation. In many scenarios, it is inconvenient for users to purify their inputs.
- Both the algorithms are sensitive to the color discontinuity which may lead to many noisy discontinuity and noise holes in the result.
- They are not very robust to inaccurate inputs.

IMPROVEMENTS TO GRABCUT

The original GrabCut algorithm uses a fixed number of Gaussians in each GMM, however it is observed by the results that the number of Gaussians used to model the foreground and background can have a significant effect on segmentation performance. This is because, for a larger value of the Gaussian components, over fitting will occur while modelling a color distribution and for a smaller value, we may not be able to model the entire color distribution. The modified GrabCut algorithm analyses the foreground and background regions prior to segmentation and estimates the optimal number of Gaussians needed in each GMM in order to best model each region. In order to predict the optimal number of Gaussians in GMM model, we use minimum description length minimization and choose K which minimizes minimum description length optimization function for each iteration of the original GrabCut algorithm. This improvement removes the variations that may be induced on the segmentation results due to the arbitrary setting of the number of components by the user.

GrabCut also utilizes edge information to identify the border between foreground objects and background. This is done by analysing the gradient (change in color) between two neighboring pixels. Altering the algorithm to construct the background GMM using only background pixels in the bounding box in subsequent iterations gives a more accurate model of background colors around the selected object. This effectively focuses the method on the region of interest by removing isolated background colors from the GMM that are not spatially close to the foreground object. In the modified algorithm, after the first iteration, the background is constrained to only be the pixels in the bounding box marked by the user, where as the original algorithm considers the entire image. This provides a more localised modelling of the background colour distribution, theoretically providing a better segmentation result. This also reduces the total number of pixels which are considered in the iterative energy segmentation and produces improvements in the speed of the energy optimisation step of the GrabCut algorithm.

With the original GrabCut algorithm, we observed that it sometimes fails to detect holes present in the foreground object and marks them as part of the foreground itself. However, with these improvements , we were able to address this issue and produce good segmentation results as displayed in images below. While the first improvement suggested above is a generic improvement to the GrabCut algorithm, the second one seems to specifically address the hole detection issue. This is because the color of the hole pixels is more similar to the background pixels inside the bounding box rather than the ones outside the bounding box thus modelling the background color distribution better to catch the holes.

The results of these modifications are compared in Experiments section including both ground truth data and bounding box locations, allowing for consistent benchmarking and quantitative performance evaluation.

A NEW SEGMENTATION APPROACH

Both lazy snapping and GrabCut model "color" for segmentation. Experimentally, we have seen that using a single image property like color may not always produce the best results. Images which have very small difference in the color distribution of foreground and background are very hard to segment using these algorithms. To overcome this shortcoming, we tried to use a different image property along with an optimized segmentation technique. Our method is based on appearance propagation images which was proposed by Pellacini [4] and is considered to be an excellent choice in modeling users inputs.

Algorithm :

- As in the previous two approaches, users are prompted to use different colors to paint part of the foreground and part of the background accordingly. It is not necessary for users to ensure the precision of their inputs because the system could identify and tolerate cases in which users accidentally touch the background with foreground strokes. This is made possible by using the Foreground cropping, where we use a threshold to determine if the pixel marked by the user as foreground was correct or not.
- Once we have the input, we apply the normalized appearance propagation to both the users foreground and background labellings independently. We label the region as "1" marked by the user as foreground and "0" for the region marked as background. This step propagates the foreground region to the whole image by brightening pixels that have similar textures to those in the "marked 1" area subject to geometric constraints. A similar approach is used for background propagation.
- Once we have labelled the pixels, we apply the standard Graph cut segmentation approach to separate the foreground from the background.

This algorithm can further be optimized by using a Normalization or an optimization step before the appearance propagation step. We intend to do this in our future work.

Results

We show in Fig.2 and Fig.3 the difference of results generated from GrabCut and from our approach. As previously mentioned, we believe that these images are usually considered hard for interactive segmentation due to similar color tones in their foregrounds and backgrounds. Even if the user attempts to provide accurate inputs it would be hard for GrabCut to generate correct result, since it relies only on color. With our approach, even with a small input from user, he might cover

(a) (b)

Figure 2. (a)Original image, (b)user inputs

(c) (d)

Figure 3. (c)output from our approach and (d)output from GrabCut

more textures and structure information during the interaction and hence the result generated is more accurate.

EXPERIMENTS AND EVALUATION

We ran our implementations on a set of images from the Berkley Image Segmentation data-set. We selected a set of images from the data-set and divided them into three sets namely Easy, Medium and Tough images in terms of number of foreground objects, color similarity in foreground and background regions, different colors present in the image(as GrabCut and Lazy Snapping rely only on the color property of the image). Then, we manually marked the foreground object boundaries in all the test images. The data-set also contains human marked boundaries for all the images, but they address general image segmentation and not the foreground-background object segmentation.

Lazy Snapping:

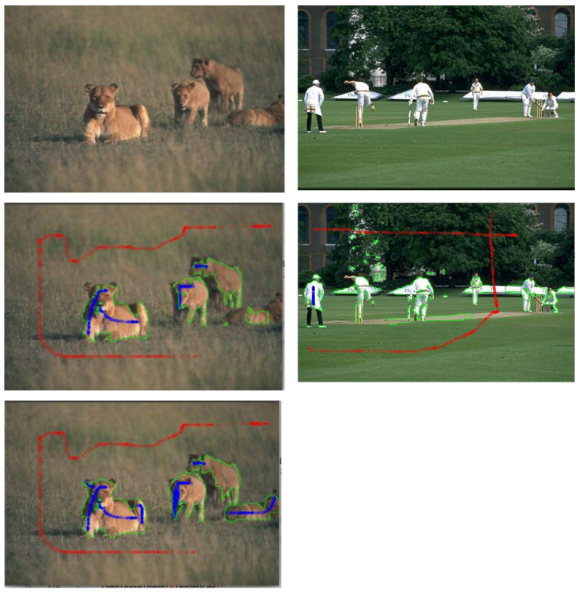
Figure 5 shows the results of the implementation of Lazy Snapping method. For each example set, the first row represents the original images, the remaining rows represent the results with user inputs. In the images, the red markings show the user input for background points and blue markings for the foreground points. After each input given by the user, the system recalculates the boundary points and marks them with green.

From figure 5, we can see that our implementation works best for easy images such as horse.jpg and relatively bad for tough images such as cricket.jpg, while not that bad for eskimo.jpg type medium images. This is due to the fact that the difference between the foreground and background color separation for horse.jpg is relatively larger than the other two sets of images.

In the lions.jpg image, the results of the second row are much better than the first row, since the second row has more strokes for both the foreground and background as compared to the first row. As the user marks more and more foreground and

(a) (b)

Figure 4. Examples showing the segmentation result on running Lazy Snapping algorithm (a) horse.jpg (b) eskimo.jpg



(c) (d)

Figure 5. Examples showing the segmentation result on running Lazy Snapping algorithm (c) lions.jpg (d) cricket.jpg

background pixels, the model gets more information about the colors in both regions, which leads to a more accurate Gaussian mixture model to approximate the probability of the image pixels belonging to foreground and background and thus results in better segmentation.

In the second row of the eskimo.jpg image, dogs are mistakenly segmented as the background, because of its great similarity in color with the color of the ice and sky which are marked as background by the user. It can also be observed in other images (second row in horse image) that foreground

regions that are similar to the background in color are quite possibly classified as background.

The performance on cricket.jpg image is worse, because of the great similarity between much of the background regions and the foreground. Even marking many pixels in foreground and background doesn't help much.

(a) (b)

Figure 6. The segmentation result using Lazy Snapping approach on an Image having color discontinuity

Figure 6 shows that the Lazy Snapping creates noisy holes in the image when there is some color discontinuity in the image.

In conclusion, what really matters in the lazy snapping is the size of the samples and the distinction of foreground from background in color. In order to get good results, the user must make sure that the strokes are rich enough, inclusive of different color patches in foreground and background. For those images with low contrast between foreground and background, more strokes are required to get good segmented image.

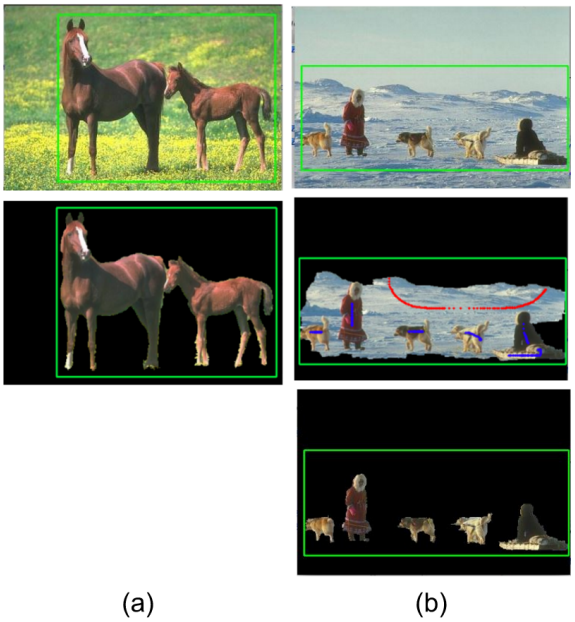
GrabCut:

The figure 8 shows the results of the implementation of GrabCut. In the example set, the first row represents the images with marked area of interest by the user, second row onwards are the segmented results with user inputs. In the images, the red markings show the user input for background points and blue markings for the foreground points. The user is given a feature to ask the system to segment the image after entering inputs.

On applying the algorithm to the easy set of images (horse.jpg), we can see that just after making the rectangle around our area of interest, the system is able to segment out the foreground object efficiently. While on the other side, for correctly segmenting medium images (eskimo.jpg, cricket.jpg), more user input is required by the system other than the rectangle.

The third and fourth column of images in figure 8 show the effect of varying user input on the segmentation results. In the third column, the user marks all the incorrectly segmented portion of image compared to the image on fourth column where user marks few of them. As we can see, the system segments out the foreground objects perfectly compared to the result on fourth column.

Figure 9 shows the segmentation result when applying the GrabCut algorithm on foreground objects having holes in them. Here the original GrabCut algorithm marks the area in between the pillars(i.e. holes) as the foreground in part b



(a) (b)

Figure 7. Example images showing the segmentation result on running GrabCut algorithm (a) horse.jpg (b) eskimo.jpg

(c) (d)

Figure 8. Example images showing the segmentation result on running GrabCut algorithm (c) cricket.jpg Image with more user input (d) cricket.jpg Image with lesser user input

which ideally should be marked as background. This is because the color of hole pixels is dark blue while the color of the background is bright blue. When the user selects the bounding box and marks the area outside of it as the background; that initial background color distribution does not model the color distribution of the hole region correctly. Instead the color of pixels close to the foreground object inside the bounding box is more similar to the color distribution of the holes. Hence, the improved GrabCut algorithm rectifies

Figure 9. Example that shows the comparison of segmentation result on running the improved GrabCut algorithm compared to original GrabCut approach (a) Original Image (b) GrabCut result (c) Improved GrabCut result

this issue by negating the effect of background pixels outside the bounding box on the background color distribution. The results of modified algorithm is displayed above in part c where the hole region is correctly segmented out as background.

To evaluate the two algorithms and compare with our approach we intentionally picked images that had complex lighting conditions. We ran all the three algorithms on almost the entire data set and found that our approach outperformed both GrabCut and Lazy Snapping.

CONCLUSION

In this project, we implemented the two most popular interactive image segmentation techniques- GrabCut and Lazy Snapping. After experimenting with various images and going through the related literature we found the advantages and the shortcomings in the two methods. We demonstrated a different interactive segmentation system that uses texture and appearance propagation as its central feature. After running our experiment on various types of images, we observed that our approach is capable of handling a wider range of images relying only on coarse user strokes. This we believe is because of the fact that most of the real life images do not have a clear difference of color distribution in their foreground and background. The main advantages of this approach is its robustness to user error, the embedded appearance propagation process and better segmentation results on intricate images. Also, some improvements were made specifically to GrabCutś iterative algorithm through which we were able to produce better results when the foreground object in the image has holes.

FUTURE WORK

Through our experiments we observed that relying on a single image property like color or texture may not be enough for the desired segmentation. Thus as our future work we intend to extend our approach of using appearance propagation modeled on a combination of color and texture (or using a combination of various image properties). We believe that adding an optimization step before the appearance propagation step can speed up the process. In this project we focused more on improving accuracy than on improving run-time complexity. As a part of future we also intend to reduce the run-time complexity by experimenting with various optimization techniques.

REFERENCES

[1] C. Rother, V. Kolmogorov, A. Blake. GrabCut: Interactive Foreground Extraction using Iterated Graph Cuts. In ACM Transactions on Graphics (SIGGRAPH), 2004

[2] 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

[3] Y Li, J Sun, CK Tang, HY Shum - Lazy Snapping. In ACM Transactions on Graphics (ToG), 2004

[4] AN X., PELLACINI F.: AppProp: all-pairs appearance-space edit propagation. In International Conference on Computer Graphics and Interactive Techniques (2008), ACM New York, NY, USA.

[5] ROTHER C., KOLMOGOROV V., MINKA T., BLAKE A.: Cosegmentation of Image Pairs by Histogram Matching-Incorporating a Global Constraint into MRFs.In Proc. CVPR (2004)

PROJECT WEBSITE

http://pages.cs.wisc.edu/ auttamani/