Fast Image Segmentation on Mobile Phone using Multi-level Graph Cut

Category: Research

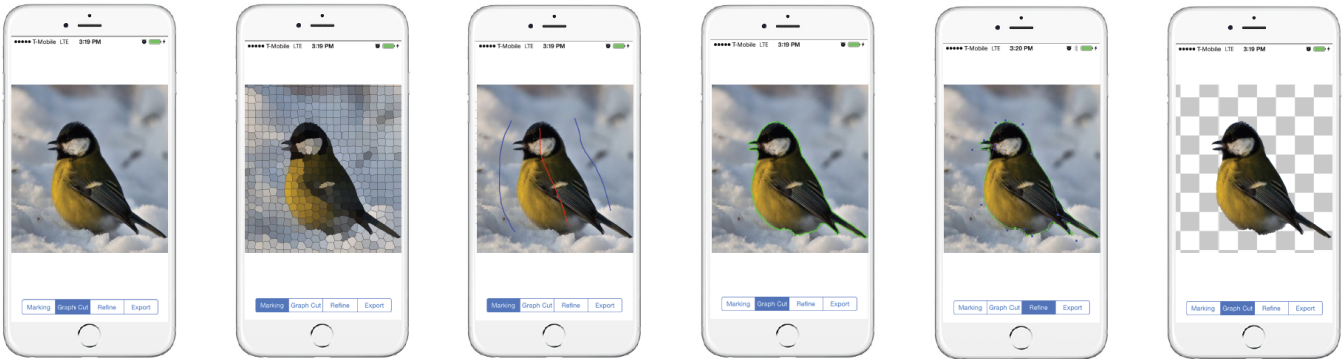


Figure 1: Our method is an interactive foreground segmentation system consisting of four steps: A pre-segmentation step, and object marking step, an initial Graph Cut step, and a boundary refinement step. In (a), our target image. In (b), the Superpixels algorithm is performed on a downsampled version of our input image. In (c), the user marks the foreground and background of the image. In (d), a graph cut is performed on the pre-segmented image using marks made by the user. In (e), the resulting foreground segmentation is upsampled to the original input image's resolution and the boundary is approximated by editable Bezier curves. A local graph cut is performed on a per-pixel basis in a dilated regions around each curve. In (f), the final segmentation.

Abstract

This paper presents a system for an efficient image segmentation on mobile phones using multi-level graph cut. As the computational capacity of mobile devices is often limited, a fluent and smooth image segmentation is a challenging task with existing segmentation algorithms, increased in difficulty by mobile phone cameras continually upgraded to take photos of higher resolution. Our solution is to carefully tweak the classic graph cut algorithm for an interactive image cutout, enhancing the performance without compromising the quality of the segmentation. This is achieved by downsampling the original high-resolution image and selecting a rough cutout region on this low-resolution image with a superpixel based pre-segmentation. The segmented foreground is then mapped back to the full-size image and the image undergoes an adaptive boundary refinement. This second segmentation performs the optimization locally and can be accomplished within milliseconds. We test our system on an Apple iPhone 6 and our experiments show that a high quality segmentation can be achieved in a lag-free manner on the mobile phone even for multi-megapixel images.

Index Terms: I.3.3 [Computer Graphics]: Methodology and Techniques—Interaction Techniques; I.4.6 [Image Processing and Computer Vision]: Segmentation—Pixel Classification;Partitioning

1 Introduction

Image segmentation is a pixel labeling technique that categorizes image pixels into two regions: a foreground and a background. The foreground typically, is associated with objects of interest (e.g, the portrait of a person) similar to using Magnetic Lasso in Adobe Photoshop®. A high quality segmentation is desired in many problems in computer vision, image processing as well as computer graphics such as face recognition, image synthesis [15] and so on. The task of manually specifying the foreground and background of an image, is a tedious process that becomes all the more laborious as image resolutions increase and constraints are placed on user interaction such as performing the process on a mobile device.

The rapid ubiquity of mobile hardware has led the portable camera phone to become the dominate approach for image acquisition and sharing. The state-of-the-art cameras equipped on mobile devices are able to take pictures at multi-megapixel resolutions. However, existing image segmentation algorithms/interfaces are mostly designed for desktop PCs with the assumption of a high-frequency CPU, sufficient on-board memory, and precise user inputs (i.e., using mouse). Naively porting these algorithms [5, 18, 25] to a mobile phone does not yield a responsive and usable experience.

Our challenges are twofold. First, although we have witnessed a significant performance boost in embedded hardware recently, solving the complex optimization problem associated with the segmentation of an HD image is still challenging for mobile devices taking dozens of seconds or even minutes. Secondly, the touch screen of the phone is much smaller than the display of a conventional desktop PC. The input method is much less precise via touch screen and obscures the view of the screen. From a human-computer interaction standpoint, we must design an interface which allows for effective control of the segmentation.

Motivated by aforementioned challenges, we present a segmentation interface specially tuned for mobile devices. The most time-consuming computation, associated with global gibbs energy optimization, is carried out on a downsampled low-resolution snapshot image with a two-level graph cut. Then the segmented foreground is mapped back to the original image and followed by an adaptive boundary refinement. Users only need to provide a couple of strokes on the screen and our method automatically produces satisfactory segmentation even for objects with fuzzy boundaries.

2 Related Work

Image segmentation has long been considered a fundamental task in many areas. Segmentation algorithms extract some features of an image (such as color similarity) and formulate them into an optimization problem: defining a foreground region that is sufficiently different from the rest of the image while enforcing that the color pattern of the foreground is consistent.

Boundary-based selection methods such as Active Contour Mod-

els (Snakes) [14] and Intelligent Scissors [19] allow users to define the boundary of the image foreground with an energy-minimizing spline curve. Unfortunately when the boundary is complicated or the object is in a highly-textured region, the user may need to perform many iterations in order to successfully obtain a satisfactory segmentation. The boundary-based approach is difficult to implement on mobile devices because unique user interface problems arise due to the precision required to specify detailed boundary regions.

Adaptive painting techniques, such as Paint Selection [18], Intelligent Paint [22], Bilateral Grid [8], and Edge Respecting Brushes [21] use a progressive painting-based tool for local selection in images. Instead of solving the global optimization problem, adaptive painting algorithms incrementally solve a set of local optimization problems defined by the users' interaction. The adaptive painting approach updates and displays the current foreground segmentation in real-time and displays no extraneous marks on the image. The Paint Selection algorithm has also been shown to handle high-resolution (multi-megapixel) images at interactive rates.

Scribble-based selection [5, 13, 20, 25] requires less user input. The segmentation is computed based on a few foreground and background "scribbles" specified by users at the beginning as the initial constraint. Graph cut [5] is a widely-adopted method for this division, derived from max-flow min-cut with the proven effectiveness. However, this algorithm is computationally intensive when high-resolution images are to be processed, resulting in unacceptable computation times on mobile devices. Maximal similarity region merging (MSRM) [20] automatically merges superpixels that are initially generated via mean shift segmentation and then constructs the foreground object boundary by labelling all non-marked superpixels as belonging to the foreground or background. Semisupervised kernel matrix learning has been applied to interactive image segmentation via kernel propagation (KP) [13]. Utilizing kernel propagation a small seed-kernel matrix is generated from the users input which is propagated into a full-kernel matrix of the target image. Foreground segmentation occurs during the kernel propagation.

Recently there have been many proposed image segmentation methods that are explicitly targeting mobile platforms. Discriminative clustering [10] uses a simplified graph-cut-like method, which can significantly accelerate the speed of the segmentation. The core idea here is to split the gibbs energy minimization into two separate steps. With this simplification each minimization becomes a much smaller polynomial problem. BCRW algorithm [12] integrates a Bayesian classifier into the random walk optimizer. User input trains a Bayesian classifier which determinines the edge weights and input labels in the random walk optimizer for image segmentation. The object class segmentation [11] algorithm targets tablets by combining a superpixel pre-segmentation with GrabCut to generate an initial solution and then allows the user to refine the selection by manually marking background superpixels. Liu et al. [**?**] merge over-segmented image regions according to the maximal similarity rule and then refine ambiguous boundary regions automatically using local and global information. The superpixel grouping method [3] begins with a superpixel over-segmentation and after the user marks the foreground and background the algorithm iteratively merges superpixels based on their color histograms. Unlike other methods that specifically target the mobile platform, our method aims to achieves interactive image segmentations on multi-megapixel images which are now commonplace on todays mobile devices. Unlike other methods that specifically target the mobile platform, our method achieves interactive rates while producing high quality segmentation results on multi-megapixel images. Our method, with its secondary refinement step also requires a minimum of user-input to achieve pleasing results; something that important when targeting the mobile platform.

Table 1: Comparative image segmentation benchmarks (in seconds): *GC*: graph cut; *LS*: lazy snapping; *GbC*: grabcut

Image	Resolution	GC	Ours	LS	GbC
Bill.jpg	(1200x1600)	10.81	0.49	4.31	2.87
Qin.jpg	(1156x1600)	4.95	0.5	8.42	6.42
Grandpa.jpg	(1200x1600)	5.37	0.53	8.77	9.26
Elephant.jpg	(1067x1600)	2.92	0.44	4.81	7.21
Pravi.jpg	(1071x1600)	5.71	0.55	4.31	4.87

3 Interface Overview

Our method builds upon the classic graph cut algorithm [4] at various levels-of-detail of the image. Figure 1 outlines each individual step of our segmentation interface. First, we create a image pyramid from the target image which is a set of lowpass copies of our image where each at each level of the pyramid the image resolutions is decreased by factor of s [2]. Pre-segmentation is then performed on a reduced resolution level of our image pyramid using the SLIC Superpixels algorithm [1]. Then, the user provided scribbles to seed the foreground and then the background of the image. Based on the user's input the first graph cut is performed to produce a rough segmentation on the image which is now comprised of superpixels. The result of this two-level graph cut is then upsampled to the original image resolution.The boundary obtained through the pre-segmented graph cut is then approximated using quadratic Bezier curves. If necessary, this allows the user to adjust the current boundary in order to improve results of the final refinement step. The final boundary refinement is a secondary per-pixel graph cut in a dilated region around each bezier curve that comprises the current boundary. Neighboring dilated regions are always overlapped so that the updated boundary is continuous across dilated regions.

4 Pre-Segmentation

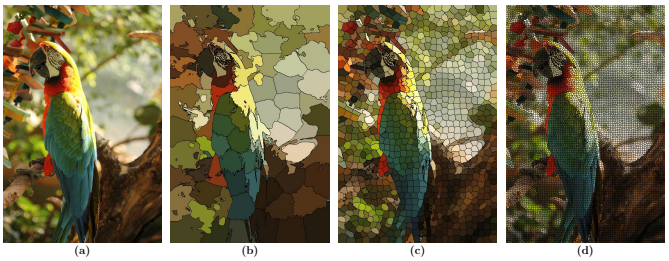


Figure 3: An image pre-segmented using SLIC superpixels with varying number of superpixels specified. (a) The original image. Image segmented into (b) 100, (c) 1000, and (d) 10,000 superpixels.

With recent advances mobile phone technology it is not uncommon for a smartphone camera to be capable of taking images of high resolution (e.g. the latest iPhone has an eight megapixel back-facing camera). In order to create an interactive foreground segmentation algorithm that can process images of this magnitude we pre-segment our target image using the SLIC Superpixels algorithm [1]. Pre-segmentation is a strategy that is adopted by many existing contributions [3, 10, 11, 17, 25].

Watershed Algorithm - In the *lazy snapping* [25] algorithm a pre-computed over-segmentation is performed which uses the watershed algorithm [16]. The watershed algorithm is relatively efficient with a complexity of $O(N \log N)$ but does not allow the user to define the number of watersheds or their density. In addition the su-

(a) (b) (c) (d)

Figure 2: A comparison of pre-segmentation methods. The resulting segmentation is colored using the average pixel color in each superpixel: In (a), we see the high frequency details that are found in the original image. In (b), the results of performing pre-segmentation using the watershed algorithm. In (c), the results of performing pre-segmentation with the Quick Shift algorithm. In (d), the results of performing pre-segmentation with the SLIC superpixels algorithm.

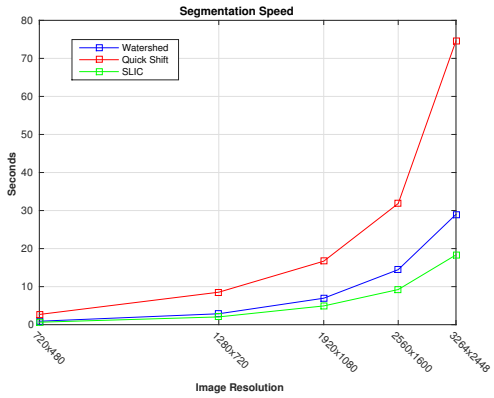


Figure 4: Time required to generate segment an image for images of increasing resolution.

perpixels generated by the watershed algorithm are highly irregular with poor boundary adherence.

Mean/Quick Shift - In the *discriminative clustering* [10] and *KP-Cut* [13] algorithms pre-segmentation is performed using the mean shift algorithm [9]. The complexity of mean shift is $O(N^2)$ and thus is not suitable for large images or implementation on a mobile device that supports large resolutions[1]. The more recent quick shift is even slower than mean shift with a complexity of $O(dN^2)$ where d is a small constant [24]. Like the mean shift algorithm, there is no explicit control over the amount or compactness of the resulting superpixels.

SLIC Superpixels - Similar to the *object class segmentation* algorithm [11] our method also relies upon an initial pre-segmentation pass of the target image using the simple linear iterative clustering (SLIC) Superpixels algorithm [1]. The SLIC Superpixels algorithm adapts a k-means clustering approach to segment an image and has multiple advantages over the other aforementioned gradient-decent-based algorithms. First, as figure 4 illustrates the SLIC Superpixels algorithm is more efficient than the other methods in our benchmark. The complexity of the SLIC Superpixels algorithm at $O(N)$ is significantly faster than the Watershed and Quick Shift algorithms with complexities of $O(N \log N)$ and $O(N^2)$, respectively. Second, as demonstrated in figure 3 the SLIC Superpixels algorithm allows the user to explicitly define both the number of and compactness of the resulting superpixels which allows for a uniform time required to calculate the graph cut solution regardless of the image dimensions. Third, the SLIC Superpixels method's most important property in regards to image segmentation is its adherance to image boundaries. As shown in figure 2 the segmentation obtained using the watershed and quick shift algorithms is highly irregular and does not preserve boundary detail as well as the SLIC Superpixels approach for the object with fuzzy boundary. Finally, the SLIC superpixel algorithm is trivial to implement in parallel which allows us to achieve an decrease in the time required for superpixel calculation in proportion to the number of cores on our mobile device.

5 Multi-level Graph Cut

As shown in figure. 6, we process the image segmentation at three different levels: the original HD image, the downsampled snapshot image and the pre-segmented downsampled image. The information of the segmentation at different level will be shared at various levels, which is similar to the famous multigrid numerical solver [7]. However, at each level we do not solve a linear/nonlienar system, instead a graph-cut based energy optimization is performed. This section details the procedures of the proposed multi-level graph cut method.

5.1 Image Pyramid

While the SLIC Superpixels algorithm is very efficient, the time required to generate a superpixel segmentation of multi-megapixel images it is unacceptable for an interactive application. For this reason, before performing the pre-segmentation step of our method we first generate an image pyramid from our target image. The image pyramid is a hierarchical data structure that supports efficient scaled convolution via reduced image representation. The data structure is a hierarchy of images where each successive level contains a copy of the original image in which the resolution is decreased at regular

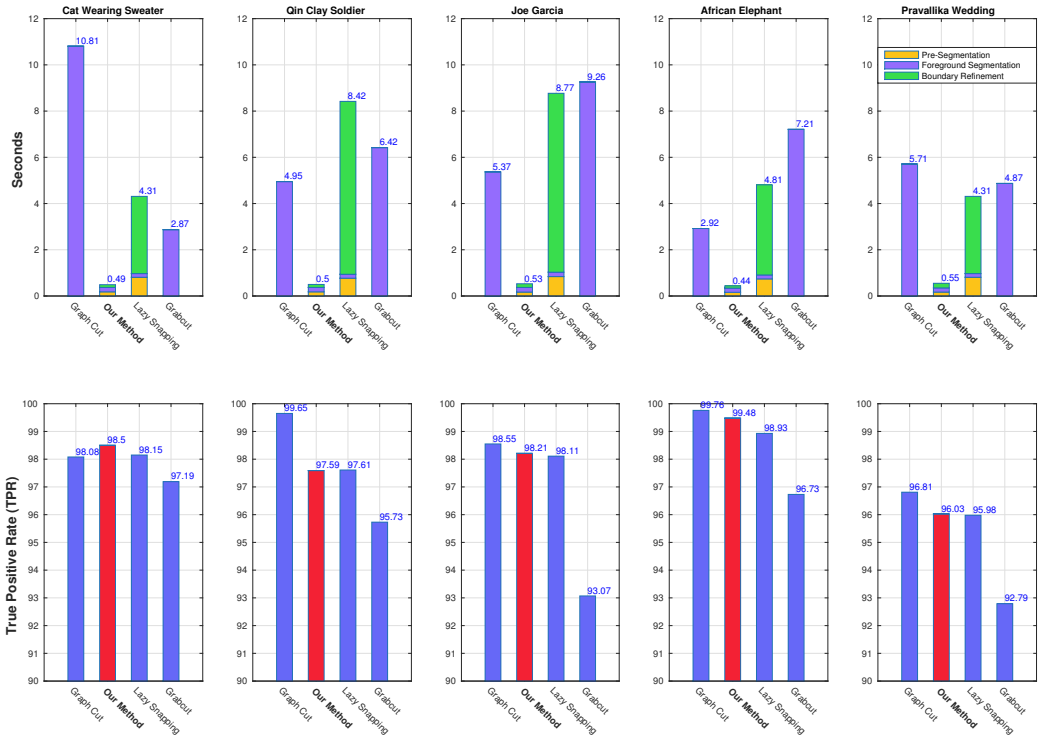


Figure 5: Performance comparison and true positive rate (TPR) of different image segmentation algorithms.

intervals. The original image comprises the bottom level our our pyramid, G_0. The original image is then low-pass-filtered and subsampled by a factor of two to obtain the next level, G_1. This process of filtering and subsampling is repeated until the desired number of pyramid levels have been generated [2]. To generate the l^{th} level $(0 < l < N)$ of our N level pyramid we have:

$$G_l(i,j) \sum_m \sum_n w(m,n) G_{l-1}(2i+m, 2j+n), \qquad (1)$$

where the weighting function $w(m,n)$ is a small separable generating kernel [2].

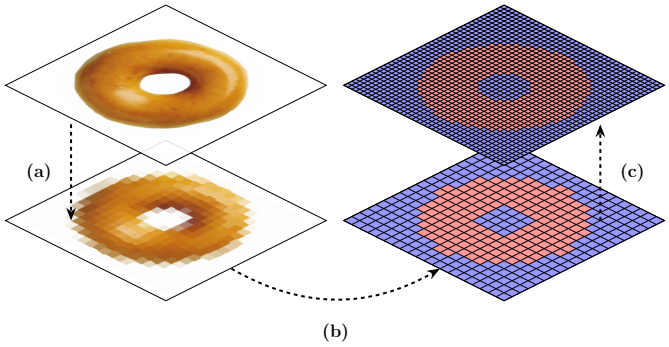


Figure 6: Multiresolution graph cut: (a) Downsample target image. (b) Graph cut is performed a reduced level of our image pyramid. (c) Map solution to original resolution.

5.2 Superpixel Graph Cut

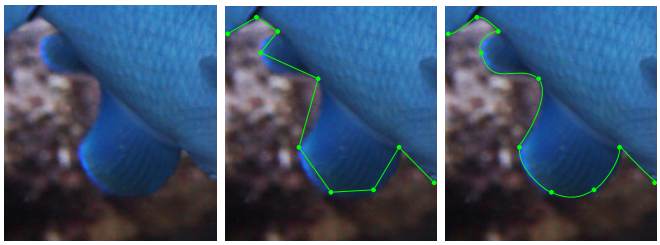


Figure 7: A comparison of boundary approximation methods: (a) Original image. (b) Boundary approximated using polygon. (b) Boundary approximated using Bezier curves.

The graph cut technique poses the foreground cutout as a binary labelling problem. The image is represented by a graph $G = \langle V, E \rangle$, where V is the set of all vertices and E is the set of all edges connecting neighboring vertices in the graph. The problem is then restated as giving a label, x_i to every vertex X_i in our graph. The solution $X_i = x_i$ is calculated by minimizing a Gibbs energy $E(X)$:

$$E(X) = \sum_{i \in V} E_1(x_i) + \lambda \sum_{(i,j) \in E} E_2(x_i, x_j), \qquad (2)$$

where $E_1(x_i)$ represents the cost of the labeling for x_i and $E_1(x_i)$ represents the cost when two vertices i and j sharing an edges are labelled x_i and x_j. $E_1(x_i)$ defines the penalty for assigning the label of vertex i as foreground or background. $E_2(x_i, x_j)$ defines the penalty for assigning adjacent vertices i and j different labels.

We use the same energy definition as in lazy snapping [25]. The downsampled image pixels are grouped into sub-regions using the SLIC Superpixels algorithm. As a result, V represents the set including all the superpixels, and E represents the set of all arcs connecting a pair of adjacent superpixels. After that, the user marks the foreground and background regions of the image by finger stroking on the touch screen as *foreground seeds*, $\mathcal{F}$ and *background seeds*, $\mathcal{B}$. Any regions not contained in $\mathcal{F}$ or $\mathcal{B}$ are considered as in the uncertain vertices $\mathcal{U}$. Once these sets are created, the average colors of the regions in $\mathcal{F}$ and $\mathcal{G}$ are clustered using the k-means method. For each vertex i, we then calculate the minimum distance from its average color to the set of foreground clusters and to background clusters. These two distances are denoted $d_i^{\mathcal{F}}$ and $d_i^{\mathcal{B}}$ for the foreground and background. $E_1(x_i)$ and $E_2(x_i)$ in Eq. 2 are then defined as:

$$\begin{cases} E_1(x_i=1) = & 0 & E_1(x_i=0) = & \infty & \forall i \in \mathcal{F} \\ E_1(x_i=1) = & \infty & E_1(x_i=0) = & 0 & \forall i \in \mathcal{B} \\ E_1(x_i=1) = & \frac{d_i^{\mathcal{F}}}{d_i^{\mathcal{F}}+d_i^{\mathcal{B}}} & E_1(x_i=0) = & \frac{d_i^{\mathcal{B}}}{d_i^{\mathcal{F}}+d_i^{\mathcal{B}}} & \forall i \in \mathcal{U} \end{cases}, \tag{3}$$

and

$$E_2(x_i,x_j) = \frac{|x_i - x_j|}{C_{ij}+1}, \tag{4}$$

respectively. Here, $C_{ij} = ||C(i) - C(j)||^2)$ is the L2 distance between vertices i and j in the RGB color space. We minimize the energy $E(X)$ using the max-flow min-cut algorithm [5].

5.3 Per-Pixel Boundary Graph Cut

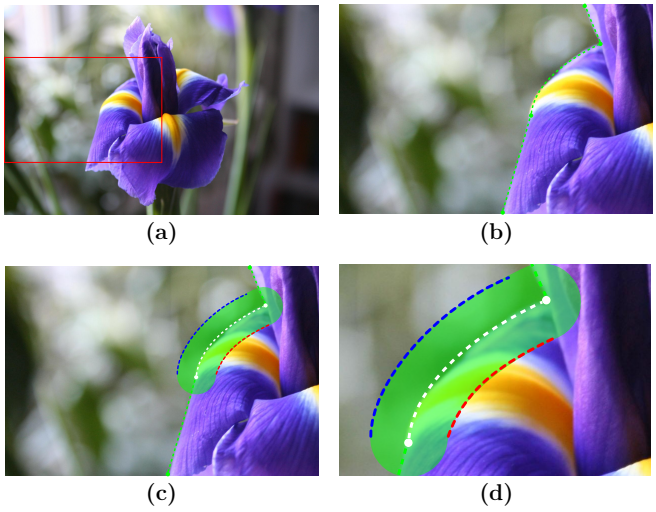


Figure 8: Localized per-pixel graph cut. In (a), the original image with our region of interest outlined. In (b), the boundary between the foreground and background is approximated by Bezier curves. In (c), a single Bezier curve is dilated to form the region U in which the local graph cut will be performed. In (d), we zoom in on a per-pixel graph cut region. The seed sets F and B (shown in red and blue) are the boundaries of U sufficiently distant from the endpoints of the Bezier curve.

Once the initial graph cut is finished, the boundary that defines the initial foreground segmentation is converted into an editable set of Bezier spline curves (figure 7). It serves a guide for a next-level per-pixel graph cut located in a small band around the curve edge. This means that V becomes the set of pixels rather than pre-segmented regions in the first cut. Obviously, increased V leads to the slower computation and the per-pixel graph cut could still be quite expensive even though we are only considering a small boundary region. To accelerate the computation, we further subdivide the boundary band into multiple overlapped sub-scripts as shown in figure 8 (c). The entire procedure is outlined algorithm 1. The advantages of this are approach are two-fold. First, it may not be necessary to perform boundary refinement over the entire boundary as in previous methods [18, 25] and we allow the user to interactively specify boundary regions that they believe are not satisfactory. Second, the per-pixel graph cut can be trivially parallelized over the sub-scripts while the size of each graph cut is significantly reduced. Therefore, our parallelized per-pixel graph cut is able to be accomplished even with a mobile CPU. We intentionally overlap neighboring sub-scripts so that the boundary refinement will not lead to any discontinuity as we linearly interpolate the boundary at the overlapped region based on the parametrization of the Bezier curves. On average, a $\times n$ performance boost can be expected with our implementation, where n is the number of CPU cores available on our mobile platform. A final foreground mask is generated using a bi-cubic interpolation over a 4×4 pixel neighborhood.

Data: Set of Bezier curves, C
Foreground mask from Superpixel Graph Cut, F_{SP}
Background mask from Superpixel Graph Cut, B_{SP}
Result: Set of labelings (foreground or background) for each region defined by the dilated Bezier curve.

```
for c ∈ C do
    Dilate c to determine set of local Graph Cut pixels P;
    Determine set of edge pixels E ⊂ P;
    F ⟵ 0;
    B ⟵ 0;
    for e ∈ E do
        if e ∈ F_SP and the distance from e to endpoints of c is
        greater than ε then
            F ⟵ F ∪ {e}
        end
        if e ∈ B_SP and the distance from e to endpoints of c is
        greater than ε then
            B ⟵ B ∪ {e}
        end
    end
    U = P − (F ∪ B);
    GraphCut(F, B, U);
end
```

Algorithm 1: Per-Pixel Boundary Graph Cut

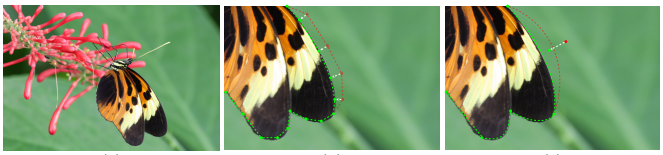
Table 2: The true positive rate (TPR) for the different methods.

Image	Graph Cut	Our Method	Lazy Snapping	Grabcut
Bill.jpg	98.08%	98.5%	98.15%	97.1972%
Qin.jpg	99.65%	97.59%	97.61%	95.73%
Grandpa.jpg	98.55%	98.21%	98.11%	93.07%
Elephant.jpg	99.76%	99.48%	98.93%	96.73%
Pravi.jpg	96.81%	96.03%	95.98%	92.79%

6 Experimental Results

Our interface was implemented on an Apple iPhone 6 using a mixture of C++ and Objective-C. The iPhone 6 is equipped with a 1.4 GHz Apple A8 (Dual Core) with 1 GB LPDDR3 RAM. The iPhone 6 has an 8 megapixel camera with capture resolution up to 3264×2448. Our desktop development was done using on an

Figure 9: A comparison of results from the different image segmentation algorithms: (a) Original image with user input strokes. (b) Ground truth. (c) Graph Cut. (d) Lazy Snapping. (e) Grabcut. (f) Our method.



(a) (b) (c)

Figure 10: A comparison of boundary representation methods. In (a), the original image. In (b), multiple vertices need to be adjusted when the boundary is approximated using a piecewise linear curve. In (c), only one control point needs to be adjusted when the boundary is approximated using a quadratic Bezier curve.

Apple Macbook Pro with a 2.93 GHz Intel Core 2 Duo and 8 GB 1067 MHz DDR3 RAM.

Quantatative Comparison

To quantitatively compare the different segmentation methods we manually created a ground truth foreground segmentation in Adobe Photoshop®. We then calculated the ratio of foreground pixels that agree with our ground truth image to the total number of pixels in each test image. This measurement is known as the true positive rate (TPR). Additionally, we measured the total segmentation time for each method accross a test suite of five images.

Comparison with Graph Cut

We first compare our method with a per-pixel implementation of Graph Cut [4]. As table 2 shows, the per-pixel Graph Cut provides the most accurate foreground segmentation results among all the methods we tested. Even though our method far surpasses Graph Cut in terms of performance we fail to achieve the same quality of foreground segmentation, even with our secondary per-pixel graph cut refinement step. The reason for the discrepancy in the results achieved with Graph Cut and our method is due to the fact that our per-pixel graph cut is localized within the regions defined by dilating our Bezier curves which define the coarse-level boundary. Therefore, the regional energy term E_1 in equation 2 contains only local information during our methods per-pixel Graph Cuts refinement step. However, while higher quality results are obtained using Graph Cut, as table 1 shows the performance of the method, with a max delay of 10.81 seconds in our tests, does not allow for a responsive experience where delays should be measured on a millisecond scale.

Comparison with Lazy Snapping

We next compare our method with lazy snapping [25] since we use the same energy formulation. We see in Figure 9 the quality of the segmentation achieved using lazy snapping and our method are quite similar. However, Table 1 reveals that lazy snapping does not maintain an interactive performance at high resolutions. This is due to the fact that lazy snapping uses a different energy formulation for the graph cut problem in the localized boundary refinement stage which incurs additional computational overhead.

Comparison with GrabCut

GrabCut [23] is another well known image segmentation algorithm. We use the implementation provided in OpenCV's Image Processing API via the *grabCut* function [6]. As Table 1 shows, the methods performance is not suitable for for mobile devices with one image from our test suite taking 9.26 seconds to process. In addition, Grabcut requires that the user specify a bounding box around the foreground subject in the image which can be problematic on the mobile phone from a usability standpoint.

7 Discussion and Limitations

We present an interface for high quality image segmentation on mobile phone at an interactive rate. This is achieved by performing graph cut, an energy optimization optimization method on a low-resolution pre-segmented version of the target image and then locally adapting/refining the boundary on full-size image. However, our current method still has many limitations. For example, the performance of a per-pixel graph cut is very sensitive to the problem size. On multi-megapixel images, large refinement regions being processed at the original resolution could result in long delays which are unsuitable for an interactive application. With the megapixel race being brought to the mobile domain by phone manufacturers this limitation will likely grow more pronounced in the future. Therefore, we may extend our method to only perform the boundary refinement step on ambiguous boundary regions instead of considering the entire boundary. With mobile-GPU compute capabilities quickly approaching that of their desktop counterpart we would also like to explore the possible parallelization of the graph cut algorithm on a mobile GPU in the future. The human-computer interaction component of developing an interactive image segmentation application cannot be ignored and we plan to release our iOS app to further refine the user experience through user studies.

8 Conclusion

We presented a system for an efficient image segmentation on mobile phones using multi-level graph cut. To meet the limitations of mobile phone hardware our solution carefully tweaks the classic graph cut algorithm. By down-sampling the original high-resolution image and selecting a rough cutout region on this low-resolution image with a superpixel based pre-segmentation we enhanced the performance without compromising the quality of the segmentation. The segmented foreground is then mapped back to the full-size image and the image undergoes an adaptive boundary refinement. We showed our system completes well-sampled cuts on an Apple iPhone 6 in a lag-free manner even for multi-megapixel images. We believe our approach to mobile image segmentation lays a foundation for further image manipulations performed in real time for mobile device users.

References

[1] K. S. A. L. P. F. A. Radhakrishna, A. Shaji and S. Susstrunk. Slic superpixels. *EPFL*, Technical Report 149300, June 2010.

[2] E. H. Adelson, C. H. Anderson, J. R. Bergen, P. J. Burt, and J. M. Ogden. Pyramid methods in image processing. *RCA engineer*, 29(6):33–41, 1984.

[3] M. Birinci and K. Ugur. Interactive image segmentation based on superpixel grouping for mobile devices with touchscreen. pages 1–6, 2014.

[4] Y. Boykov and M. Jolly. Interactive graph cuts for optimal boundary and region segmentation of objects in nd images. *Proc. ICCV*, pages 105–112, 2001.

[5] Y. Boykov and M. Jolly. An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision. *IEEE PAMI*, 26(9):1124–1137, 2004.

[6] G. Bradski. The opencv library. *Dr. Dobb's Journal of Software Tools*, 2000.

[7] W. L. Briggs, V. E. Henson, and S. F. McCormick. *A Multigrid Tutorial*. SIAM: Society for Industrial and Applied Mathematics, Philadelphia, PA, 2 edition edition, June 2000.

[8] J. Chen, S. Paris, and F. Durand. Real-time edge-aware image processing with the bilateral grid. *ACM Trans. Graph.*, 26(3), July 2007.

[9] D. Comaniciu and P. Meer. Mean shift: A robust approach toward feature space analysis. *IEEE Trans. Pattern Anal. Mach. Intell.*, 24(5):603–619, May 2002.

[10] L. G. S. D. Lui, K. Pulli and Y. Xiong. Fast interactive image segmentation by discriminative clustering. *MCMC '10 Proceedings of the*

2010 ACM multimedia workshop on Moblie and cloud media computing, pages 47–52, 2010.

[11] I. Gallo, A. Zamberletti, and L. Noce. Interactive object class segmentation for mobile devices. *image*, 18:20.

[12] Y. Gao and X. Liu. Integrating bayesian classifier into random walk optimizer for interactive image segmentation on mobile phones. pages 1–6, 2014.

[13] C. Jung, M. Jian, J. Liu, L. Jiao, and Y. Shen. Interactive image segmentation via kernel propagation. *Pattern Recognition*, 47(8):2745–2755, 2014.

[14] M. Kass, A. Witkin, and D. Terzopoulos. Snakes: Active contour models. *International Journal of Computer Vision*, 1(4):321–331, 1988.

[15] V. Kwatra, A. Schödl, I. Essa, G. Turk, and A. Bobick. Graphcut textures: Image and video synthesis using graph cuts. In *ACM SIGGRAPH 2003 Papers*, SIGGRAPH '03, pages 277–286, New York, NY, USA, 2003. ACM.

[16] V. K. L. Vincente and C. Rother. Watersheds in digital spaces: an efficient algorithm based on immersion simulations. *IEEE PAMI*, 13(6):583–598, 1991.

[17] D. Liu, Y. Xiong, L. Shapiro, and K. Pulli. Robust interactive image segmentation with automatic boundary refinement. pages 225–228, 2010.

[18] J. Liu, J. Sun, and H.-Y. Shum. Paint selection. *ACM Trans. Graph.*, 28(3):69:1–69:7, July 2009.

[19] E. N. Mortensen and W. A. Barrett. Intelligent scissors for image composition. In *Proceedings of the 22Nd Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '95, pages 191–198, New York, NY, USA, 1995. ACM.

[20] J. Ning, L. Zhang, D. Zhang, and C. Wu. Interactive image segmentation by maximal similarity based region merging. *Pattern Recognition*, 43(2):445–456, 2010.

[21] D. R. Olsen, Jr. and M. K. Harris. Edge-respecting brushes. pages 171–180, 2008.

[22] L. Reese and W. Barrett. Image editing with intelligent paint. 21(3):714–724, 2002.

[23] C. Rother, V. Kolmogorov, and A. Blake. "grabcut": Interactive foreground extraction using iterated graph cuts. *ACM Trans. Graph.*, 23(3):309–314, Aug. 2004.

[24] A. Vedaldi and S. Soatto. Quick shift and kernel methods for mode seeking. pages 705–718, 2008.

[25] C.-K. T. Y. Lit, J. Sun and H.-Y. Shum. Lazy snapping. *ACM Trans. Graph.*, 23(3):303–308, 2004.