Random Walker Image Segmentation on iPhone and iPad

By Quinn Liu

Faculty Mentor: Rene Vidal

Center for Imaging Science, Johns Hopkins University

Abstract

The goal of this project was to update the image segmentation app, iMixPics, originally developed for the iPhone 3GS to iPhone 5 and iPad. The new app, iMixPics2, was first adjusted to fit the new screen sizes of the new devices. Next we implemented the random walker algorithm to compute intermediate segmentations to be used as guesses for computing the final segmentation for increased accuracy and efficiency. Lastly, because the new iPhone 5 and iPad processors greatly vary in speed, the iMixPics2 was optimized specifically for each device.

1 The iMixPics2 App Overview

iMixPics2 is a backwards compatible iOS app for the newest model iPhone 5 and iPad. The first task given to the user is to choose an image to be segmented (Fig. 1a, b). After the image is chosen the user can color any pixels in the image green or red by using their finger. Pixels overlaid with a transparent green represent the background of the image and will be cut out (Fig. 1c). Pixels overlaid with a transparent red represent the object of the image that will be kept (Fig. 1d). When the user is done, the random walker algorithm can be executed to perform an optimized segmentation for the specific device the app is running on. The algorithm calculates the probability that each pixel in the image belongs to the background or to the object. The result is an image containing only the pixels that are more likely to be an object than part of the background (Fig. 1e, f). Finally a new background image can be presented as an image layer underneath the segmented image and the user now has the option to save the final image or start over (Fig. 1g, h). Figure 1 shows an example of how the app can be used for the iPhone 5:

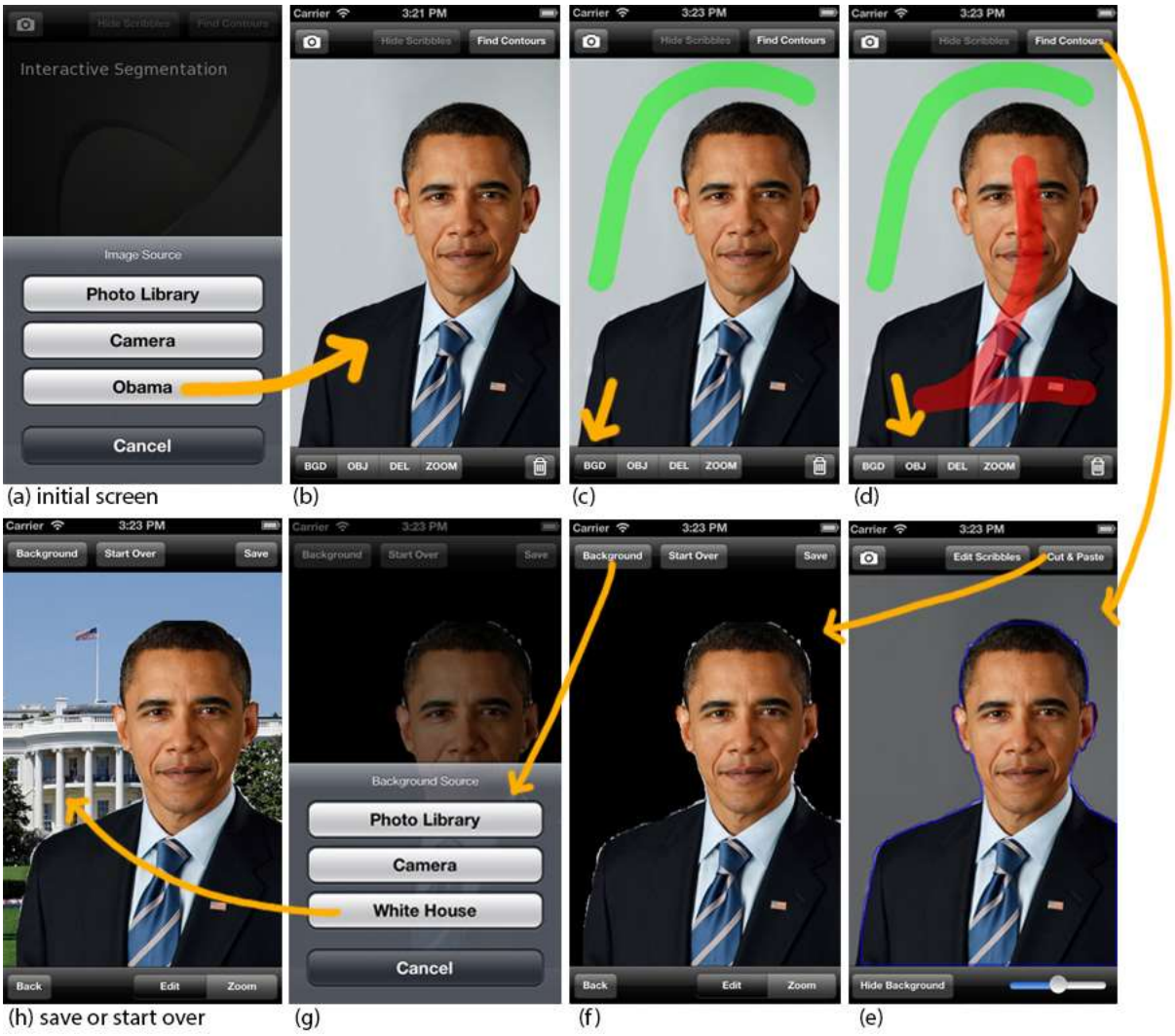

Figure 1: Example of using the app on the iPhone 5. The orange arrow represents actions the user made using the app. (a)-(d) In this example the user picked “Obama” and scribbled with their finger labeling pixels colored in green as the background and pixels in red as the object. (e), (f) After pressing “Find Contours” and “Cut and Paste” an image with the background pixels removed is presented. (g), (h) The user then picked “White House” as a background image. Finally the user can save the image or go back to the initial screen by pressing “Start Over”.

2 Fitting iMixPics2 App to all Screen Sizes

To allow the app to be correctly displayed specifically for the iPhone 5, the autosizing properties of every object seen by the user needs to be set correctly. This process is done in the visual app editor of Xcode. One example of an object with the correct autosizing properties set in Xcode is shown in Figure 2. In Figure 2 the red horizontal and dotted vertical arrows represents that the toolbar will resize correctly horizontally but will not adjust in size verticall. The three

red “I” letters represent that the toolbar object will always be attached to left, top, and right of the entire screen.

Figure 2: The above toolbar highlighted in dark blue with a visual depiction of its “Autosizing” properties displayed on the right above “Example”.

This is just an example of one object with the correct auto sizing properties being set. A similar process must be done for all other visual objects in the app.

3 The Random Walker Algorithm for Image Segmentation

After choosing an image, the user is presented with an image as in Figure 1b. The image can be represented as a graph, $G = (V, E)$. In this implementation a 4-connected graph was used to represent an image. Where v is a pixel in the image and e is pairs of pixels that are connected. Namely, a pixel is connected to the pixel above, below, to the left, and to the right of itself. Each of the edges is also assigned a weight representing how similar the corresponding vertices are in color intensity. The weight between 2 vertices i and j is defined as $w_{ij} = 2^{-B(g_i - g_j)^2}$ where g_i and g_j represent the color intensity value of the corresponding vertices in the image. B is a free parameter. This common weight equation creates edge weights that are large when given vertices have similar color intensity values and small when given vertices have dissimilar color intensity values.

When the user labels a subset of the vertices in the image graph, these vertices are labeled 1 for background and 2 for object. The goal of the random walker algorithm is to compute the labels for all the unlabeled vertices. This is done by determining a probability distribution for each unlabeled vertex. The probability for an unlabeled vertex to be labeled an “object” pixel is defined as the chance a random walker starting at the unlabeled vertex will reach an object labeled vertex before a background labeled vertex and vice versa. Because of this definition, each vertex will have 2 probabilities.

The probabilities for all the unlabeled vertices can be solved simultaneously, as shown in [1] using the Combinatorial Dirichlet problem. A linear system can be set up where there is an unknown variable for each unlabeled vertex. Let U be the number of unlabeled vertices and M be the number of labeled vertices. Let L_{UxU} be a Laplacian matrix that is equal to the degree matrix of the image minus the adjacency matrix of the image. Let x_{UxK} be a matrix with a row for each unlabeled vertex x that will hold the probabilities for each of K number of labels in one of the columns from 1 to K. Let $-B^T_{UxM}$ …

Let x_{MxK} contains the probability for each of the labeled vertices. Then solving for the probabilities x_{UxK} can be down in equation (1) since all of the other terms are known.

$$L_{UxU} * x_{UxK} = -B^T_{UxM} * x_{MxK} \quad (1)$$

For more information on the derivation of equation (1) please read [1].

4 The Random Walker Algorithm Implementation

The random walker algorithm can be summarized as follows.

randomWalker(originalImage, labeledImage, intermediateProbabilities, finalProbabilities) {

1. calculate weights of all edges in image graph in O(N^2)
2. assign known values in L_{UxU} , x_{MxK}, and $-B^T_{UxM}$ matrix in O(N)
3. fill in initial guess in O(N^2)
4. solve for x_{UxK} in O(N) + O(4 * N^2)
5. fill in calculated probabilities into finalProbabilities in O(N)

}

Running the random walker algorithm once will take O(N^2+ N + N^2+ N + 4 * N^2+ N)

= O(3N + 6N^2), where N is the number of vertices in the input image graph.

5 Random Walker Optimization for different iOS devices

The motivation for optimizing the algorithm and down sampling input images is because running the random walker algorithm on the original input image takes too long. For example, running the app on a Mac 2.4GHz Intel Core 2 Duo with the iPhone 5 Simulator takes over 180 seconds to segment an image. Furthermore, since the app was previously optimized for the iPhone 3GS, the algorithm needs to be optimized for the newest devices including the iPhone 5 and iPad 4. The processing power of the target iOS devices compared with the iPhone 3GS are summarized by the following equation (2):

$$iPhone\ 5\ speed \approx .91 * iPad\ 4\ speed \approx 1.91 * iPad\ 3\ speed \approx 2.07 * iPad\ 2\ speed \approx 5.8 * iPhone\ 3GS\ speed\ [2] \quad (2)$$

The key to making the algorithm run accurately and efficiently is by doing the following iterative process shown in Figure 3:

1) down sample the original image
2) run the random walker algorithm on the down sampled image with no initial guess and generate the probability matrix
3) down sample the original image less

4) use the previously generated probability matrix as input for the random walker algorithm on the less down sampled original image
5) stop when a desired probability matrix accuracy or maximum computation time has been reached

Figure 3: An example of how the random walker algorithm is used multiple times to calculate finalProbabilities.

All input images whether from photo library, camera, or Obama are initially down sampled to fit within a 320x460 matrix for the iPhone 5 and a 768x915 matrix for the iPad. To find out what down sample rate is both accurate and efficient I created 8 different down sampling configurations to test while running on the iPhone 5 and iPad device. The notation I use to describe a down sample configuration is a list of numbers. The example in Figure 4 is of configuration 1 applied to the iPhone 5's 320x460 input image size. The respective configuration notation is 64, 32, 16, 8, 4, 2, and 1. The list of numbers represent how much the input image is down sampled in between runs of the random walker algorithm as shown in Figure 3.

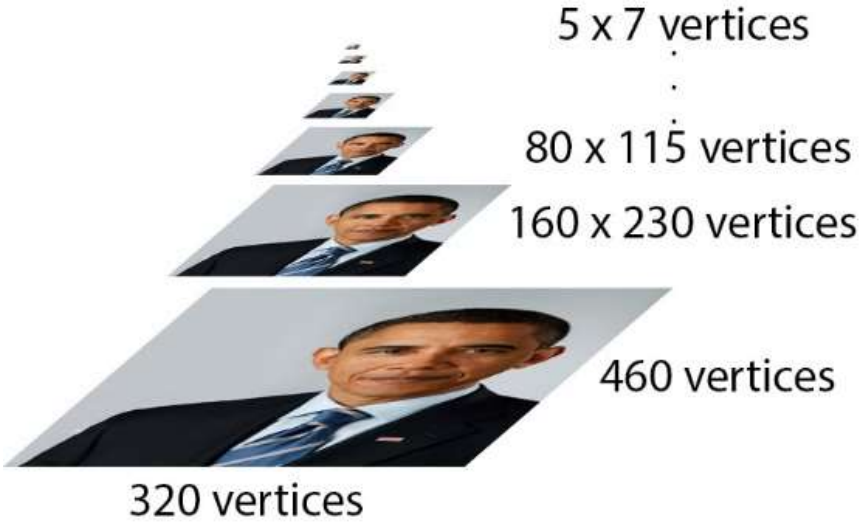

Figure 4: Example of configuration 1 for the iPhone 5 screen. This down sampling configuration 1 can also be represented as 64, 32, 16, 8, 4, 2, and 1

The input 320x460 image matrix is first down sampled by 64 in both dimensions creating a 5x7 matrix representation of the original image. Then the random walker algorithm is run provided the 5x7 down sampled image matrix with labels to compute a 5x7 probability matrix. Next, the original 320x460 image matrix is down sampled by 32 creating a 10x14 matrix representation of the original image. However, this time the random walker algorithm uses the previously computed 5x7 probability matrix as an initial guess for computing the 10x14

probability matrix thereby reducing the computation time. This process is then repeated until you reach the last number in the configuration list which in this case is 1. This means that the final probability matrix will be as accurate as possible because in the final run of the random walker algorithm the original image is not down sampled (320/1 = 320). The key here is that because we have 160x230 probability matrix from the previous run of the random walker, the 160x230 probability matrix can serve as an initial guess for this final run of the random walker causing it to take less time.

Because it eventually uses the original non down sampled image, this configuration creates the most accurate segmentation as shown in Table 5, but it is also the slowest configuration. To find an optimal configuration I evaluated the speed and accuracy of 8 different configurations on the iPhone 5 and iPad.

	iPhone 5	iPad	comments
Configuration 1	64, 32, 16, 8, 4, 2, 1	128, 32, 16, 8, 4, 2, 1	Most accurate but slowest out of the 8 configurations.
Configuration 2	64, 32, 16, 8, 4, 2	128, 64, 32, 16, 8, 4, 2	
Configuration 3	64, 32, 16, 8, 4, 3	128, 64, 32, 16, 8, 4, 3	
Configuration 4	64, 32, 16, 8, 4	128, 64, 32, 16, 8, 4	
Configuration 5	64, 32, 16, 8, 5	128, 64, 32, 16, 8, 5	
Configuration 6	64, 32, 16, 8, 6	128, 64, 32, 16, 8, 6	
Configuration 7	64, 32, 16, 8, 7	128, 64, 32, 16, 8, 7	
Configuration 8	64, 32, 16, 8	128, 64, 32, 16, 8	Least accurate but fastest out of the 8 configurations

Table 5: All configurations for iPhone 5 and iPad models

For testing a 320x460 Obama image for the iPhone 5 and a 768x915 Obama image for the Xcode iPad simulator was used. I connected the device to a Mac running Xcode to view code that printed the timestamp immediately before and after running the random walker algorithm. I ran the app 3 times for each of the 8 configurations and recorded the average time it took to run the configuration summarized in row 3 of Figures 6 and 8. In rows 2 of Figure 6 and 8 I computed the theoretical $_{\text{maximum}}$ number of unlabeled vertices that would need to be labeled. This maximum is theoretical because it does not account for labeled vertices created when the user overlays the image with green or red as in Figure 1c and 1d.

Actual iPhone 5 Results	Config. 1	Config. 2	Config. 3	Config. 4	Config. 5	Config. 6	Config. 7	Config. 8
Theoretical Maximum Number of Total Vertex Labels to Compute	196215	49015	28433	12215	8903	7043	5940	3015

Average Runtime after 3 Trails in seconds	19.54	3.46	1.29	0.65	0.29	0.25	0.23	0.11

Table 6: Actual runtime results were collected for the iPhone 5 device

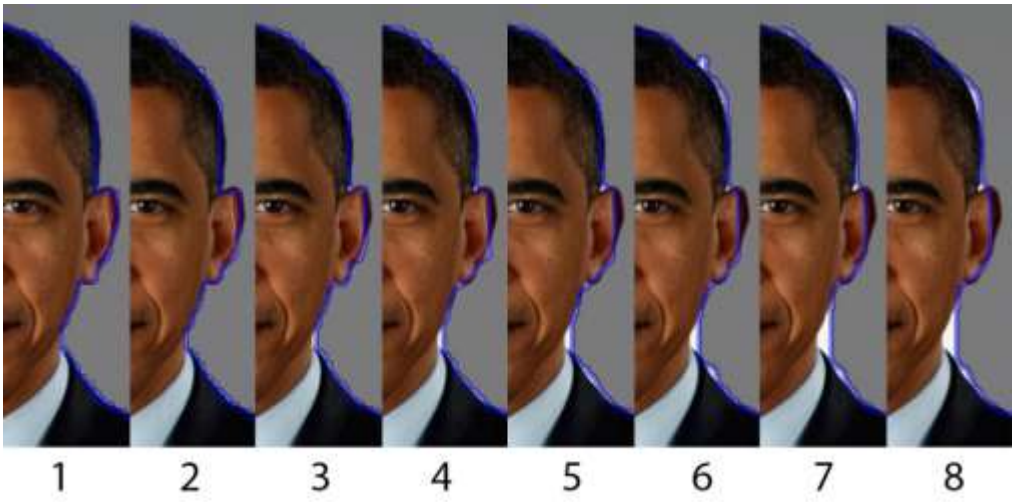

Figure 7: Configurations 1 to 8 segmentation accuracy for iPhone 5 screen

By using the iPhone 5 device test results and knowledge of the screen size of any other iPhone or iPad model, it is possible to estimate the computation times on the corresponding actual device. This estimation strategy can be summarized by the following equation:

$$iPhone\ 5\ runtime_i * \frac{device\ X\ total\ vertices_i}{iPhone\ 5\ total\ vertices_i} * \frac{device\ X\ runtime_i}{iPhone\ 5\ runtime_i} \approx\ device\ X\ runtime_i \quad (3)$$

In (3) "device X" can be any iPhone or iPad with a known screen size and speed relation to the iPhone 5 [2]. And subscript "i" can be any of the 8 configurations. By substituting "iPad 3" for "device X" in equation (3) we can determine the third term in the equation is: $\frac{iPad\ 3\ runtime_i}{iPhone\ 5\ runtime_i} \approx 1.91$ from equation (2).

By using equation (3), row 3 of Table 8 can be calculated creating an estimated runtime for an actual iPad 3.

Estimated iPad 3 Results	Config. 1	Config. 2	Config. 3	Config. 4	Config. 5	Config. 6	Config. 7	Config. 8
Theoretical Maximum Number of Total Vertex Labels to Compute	936546	233826	136418	58338	42561	34018	17788	14562
Average Runtime after 3	178.14	31.53	11.82	5.93	2.65	2.31	1.32	1.01

Trails in seconds								

Table 8: Estimated runtimes for iPad 3 using equation (3)

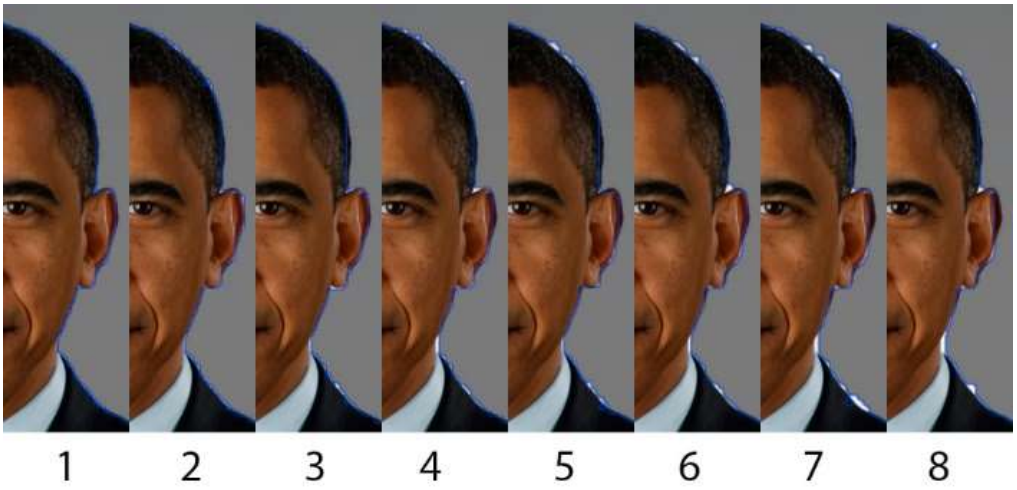

Figure 9: Configurations 1 to 8 segmentation accuracy for all iPad models

This process can similarly be done for all other models of the iPhone and iPad. The following chart summarizes estimated runtimes for the 8 configurations:

Runtime in seconds	Config. 1	Config. 2	Config. 3	Config. 4	Config. 5	Config. 6	Config. 7	Config. 8
iPhone 5(actual)	19.54	3.46	1.29	0.65	0.29	0.25	0.23	0.11
iPad 2(estimated)	192.78	34.12	12.79	6.42	2.87	2.50	1.42	1.10
iPad 3(estimated)	178.14	31.53	11.82	5.93	2.65	2.31	1.32	1.01
iPad 4(estimated)	83.94	14.86	5.57	2.79	1.25	1.09	0.62	0.48

Table 10: Summary of actual runtime for the iPhone 5 and estimated runtimes for the iPad 2, iPad 3 and iPad 4 using equation (3)

6 Conclusion

After reviewing Figure 10 for the iPhone 5 I decided to implement configuration 3 for when the app is run on the iPhone 5. This is because the accuracy of the application shows little improvement in Figure 7 while the time it takes to compute the segmentation increases exponentially. Similarly for when the app is running on an iPad I decided to implement configuration 4.

This internship provided many challenges. To overcome the challenges I am very grateful I was able to learn objective-c, Xcode, iOS, and develop a deep understanding of the random walker algorithm. Most importantly I learned the importance of code documentation after spending weeks debugging an uncommented project. However after documenting and implementing how the random walker algorithm is used, future developers of this app will easily be able to change how the random walker algorithm is implemented by adjusting only a couple lines of code.

References

[1] Leo Grady, Random Walks for Image Segmentation, IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 28 No. 11, Nov. 2006

[2] "IPhone, IPad, and IPod Benchmarks." - *Geekbench Browser*. Primate Labs Inc., 2013. Web. 25 July 2013. <http://browser.primatelabs.com/ios-benchmarks>.