CrowdBrush: A Mobile Photo Editing Application with a Crowd Inside

Yin-Chia Yeh

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2012-161
http://www.eecs.berkeley.edu/Pubs/TechRpts/2012/EECS-2012-161.html

June 4, 2012

Acknowledgement

We would like to thank Joel Brandt and Bjoern Hartmann for their continuous supervision throughout this work. Additionally, Joel Brandt provided us server access and funds to run experiments on Amazon Mechanical Turk and Bjoern Hartmann provided us Android handsets (Google Nexus One) to develop an android application and conduct user studies. We would also like to acknowledge the creative common image owners, whose images were used by us for this study. Also, I would like to thank our fellow classmates at UC Berkeley who helped us in initial user testing of our Android Application and gave us invaluable insights and opinions about our UI design and application features.

CrowdBrush:

A Mobile Photo Editing Application with a Crowd Inside

Abstract

We present a mobile photo editing application— CrowdBrush, which uses Google Android as front-end platform and Amazon Mechanical Turk as back-end platform. As there are increasingly more images were captured by smart phone these days, providing photo manipulation functions on the mobile phone directly become very important for better user experience. This is because most users want to edit and share their images directly after capturing, without having to copy them to PC. However, current photo editing software on smart phone usually has limited functions due to the restricted input methods on smart phone. Our system aims at providing more advanced photo manipulation features by uploading a user's image to our backend server, having workers on the internet help to process the image, and then sending the processed result back to the mobile phone. Specifically, we ask workers to perform a rotoscoping task on uploaded images. Rotoscoping is an essential preprocessing step for many visual effects. By having online workers label the objects and background in images, our system can provide various photo editing features which cannot be supported by current mobile photo editing applications. Our evaluation results show that our system can consistently generate high quality rotoscoping results cannot be done by computer vision techniques. The user study reveals important future working directions for users to adopt our system.

Introduction

Rotoscoping refers to the technique of creating a matte of a foreground element in image. This object matte is essential for various visual effects such as compositing an object to a different background or applying different visual effects on object and background. High quality rotoscoping has long been a labor intensive and expert oriented task because it requires marking the outline of object precisely. According to a discussion with engineers working on Adobe AfterEffects, a mainstream visual effects editing software, an expert rotoscope artist's hourly pay can be more than 70 dollars and rotoscoping a complex 30 seconds video clip could take up to 15000 dollars and more than 200 working hours. This fact results in applications require high quality rotoscoping inaccessible to low end market, such as consumer applications. Therefore, we believe that there exists value in discovering new solution for rotoscoping. Informational interviews with computer scientists working on computer vision suggest that a fully automatic solution for rotoscoping will not be there in the near future. This is mostly due to the difficulty of recognizing object boundaries in difficult conditions, such as poor lighting, occlusion, or complex shapes (e.g., fur). Therefore we conclude that human computer interaction is still necessary for rotoscoping. However, can we change the way rotoscoping is done now? As rotoscoping is a complicated expert oriented task now, one possible direction of improvement is to break it into small pieces of works that are easy for untrained workers be completed, which relates to the concept of crowdsourcing. If we can enable many untrained people working on rotoscoping instead of hiring few rotoscoping artists, how will it affect the labor cost and process speed of rotoscoping? In this project, we test the feasibility of using crowdsourcing technique to replace a hired rotoscoping artist to create a cheaper and faster rotoscoping solution.

To enable crowd workers to perform rotoscoping tasks, there are two main difficulties. First, the task must be divided into small pieces that untrained people can easily learn and perform well. Trained rotoscoping artists use professional software with some complicated curve editing tools to mark the outline of object. Such tools require significant expertise to use well. To enable untrained crowd workers to perform the same task, we must design a simpler tool for them. The second difficulty is how to aggregate pieces of work from crowd workers to an acceptable object matte. Quality control is especially important because the quality of work of crowd workers varies a lot. This is because either they do not understand the purpose of the task or they try to earn more money by completing more tasks with poor quality. We develop a crowdsourced rotoscoping pipeline for single image which first asks crowd workers to draw a matte for the image with a simple and easy to use interface; then a different group of workers will vote for the best working results in previous step. Finally, we combine those best results to generate the matte for input image.

To verify the feasibility of our crowdsource rotoscoping solution, we implement a mobile image editing application — CrowdBrush, which integrates our solution as a backend to extract foreground object from mobile phone images. This application enables some image editing features we have not seen in current mobile phone application market. Those features are cutting and pasting of objects into different background and applying different visual effects on objects and background. The evaluation we conducted shows that the rotoscoping quality of our crowdsource system outperforms benchmarking computer vision techniques. Our user study shows that users are interested in those image editing features we provided while there are still several aspects needed to be improved to provide better user experience.

The rest of this paper is organized as follows: section 2 summarizes the related works; section 3 describes our system and how it is implemented; section 4 shows how we evaluate our system; section 5 describes evaluation results of our system and discussion; section 6 is the future work.

Related Work

Crowdsourcing

The idea of crowdsourcing is to outsource tasks to group of people, such as internet online workers in our system. Here we review several crowdsoucing applications that inform our system design. Quinn et al [3] classify existing human computation systems based on their similarities and also provide insights towards future research work in human computation. Soylent [4] aims to utilize human intelligence from crowdsourcing to improve writing efficiency and quality. The three useful features of Soylent: Shortn, Crowdproof and The Human Macro illustrate different procedures which can be group-sourced. They also propose a Find-Fix-Verify model to break complicated article editing tasks into small chunks of tasks manageable by crowd worker, which informs our crowdsourced pipeline design. Von Ahn et al introduce the concept of GWAP (Game-With-A-Purpose) [5], which gives insight into the motivational power of games over players who participate in human computation activities. Little et al. [6] systematically analyze the performance and trade-off between iterative and parallel processes of human computation. Spiro et al [7] designed a crowdsourcing application of gesture annotation which performs video labeling. They identify the most important factor for success on the crowdsourcing platform is the design of user interface for crowd workers. They also point out that the design of the pipeline could consist of both human workers and computer, which is adopted by our crowdsourced pipeline as well.

Image Selection Interfaces

Wang and Cohen [8] give an overview of current performance of computer vision algorithms on image or video matting. They conclude that while computer vision algorithms do a good job on a single image (such as Chung et al. [9] and Sun et al. [10]) current performance of video rotoscoping algorithms is still limited. Wang and Cohen [11], Bai and Sapiro [12] present scribble-based interfaces which require users to simply specify a few scribbles of foreground and background as input and the algorithm generates the foreground mask for the input image. Interactive Video Cutout [13] offers a novel approach to treat video as 3D volume and provides painting based UI to indicate foreground object across space and time. However, it is still unclear if this kind of user interface is good for online workers since they are mostly more familiar with traditional frame based UI.

CrowdBrush System Design and Implementation

Pipeline Overview

Figure 1 shows the high level overview of our system. Users can upload images from their Android mobile phone to our web backend, which is composed of PHP/MySql and Python scripts. The backend system will post tasks on Amazon's Mechanical Turk platform. Crowd workers can then use our Flash drawing application to draw an object mask on the image. The object mask will be sent back to our web backend. The web backend will combine workers' result to generate final object mask, which will be downloaded to Android device for user to perform various image editing operations. We will explain our design and implementation of each of these components in the following paragraphs.

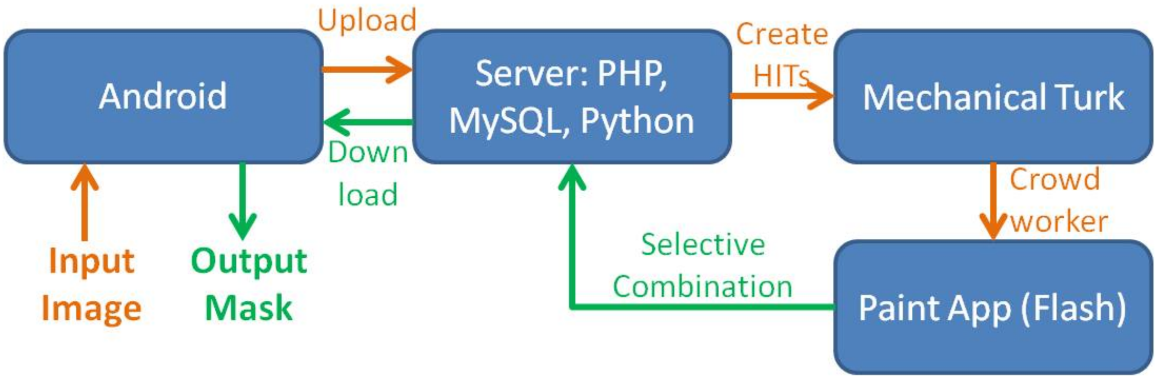

Figure 1: The overall pipeline of CrowdBrush system. The image captured by Android device is uploaded to the web server and being processed by crowdsource platform. Once the object mask is created the Android device will download the mask for image editing.

Web Backend

Figure 2 illustrates our overall pipeline design for the web backend system. When the input image uploaded to our web backend, we post a task on Amazon Mechanical Turk platform. Since working results from crowd workers varies a lot, we ask for several workers to draw the object mask of that image using our Flash web application. After collecting those masks from different workers, our web backend will run a heuristic pruning algorithm to filter out masks differ from the median of input image too much. For those masks passing the pruning stage, we post another task on Mechanical Turk asking workers to vote for best three masks. Here again we ask multiple workers to vote to eliminate the variance of workers' output. The web backend will then combine those top ranked masks to generate the final output mask.

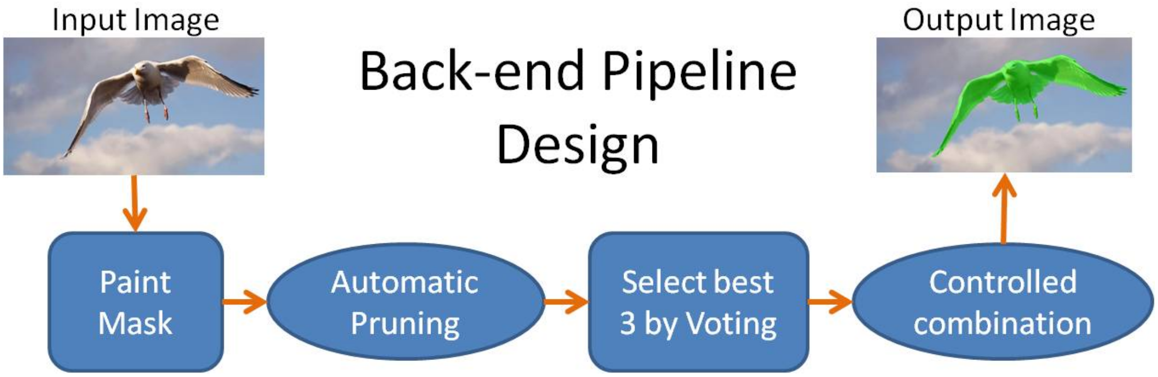

Figure 2: The overall pipeline of web backend

My main contribution on the web backend is the Flash drawing application and the flow control of the PHP/MySql part. For the Flash drawing application, the main design goal is to provide untrained workers an interface that is easy to learn and use. We choose to implement a classic painting program similar to the built-in painter program of windows operating system, in which workers simply use mouse to control the brush to draw on the uploaded image and the labeled pixels will be marked by transparent green color. To help workers perform more efficiently, we provide options such as changing the brush size, eraser brush, region filling, undo, redo and mask preview. The application is implemented by Flash action script 3 with Graffiti action script3 bitmap drawing library [16]. The user interface of the Flash drawing application is shown in figure 3 below.

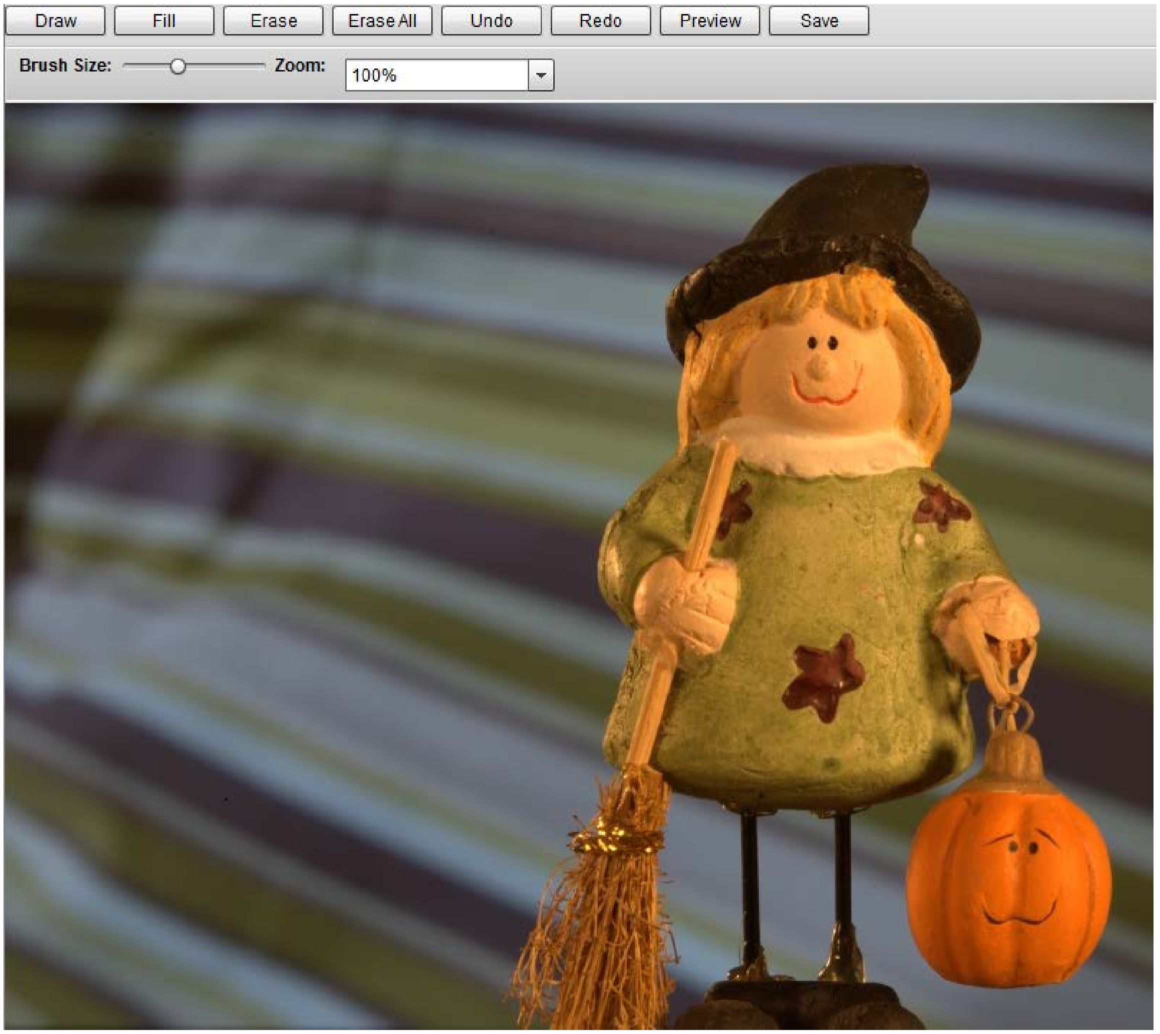

Figure 3: Painting UI with features and user submissions

For the PHP/MySql part, I worked on the interface between PHP and MySql database and the interface between PHP and Python scripts. We use the MySql database to track the work progress of each uploaded image in our pipeline. When an image is uploaded, we create a database entry to track how many masks corresponding to this image has been submitted. The Flash application also relies on this record to decide which image should be shown to workers to work on. Once this number exceeds a predefined threshold, we set a flag in database so the image will not be shown to workers. We then call the pruning Python script to prune outlier input masks and post the voting task on Mechanical Turk. When the voting is done we will call another python script to generate the final object mask for input image.

Android Application

The Android application contains several functions: capture and upload image to web backend; retrieve the output mask from web backend; perform image processing with the downloaded mask, where the image processing functions are owned by me. We have implemented three image processing functions that involves using the object masks. They are switch background, color splash and sticker. See figure 4 below for an overview of these features. The switch background feature allows user to load one image as background and paste the object onto arbitrary position of that image, which is the essential purpose of rotoscoping. The color splash feature demonstrates the other purpose of rotoscoping, which is filtering the object and background with different image filters. In our current implementation, the background is converted to gray scale while the object remains the same. We also allow users to adjust the overall brightness of object and background separately. Finally, the sticker feature is to draw a color bar on the boundary between object and background. We also learned something through the implementation of this Android application. First, due to the memory constraint of our testing Android device, we limit the image size to be processed to 480x640. Second, seemly simple image down-sample operation can actually be difficult. The bitmap scaling function provided by Android SDK will generate a lot of aliasing noise when the down-sample ratio is large. However, the down-sample routine built in the Android JPEG decoding function performs pretty well no matter the down-sample ratio is large or not.

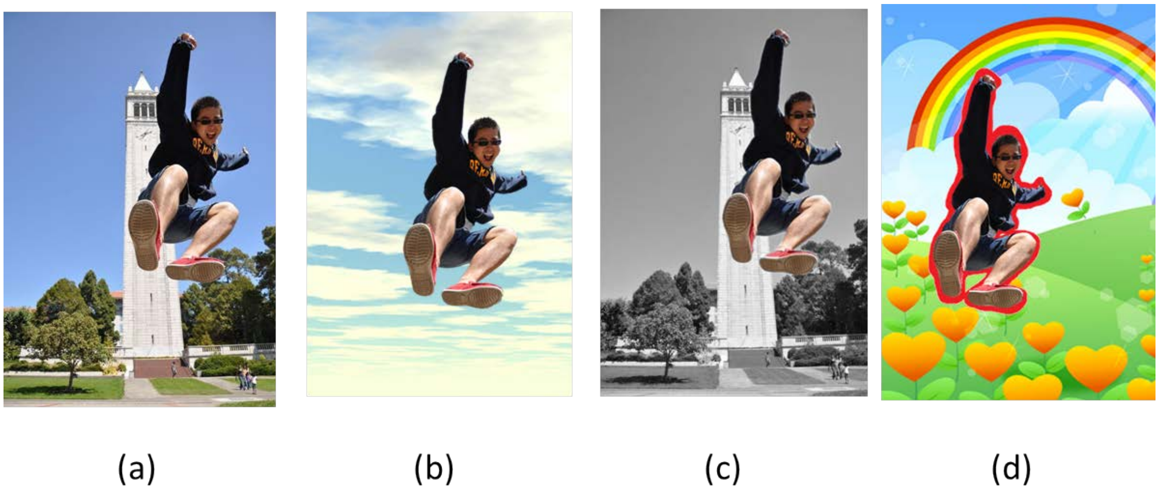

Figure 4: Overview of CrowdBrush image editing features. (a) Original Image (b) Switch background (c) Color Splash (d) Sticker

Evaluation Design

Evaluation of Backend Pipeline

In the first half of our project, we developed the backend pipeline on Mechanical Turk platform. The design goal of this backend pipeline is to achieve a balance between cost, latency, and performance, which means we want to achieve a minimum cost and latency while maintaining an acceptable performance of object marking. We have tested three different pipeline settings as in figure 5. They are

1. Final mask is generated from merging the top ranked masks after the pruning step, which is exactly the same pipeline as described in previous chapter and figure 2.
2. Similar to above method 1, but a region based iterative refinement step is appended in the end of pipeline to obtain optimized boundary.
3. Similar to method 2, but only taking one initial mask and skipping the pruning and voting step.

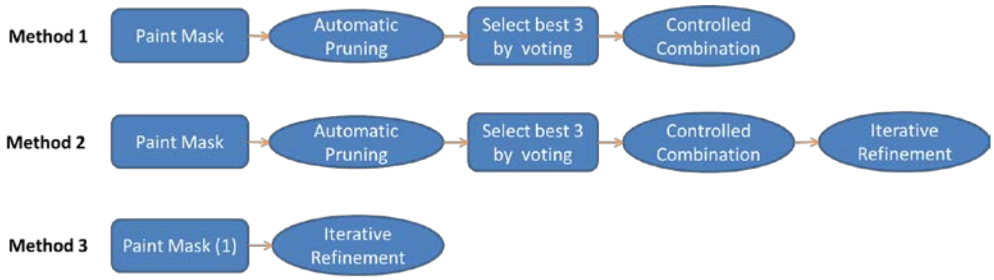

Figure 5: Three different pipeline settings we have evaluated in this report.

We applied these three different pipeline settings to a set of six test images categorized into three categories according to difficulty to mark the object within the test image.

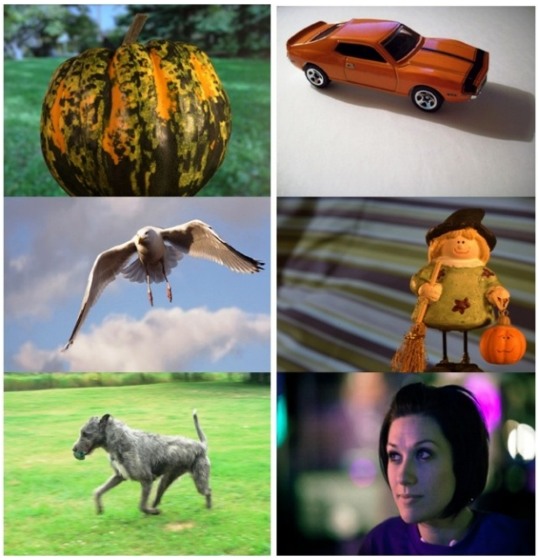

Figure 6: Test images of Mechanical Turk backend pipeline.

Figure 6 shows the six test images. The difficulty level is easy, medium, and hard from the top row to the bottom row respectively. The experiment results and discussion will be presented in the next chapter.

Evaluation of Android application

To evaluate our Android application design, we perform a user study to observe how users interact with this application directly. Our test user group consists of 20 subjects. All of their ages are between 18 and 25 and the average experience of using smart phone is two years. 60% of them are iPhone users while the rest are Android users. During this survey, we first ask them to fill in a background information survey, then take a picture and upload the image to the server. From here we split subjects to 2 groups, the first group's image is processed by our full crowdsourced pipeline and the second group's images are processed by us behind the scene to speed up the process. This variable should tell us how the users' feedback changes according to processing speed of uploaded image. Once the object mask is created, we ask users to try out those image features we implemented and fill in the user study survey attached in appendix.

Evaluation Results and Discussion

Evaluation Result of Backend pipeline

We report the spent time and money comparison with or without iterative refinement in figure 7. In sum, iterative refinement step takes a lot more time and money to complete. The cost with or without iterative refinement per image is 593 cents and 91 cents respectively and the process time is 52.5 hours and 6 hours. These numbers are all done with the settings of obtaining 10 initial masks and 7 voters. While iterative refinement step takes much more time and money, the output of this step does not necessarily improve. In figure 8, we can see that though there is little improvement in some parts of the mask, there could also be some parts of mask get worsen. We also find that method 3 does not work well. This is not only because of the inefficiency of iterative refinement, but also the instability issue caused by using only one initial mask. Iterative refinement relies on a good initial mask to extract regions to be optimized. When the initial mask is bad, the output of iterative refinement is always poor too. Therefore, we finally pick method 1, which uses multiple input masks and does not adopt iterative refinement, as our final backend implementation. Figure 9 is the sample output of method 1 pipeline. Note that in our final implementation we change the pipeline settings to 7 initial masks and 5 voters, which generates about the same quality level of object mask with lower cost (40 cents) and process time (~4 hours).

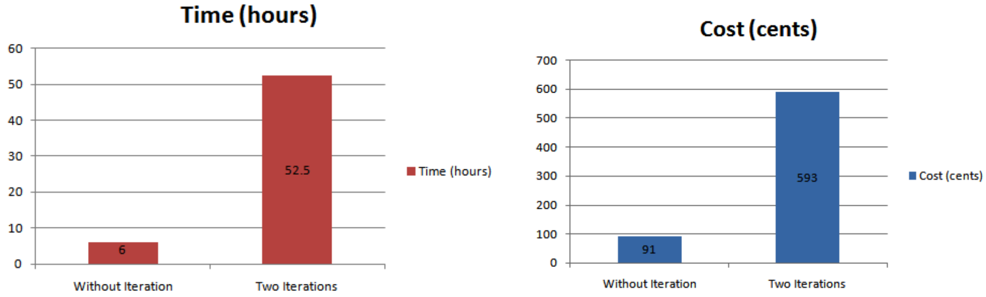

Figure 7: Comparison of spent time and monetary cost between the pipeline with or without iterative refinement.

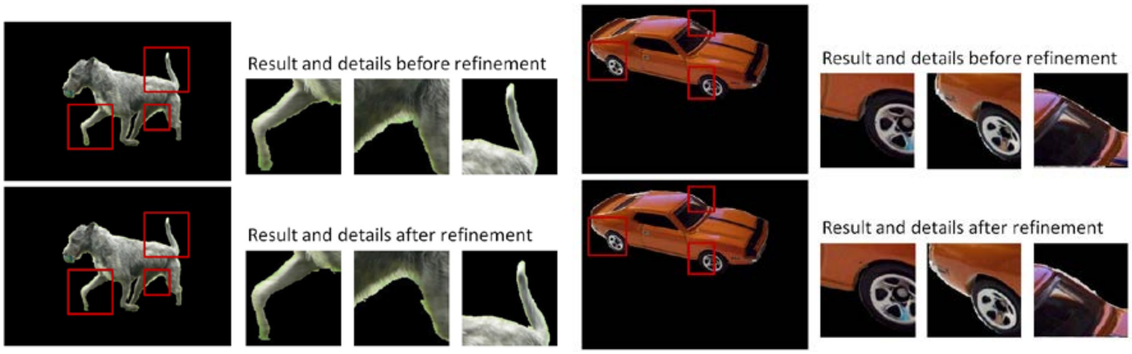
Result and details before refinement
Result and details after refinement
Result and details before refinement
Result and details after refinement

Figure 8: Iterative refinement results. In the left image, the boundary of the fur is slightly improved. In the right image, the boundary between the tire and car body is slightly improved but the boundary above the front window is worsen.

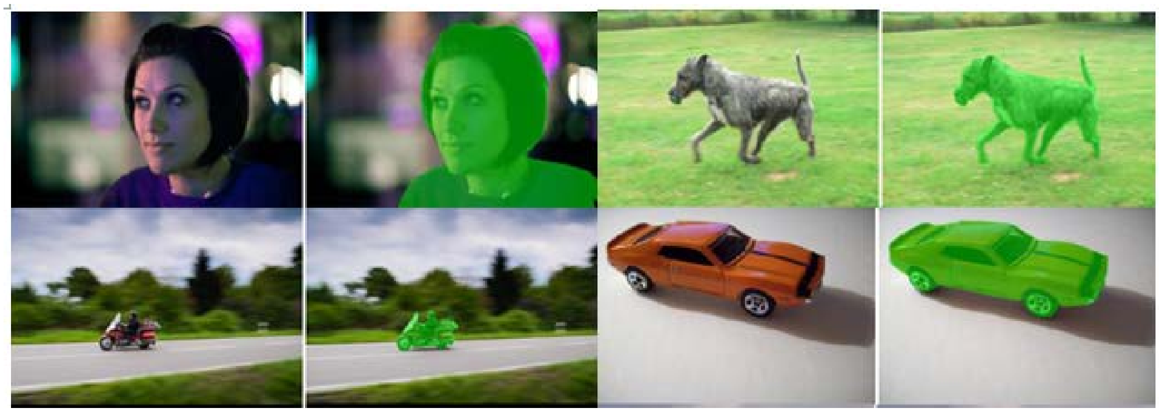

Figure 9: Sample output mask from our crowdsourced pipeline

Comparison with Computer Vision Approach—Grabcut

As another validation of our crowdsourced pipeline, we also compare our output object mask with one computer vision image matting approach—Grabcut [17]. The Grabcut algorithm models foreground and background color distribution by a Gaussian mixture model and models the image matting problem as binary classification of each pixel as foreground or background. Figure 10 shows the result from our pipeline and Grabcut. In (a), we see that in some simple scene both crowdsourced pipeline and Grabcut can generate good object mask. (b) is an interesting case where there is no obvious foreground object and Grabcut is confused. However our crowdsourced pipeline can still generate mask making sense semantically. In (c), Grabcut fails to capture the white font on the jacket because its color is similar to the white wall. Even though Grabcut does provide an iterative refinement mechanism for users to specify where the algorithm errs on the image, Grabcut still fails to capture the font after iterative refinement. One more thing to note is that though Grabcut fails to capture the font, it does create a better boundary of the person than crowdsourced pipeline, as we can see there are some white pixels marked as foreground along the boundary of the person. In (d), the color of the jeans and carpet is similar so Grabcut fails to distinguish them. Moreover, the result after iterative refinement eliminate part of the jeans from the foreground, which shows that iterative refinement does not necessarily converge to better result. In sum, we conclude that while there does exist computer vision approach that can perform well in certain scenes, it can still fail in more complex real world scenes. On the other hand, our crowdsourced pipeline can generate reasonable results consistently.

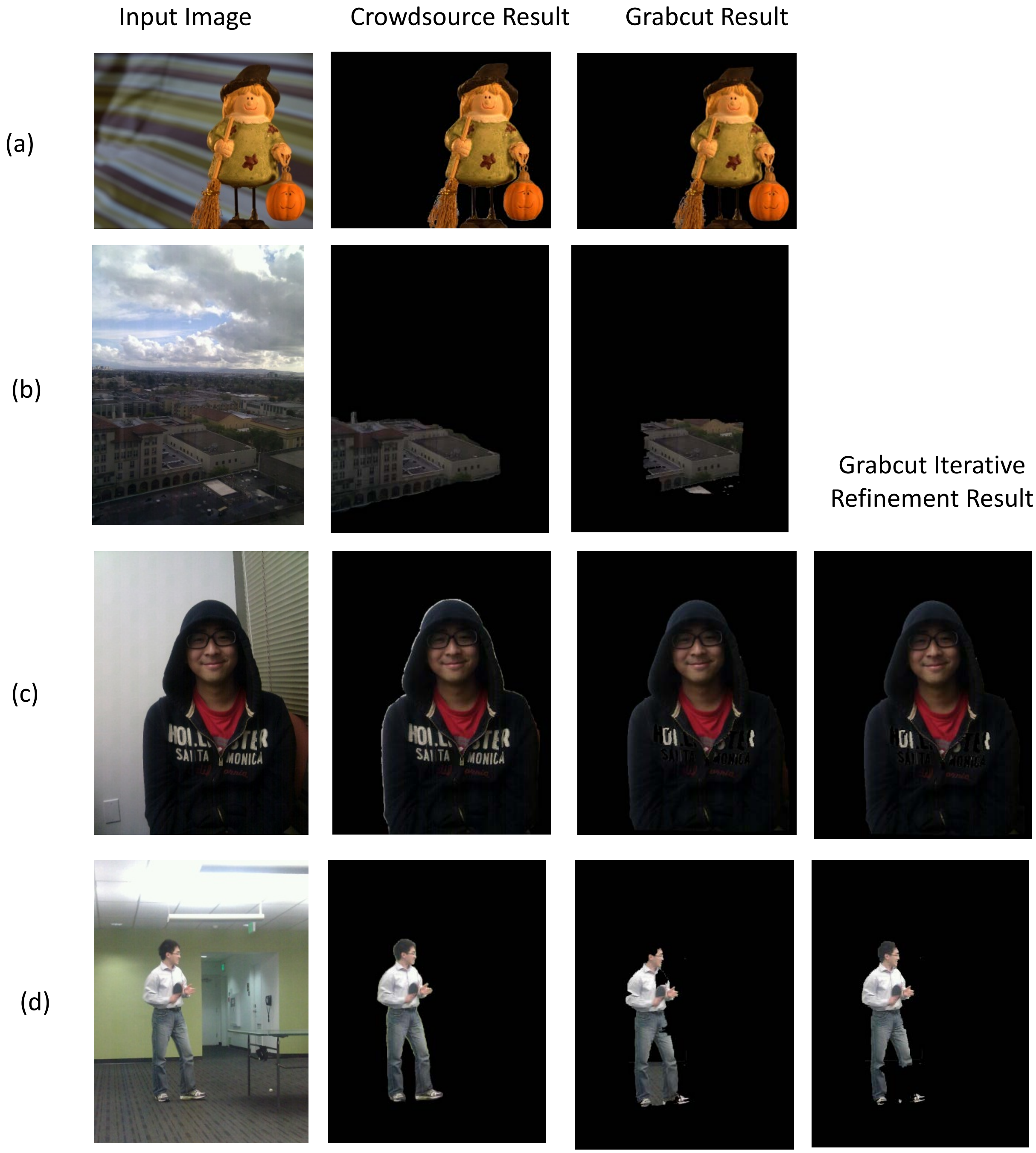

Figure 10: Comparison between crowdsource results with Grabcut result. The first column is input image; the second column is the result mask of our crowdsource pipeline; the third column is the result mask of Grabcut; the fourth column is the result of iterative refined Grabcut

Results of Android Application User Study

Among all 20 subjects, 80% of them used Facebook for sharing photos and on an average use social media on their phone 1-2 days every week. Only 10% users do in-app purchases but none of them is willing to pay for per image purchasing in CrowdBrush. On a scale of -3 to +3, the average user experience rating was 1.5. Most of complaint comes from the app is not responsive enough. This is due to the unstable network speed and the touch interface of our testing platform is not as good as latest phones. Among the three features we implemented, users are most interested in switch background, which they can paste their friends into a different background. 50% of them do use this feature more than once to create different images. Our users show a preference on instantly generated masks than current Crowdsourced pipeline. The most wanted feature from users is the ability to scale the object mask. Some other feedback suggest we should provide more background images to play with and also some built-in mask so users can play around it while their own image is still being processed.

Future Work

From the results of our evaluation, we conclude that there are two major directions of future work to improve this application

1. Improve the processing latency of Mechanical Turk backend: this can be done by maintaining a real-time worker pool as in [18] to make sure incoming tasks served immediately.
2. Improve the boundary accuracy of object mask: it will be interesting to incorporate some computer vision technique to help crowd workers to outline the object. For example, using computer vision approach to generate an initial result and let human work on it. However, it is debatable whether this will help human work faster or not.

If above problems can be solved, we believe not only our photo editing application will benefit, but also we can really build a video rotoscoping system based on crowdsource engine. Of course, there will be other problems needed to be solved in order to pursue this direction, for example, how to maintain the temporal coherency between frames and how to interpolate object mask between frames so that we do not have to upload each frame of a video clip on to Mechanical Turk.

Acknowledgements

We would like to thank Joel Brandt and Bjoern Hartmann for their continuous supervision throughout this work. Additionally, Joel Brandt provided us server access and funds to run experiments on Amazon Mechanical Turk and Bjoern Hartmann provided us Android handsets (Google Nexus One) to develop an android application and conduct user studies. We would also like to acknowledge the creative common image owners, whose images were used by us for this study. Also, I would like to thank our fellow classmates at UC Berkeley who helped us in initial user testing of our Android Application and gave us invaluable insights and opinions about our UI design and application features.

Reference

[1] Kulkarni, A., Can, M. and Hartmann, B. Turkomatic: Automatic Recursive Task and Workflow Design for Mechanical Turk. CHI 2011.

[2] Dow, S., Kulkarni, A., Klemmer, S., Hartmann, B. Shepherding the Crowd Yields Better Work. CSCW 2011.

[3] Quinn, A., Bederson, B. Human Computation: A Survey and Taxonomy of a Growing Field. CHI 2011.

[4] Bernstein, M., Little, G., Miller, R.C., Hartmann, B., Ackerman, M., Karger, D.R., Crowell, D., and Panovich, K. Soylent: A Word Processor with a Crowd Inside. UIST 2010.

[5] Von Ahn, L., Dabbish, L. Designing games with a purpose. Communications of the ACM 51, 8 (Aug. 2008), p. 58-67.

[6] Little, G., Chilton, L., Goldman, M., Miller, R.C. Exploring Iterative and Parallel Human Computation Processes. HCOMP 2010.

[7] Spiro, I., Taylor, G., Williams, G., Bregler, C. Hands by hand: crowd-sourced motion tracking for gesture annotation. CVPR Workshop on Automatic Vision with Humans in the Loop, 2010.

[8] Wang, J., Cohen, M. Image and Video Matting: A Survey. Foundations and Trends in Computer Graphics and Vision, Vol. 3, No.2, 2007.

[9] Chuang, Y., Curless, B., Salesin, D.H., Szeliski, R. A Bayesian Approach to Digital Matting. SIGGRAPH 2001

[10] Sun, J., Jia, J., Tang, C., Shum, H. Poisson Matting. SIGGRAPH 04.

[11] Wang, J., Cohen, M. Optimized color sampling for robust matting, in Proc. of IEEE CVPR, 2007.

[12] Bai, X., Sapiro, G. A geodesic framework for fast interactive image and video segmentation and matting, in Proc. of IEEE ICCV, 2007.

[13] Wang, J., Bhat, P., Colburn, A., Agrawala, M., Cohen, M. Interactive Video Cutout, in Proc. of ACM SIGGRAPH, 2005.

[14] Bai X., Wang J., Sapiro G.. Dynamic Color Flow: A Motion-Adaptive Color Model for Object Segmentation in Video. Proc. ECCV 2010.

[15] Bai X., Wang J., Simons D., Sapiro G. 2009. Video SnapCut: Robust Video Object Cutout Using Localized Classifiers. ACM Transactions on Graphics (Proc. SIGGRAPH), 28(3)

[16] http://www.nocircleno.com/graffiti/

[17] Carsten Rother, Vladimir Kolmogorov and Andrew Blake. Grabcut: Interactive Foreground Extraction Using Iterated Graph Cuts. SIGGRAPH 2004

[18] Michael Bernstein, Joel Brandt, Rob Miller, and David Karger. Crowds in Two Seconds: Enabling Realtime Crowd-Powered Interfaces. UIST 2011.

Appendix

User Study template:

Survey 1 (In - Application testing)
1. Do the following:
Login, Take picture, Save picture, Submit the picture for mask generation, wait till mask comes back, change background.
2. Select the mask,try the sticker feature, save the image.
3. Select the mask,try the color splash feature, save the image.
Survey 2
1. Sex : Male / Female / Prefer not to answer
2. Age group: 18-25 / 26-35 / 36+ / Prefer not to answer
3. Education background: High school / College Degree / Masters / PhD / Prefer not to answer
4. How long have you been using smartphones? ____ years and ____ months
5. Have you used both Android and iPhone? _____________________________
6. Which one are you using currently? __________________________________
7. How often do you use camera in your smartphone to take pictures?
Less than once a week /1-2 times a week/ Once a day/ More than 3 times a day
8. Do you use social media on your phone (ex: facebook, google+, twitter, tumblr etc)? Yes/ No
If yes, which ones do you frequently use?
____
9. Do you upload pictures to the social media websites from your phone? Yes / No
If yes, which ones (check all that apply) - facebook / twitter / tumblr / google+, __
10. How often do you upload to social media websites using your phone?
Less than once a week /1-2 times a week/ Once a day/ More than 3 times a day
11. Which photo uploading apps do you use, if any?
____
12. Any specific features you like about that application/ What made you choose that application?
____
13. Do you buy apps, or spend money on in-application purchases? Yes/No
If yes, which apps do you spend money on? _____________________________________
Survey 3 (Post Application testing)
1. How was the experience? (-3: Hated it, 3: Loved it)
-3 -2 -1 0 1 2 3
2. Did you feel that the application did something unexpected during your navigation? Yes/No
If yes, explain ____
3. Do you think if something like this was available for free, you would use it? Yes/No
4. If you were to pay for this application, how much would you be willing to pay? _______________
5. Please list the features (which you remember) this application has.
____
6. What feature interested you the most, why?
____
7. Is there something else you would have wanted to do with the mask?
____

8. How much did the amount of wait time influence your likability of the application?
-3 -2 -1 0 1 2 3
9. What kind of images do you think you would like to use this application for?
____
10. Anything else you would like to add /comment?
____