Foreground Detection and Prototyping of Photographic Composition on Android

Marek Salat*
Supervised by: Adam Herout†

Faculty of Information Technology
Brno University of Technology / Czech Republic

Abstract

The goal of the project is to create a new image prototyping application. The application enables users to capture scenes and enhance reality in an innovative way using an ordinary smartphone. They can replace background of the captured scene and create collages or new original images. The creation image can be created and shared within seconds with minimal interaction. Proper foreground/background detection (image matting) is a vital process. The solution I am suggesting uses appropriate computer vision and image processing algorithms, namely *Global Sampling Matting*. The application is built for the Android platform and uses SDK together with NDK. Sections of the core algorithm are accelerated in the GPU via an OpenGL ES 3.1 compute shader. One part of my work focuses on optimizing algorithms and effective image processing on Android devices. Another part of the work aims to create an intuitive user interface that requires minimal interaction. At the moment, the application is in a pre-release state (open alpha testing) published on Google Play – ViralCam[1].

Keywords: Image composition, Scene prototyping, Foreground detection, Background extraction, Image matting, Global sampling matting, Matting, Trimap, Alpha mask, Android, Compute shader, GPGPU

1 Introduction

Users often have an idea on how to incorporate elements of one picture into another picture (see Fig. 1, 2 and 4). However, in many cases, they are not familiar with image editors (Photoshop, Gimp etc.). Most of the times, the programs are either too complicated or available for a price that is prohibitive to this kind of audience. They are not able to create what they want and they must rely on others (Fig. 2 and 1). Even if they try to solve the task using traditional tools it can be an unpleasant task (for example *magic lasso tool* in Photoshop). The task gets even more difficult when the user has to use a combination of techniques to adjust areas containing hair and semitransparent objects to achieve a satisfactory result.

Figure 1: The example of the image composition. The iconic iron throne from the *Game of Thrones* with a person sitting on it.

The inability to produce own composition or the frustration of doing so are the main issues which led to the project. With just a little help, these frustrated users themselves will be able to produce more creative work. That is the main reason for building *ViralCam*. It allows the user to see in real-time what is being captured and how this scene fits to the other picture. Combining the images together requires minimum interaction; the user just selects foreground and places it over a camera preview. The *ViralCam* is created for Android platform (Section 4).

Selecting the foreground or background (foreground or background detection) is a vital task for this project. The task is more commonly called *image or alpha matting* and is described in the following Section 2.

The project uses known image matting algorithms such as *Global Sampling Method for Alpha Matting* [6] covered in Section 3. The algorithm has been implemented in Android NDK. The nature of the algorithm, and the fact that some newer Android devices are equipped with *OpenGL ES 3.1 Compute Shader*, allowed that it has been massively parallelized and run on GPU (described at Section 4.2). The GPU parallelization sped up the whole process from 4 – 7 seconds to less then a second which could brought

*salat.marek42@gmail
†herout@fit.vutbr.cz

[1] Available for Android 4.1 – 6.0 https://play.google.com/apps/testing/com.salat.viralcam.app

Figure 2: Image shows that it does not pay off to ask for image editing on the Internet. The scene is simple, the user had a photo and wanted to put his own figure over the first photo. Request from *CollegeHumor* [4].

the project closer to a nice-to-have feature *real-time scene visualization*.

2 Alpha Matting

Throughout this paper, it is assumed that a color image **I** consists of a 3D discrete array of pixels (red, green, blue). The Alpha matting or the digital matting is associated with the problem of softly separating an image into a foreground image **F** and into a background image **B** from a single input image **I** along with with its opacity mask α. It means that **I** is formed by linear blending of **F** and **B** using α. These three images relate by matting Equation (1). Image matting is used in interactive image editing, video segmentation and also in film making.

$$\mathbf{I} = \mathbf{F}\alpha + \mathbf{B}(1-\alpha) \tag{1}$$

The matting problem cannot be solved uniquely since there are many possible foreground and background explanations for the observed colors [10]. Equation (1) shows that there are seven unknowns on one hand but only three equations (three unknowns for foreground color, three for background color and one for alpha, the only known is image color) solving them.

Despite the fact that the problem is inherently underconstrained, it could be solved by adding more information about the image. The additional information could take form of a scribble-set or trimaps (see Figure 3). Such information labels pixels into two groups: the first group defines pixels which are definitely foreground, the second group labels pixel which are definitely background. The remaining pixels are marked as unknown. The alpha value α is then calculated for unknown pixels only.

Even with a known alpha value and these constraints, the problem is still ill-posed (the alpha value may be estimated incorrectly in favour of foreground or background). Therefore, several solutions proposed other additional constraints [15].

2.1 Trimap as a User Input

As mentioned before, the input can be in the form of scribbles [9] or trimaps [8]. I have found scribbles to be unintuitive and most of the times, it was difficult to predict the result without knowing the principles of the underlying algorithm. Very often, the resulting image was completely different from the expectations.

In this project, I chose trimap as the input constraint. Trimap segments the image into three regions: *definite foreground*, *definite background*, and *unknown*. Alpha values are calculated only for unknown pixels using knowledge from other regions since theirs alpha values are known.

The trimap can be easily drawn and with a little help. A user is then able to create it on a first try in few seconds [8]. More about constructing trimap in the Anroid application can be found in Section 4.

The trimap quality plays a significant role in the precision of the resulting alpha mask (or image). Very good trimap can reduce the number of unknown variables that imply fewer variables to estimate. The thickness of the unknown region creates a considerable factor of a good trimap [13].

The project aims at effective trimap creation requiring minimal interaction and, at the same time it must be intuitive. Quality of the computed alpha mask and time to compute trimap are also taken in account.

Figure 3: Example of the matting problem. From the left *first:* input image. Second: user-defined trimap (blue for background, red for foreground and green as an unknown region). Third: computed alpha mask. Fourth: original image without background (foreground area and unknown region are merged). Last: background is replaced. Figure is borrowed from book *Image and Video Matting: A survey* [13].

3 A Global Sampling Method for Alpha Matting

The algorithm is inspired by *Robust matting* [12], *Knockout* [3], *Shared matting*[5]. All these methods collect nearby (in a certain metric, e.g., Euclidean/geodesic distance, or nearest on a ray) samples. He et al. [6] proposes use a global sample set that contains all available samples. The spatial distance and the color fitness are then considered simultaneously for selecting the good samples from this set [6].

The goal is to select a good pair of samples (foreground, background) for any unknown pixel from all candidate pairs. The algorithm comprises following steps (the following steps are brief summary of the algorithm, more information can be found in the seminal work [6]):

Create global sample set – foreground (background) sample set consists of all known foreground (background) pixels on the unknown region's boundary. Global samples are denoted as **FB** search space.

Extend global sample set – Add random pixel to global set. I have chosen to add the same amount of pixels as the total number of pixels in the global set.

Initialize samples – Each of the unknown pixel has its own sample. The sample consist of foreground pixel, background pixel, closest distance to foreground and background boundary, cost value and alpha. Sample initialization assigns random boundary pixel from the global set, cost value is set to infinity. The closest distance is found by iterating over global set and comparing distances.

Apply *SampleMatch* algorithm – All pixel from global set are sorted by intensity (actual sorting criteria does not matter [6]). The goal is to find a pair (foreground, background) of points in the **FB** search space for each unknown pixel which has the (approximately) smallest cost. The method iterates over *propagation* and *random search* stages. As He et al. [6] claims, ten iterations are sufficient.

Propagation – For the unknown pixel being scanned, its cost is updated by considering the current optimal sample pairs of its neighbouring pixels. I have chosen to search in 8-neighbourhood.

Random search – Intuitively, the random search step tests a sequence of random points in the neighborhood (in the **FB** space) of the current optimal point. The neighborhood radius decreases exponentially (in each iteration).

4 Android Application

The most important part of the project is to create the application that will respect UX and design principles. For this part I decided to implement an application for Android. According to Gartner, [14] Android shares more than 84% of the market at the moment. The second place belongs to iOS.

4.1 Android NDK Utilization

Implementing a real-time image processing application in interpreted language (Android main programming language is Java) can be challenging. Fortunately, Android offers capabilities of native code via Android NDK. Programmer can choose from C or C++. All devices are equipped with standard libraries such as STD for C++. There are some limitations, but for most cases, it is not an issue.

Programming in NDK brings significant performance improvement. The programmer may also target specific architecture and can compile optimized code for multiple CPU architectures (x86-x32, arm7, arm8). The current implementation of the *A Global Sampling Method for Alpha Matting* for images *800x600* on Nexus 5X takes 4–7 seconds depending on the size of the unknown region.

From my experience, the bottleneck on the Android platform is usually the memory. It is not recommended to use more than three images of the same size as the device screen. Older platforms allow allocating only 16 MB per application [1]. Furthermore, allocation of more continuous memory may be an issue; e.g. in case of a bitmap with dimension of the size of the screen. Other problems could be caused by reading randomly from a bitmap due to skipping large chunks of memory which are not cached. All these things must be considered when porting code to

android NDK.

My application assumes that users will create photos quickly, spontaneously and the results will be shared on social media. The typical image size on Facebook, Twitter, or 9gag is around one mega pixel. On the grounds of performance reasons and the fact that the image does not have to be in full resolution I have decided to scale input images to a lower resolution, somewhere around 0.5 to 1 mega pixel, typically 800×600.

4.2 GPGPU via OpenGL ES 3.1 Compute Shader

Some older Android devices supported OpenCL. It was a great tool for massive parallelization. Unfortunately, Google dropped the support and introduced their own parallelization library called RenderScript. From my point of view, the tool is badly documented and the programmer does not have a good control of whether the code ends up running on the GPU or just on the CPU. Android 5.0 introduced OpenGL ES 3.1 with compute shaders. At the moment, compute shaders are supported at least on 7.1 % devices [2]. It needs to be mentioned that a device running Android 5.0 or higher *may or may not* support the feature.

Programmers might take advantage of this technology. However, compute shaders are still bound to rendering pipeline and cannot be used without rendering on the screen (*GLSurfaceView* and *GLSurfaceView.Renderer must be used*[2]).

4.3 Implementation of the Compute Shader

The matting algorithm comprises of the following steps:

1. **Find the boundary for foreground and unknown region and boundary for background and unknown region.** Boundary is found by checking neighbourhood each pixel. If the pixel represents the foreground and at least one neighbour is labelled unknown, the pixel is assigned to the foreground set, similarly for the background pixel. The step is independent on other pixels. However writing to the global set must be synchronized in the way that no value is rewritten or maximal size is preserved. In the compute shader implementation atomic counters are used to prevent both issues.

2. **Extend the boundary by adding random pixels to the global set.** The step assigns random pixel from foreground region to the foreground boundary and analogous to the background pixel. The pixel is not assigned to the global set if global set is contains more than twice the amount of the original boundary size. The step is also independent on other pixels and can be run in parallel.

3. **Initialize samples for each unknown pixel.** First, the initialization, finds the distance to the nearest foreground and background pixel from the global set (measured by euclidean distance). Secondly, it assigns random foreground and background pixel from global set. The cost value is set to infinity, the alpha value is set to zero. The initialization is also independent on other samples.

4. **Iterate over *propagation* and *random search*.** The propagation searches in neighbourhood for better sample selection. Random search follows immediately afterwards. It selects random pixel from the global set by decreasing radius exponentially. Both propagation and random search update the cost value and the alpha value. As mentioned above in Section 3, the step is run for each sample (unknown pixel) and each sample processing is independent from others.

5. **Update alpha mask.** At this point the algorithm calculated all alpha values for unknown pixels. The step fills whole alpha mask by corresponding alpha values (255 for definite background, 0 for definite foreground and sample alpha value for its unknown pixel).

Each step must be completed before the next step can be run; the same applies to the iteration over *propagation* and *random search*. Thus, each step represents one shader (kernel) which is dispatched only if previous step has finished. The CPU implementation loops over all pixels in the image for in each step; each thread computes one pixel. The dimension of the workgroup is 32×32 and the whole problem has the dimension of the image (rounded and divided by size of the workgroup).

Better performance can be achieved by using texture units for reading from images. The texture unit caches images very well for access in neighbourhood around pixel unlike access in buffer.

Computing alpha mask has been sped up from 4-7 second to 300-500 milliseconds. This has been measured on Nexus 5X.

4.4 User interface

Figure 4 shows application capabilities. When the application is started, background is selected first. The Application offers several predefined images. The user can also choose pictures from an image library.

Scene capturing and visualization – At the moment, for the sake of MVP (minimal viable product) simplicity, I have chosen to simply overlap the camera output by semi-transparent background. It turns out that this solution is sufficient enough for visualizing the scene. The user can

[2] You can download the fragment I use at GitHub repository. You can also find there few examples showing how to compile the shader, bind buffers, textures, pass data through shader and also how to run them. Please visit *https://github.com/MarekSalat* for more information.

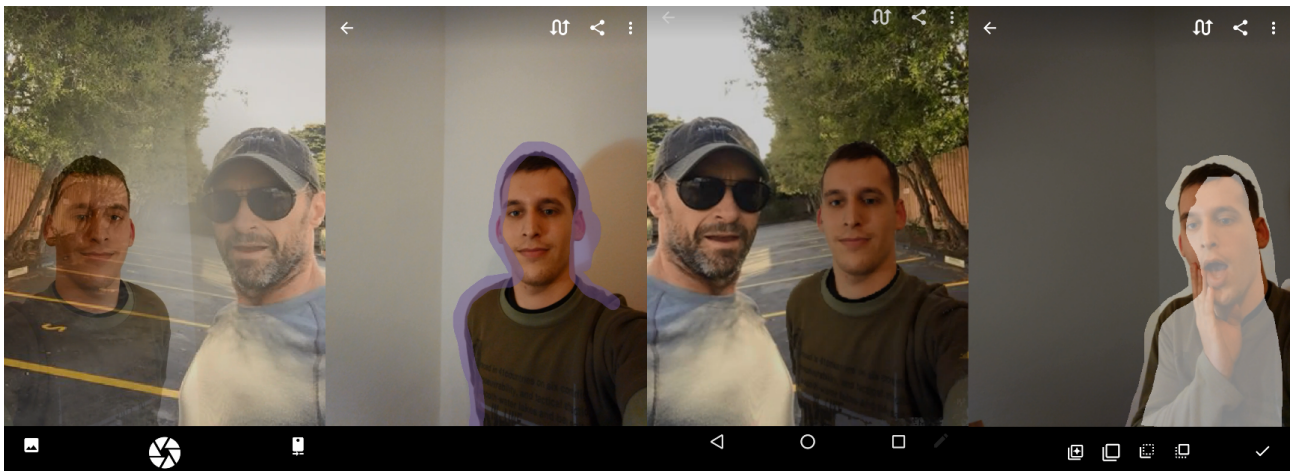

Figure 4: The first image shows the visualization of scene capturing. The second image shows trimap initialization. The third image is the result. Last picture contains trimap editing. .

swipe over the image to increase or decrease background transparency.

Trimap initialization – User roughly marks the object edge. This does not have to be precise to the pixel. The inner area is automatically marked as definite a foreground. Outer area is marked as definite a background. If the image contains larger transparent areas, such as hair or spikes, the trimap can be edited and these areas can be marked as an unknown region.

Show result – After trimap initialization, the application displays the result. At this step, the user can pinch to zoom to validate all the problematic parts of the image.

Edit trimap – User can always edit the trimap by simply drawing a definite foreground, background or unknown region (application user brush named *Clear*). In most of the applications, this step requires switching between different brushes by the user. To improve the user experience, I introduced a smarter brush which selects the brush based on the starting point of the user's swipe motion. In other words, if the user starts drawing from the background, the background brush is selected, other brushes are initiated by drawing from their respective areas. The final step is to share the result on social media.

5 Conclusions

At the moment, the published alpha application is using only the NDK implementation. The compute shader variant is not ready for production. The performance and quality has been tested on *LG Nexus 5X* with a standard dataset [11]. Provided dataset contains various images with trimaps with respective true alpha mattes. The dateset is composed of images from non-transparent to semi-transparent or even fully transparent images, also images with short or long hair.

Measured average time per unknown pixel is 0.11 milliseconds which is 7.2 seconds per image (roughly half megapixel). The 7-seconds processing time it is not close to the *real-time preview* goal. However, the compute shaders seem to be promising. For *real-time preview* the quality may be lower and the whole process may be sped up by scaling the image to a lower resolution. On the other hand, a significant drawback for the compute shader is lower support on Android devices (less than 7%).

The average MSE error without pre-processing or post-processing is 353. For comparison, the *Robust matting* [12] MSE is 350 on the same data set and the method is ranked on *alphamatting.com* as 36th. The quality could be better and it will be addressed in future releases, still it is sufficient enough for *MVP* (*minimal viable product*). The general image quality and composition perception will be a part of the future user testing evaluation.

There are several ways to increase the matte quality. First way is trimap pre-processing where colors closer to the unknown region boundary with similar color properties (color and spatial distance) are considered to be known depending on other regions. Such a pre-processing reduces the number of unknown variables and increases overall matte quality. The other method is the post-processing as He et al. [6] proposed. They used *Fast Guided Filter* [7] which ran 0.3 second per mega pixel, but it could be estimated to run slower on regular mobile device.

The application is published as an open alpha version on Google Play – ViralCam[1] for download.. Within few weeks, it is going to be published to the production. The most common issue so far was a lack of a help which was added in version 1.3. Other important issue coming from users were problems with camera focus and, on some devices, also image rotation after capturing a scene. All these issues have been addressed in last the update. However, the camera rotation issues still persists on *Sony Xperi E4g*. For the production version, *Google Analytics* will be integrated to gather more precise data about user acquisition and behaviour within the application.

References

[1] Android compatibility downloads. http://source.android.com/compatibility/downloads.html, 2016.

[2] Dashboards. http://developer.android.com/about/dashboards/index.html, 2016.

[3] A. Berman, A. Dadourian, and P. Vlahos. Method for removing from an image the background surrounding a selected object, October 17 2000. US Patent 6,134,346.

[4] CollegeHumor. 12 people who had photoshop requestsand got just perfect results. http://www.collegehumor.com/post/6997451/12-people-who-had-photoshop-requestsand-got-just-perfect-results, 2014.

[5] Eduardo S. L. Gastal and Manuel M. Oliveira. Shared sampling for real-time alpha matting. *Computer Graphics Forum*, 29(2):575–584, May 2010. Proceedings of Eurographics.

[6] K. He, C. Rhemann, C. Rother, X. Tang, and J. Sun. A global sampling method for alpha matting. In *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*, pages 2049–2056, June 2011.

[7] K. He, J. Sun, and X. Tang. Guided image filtering. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(6):1397–1409, June 2013.

[8] C. L. Hsieh and M. S. Lee. Automatic trimap generation for digital image matting. In *Signal and Information Processing Association Annual Summit and Conference (APSIPA), 2013 Asia-Pacific*, pages 1–5, Oct 2013.

[9] A. Levin, D. Lischinski, and Y. Weiss. A closed form solution to natural image matting. In *Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference on*, volume 1, pages 61–68, June 2006.

[10] Richard J. Radke. *Computer Vision for Visual Effects*. Cambridge University Press, New York, NY, USA, 2012.

[11] C. Rhemann, C. Rother, J. Wang, M. Gelautz, P. Kohli, and P. Rott. A perceptually motivated online benchmark for image matting. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 1826–1833, June 2009.

[12] J. Wang and M. F. Cohen. Optimized color sampling for robust matting. In *Computer Vision and Pattern Recognition, 2007. CVPR '07. IEEE Conference on*, pages 1–8, June 2007.

[13] Jue Wang and Michael F. Cohen. Image and video matting: A survey. *Found. Trends. Comput. Graph. Vis.*, 3(2):97–175, January 2007.

[14] Meulen R. Woods V. Gartner says emerging markets drove worldwide smartphone sales to 15.5 percent growth in third. http://www.gartner.com/newsroom/id/3169417, 2015.

[15] Y. Zheng and C. Kambhamettu. Learning based digital matting. In *Computer Vision, 2009 IEEE 12th International Conference on*, pages 889–896, Sept 2009.