An Enhanced Method for HDR Imaging: Artifact-Free and Optimized for Mobile Devices

Jihyeon (Janel) Lee

I. Personal Section

With the advent of the smartphone, cameras have suddenly become very convenient. Like many others, I enjoy taking pictures on my rather old, outdated phone. I became interested in how to take better-quality photos despite my device's limitations, and a Google search pointed me toward high dynamic range (HDR) imaging. The process involves taking multiple photos instead of just one to produce a final image that shows a real-world scene more completely. I started researching the limits of what HDR imaging currently has to offer, and many of the methods had yet to overcome obstacles, most of which often originated common problems (e.g. camera shake). I thought it was intuitive for there to be a way to automatically correct a picture when taken, especially when the problems precluded HDR imaging from working properly, and that's where my investigation for a solution began. My research goal was to create a comprehensive approach that could account for the various issues in taking HDR photos but also to optimize the algorithm to be implemented on mobile devices.

My project was almost entirely independent, and I conducted my research at home, turning the unstructured space of my room and PC into a lab, appreciating the versatility of computer science. Throughout the process, I did learn some additional concepts in math and science, especially related to computational photography, but more importantly, I learned new ways to apply concepts. For example, one of the problems I tackled was getting rid of blurring, caused by movement of the camera. In order to quantify the motion, I used cross-correlation to compare input images to each other and then calculated a moving vector to correct images that had moved with respect to a reference image. I had learned about both cross-correlation and vectors in school but in a much more rigid setting, restricted to homework problems or graphs in textbooks; while learning those concepts at the time, I never would have imagined I would be

using them in photography. Certainly, science and mathematics became more alive and real to me through research than through any other experience. Seeing and understanding them in action was beyond any example in a lecture, and actually utilizing and integrating concepts to create something new allowed me to synthesize information in a way I never had before in a classroom.

To other high school students who would like to undertake a project combining science and mathematics, my advice is to be prepared to face continuous challenges and to learn. I believe my research experience was a series of small discoveries that allowed me to eventually be able to present a comprehensive solution, and that process, although arduous, was very rewarding. There will be many times that you do not obtain the results that you expect or the output you desire, but the exhilaration of discovery will be worth the hard work.

II. Research Section

1. Introduction

The scenes of the real world that humans can observe have a high dynamic range (HDR), or a high ratio of the maximum to minimum amount of light intensity, which cannot be captured completely by a camera's sensors. Most digital cameras have a limited dynamic range and spatial resolution than that of natural scenes or even that of the human eye. The low dynamic range (LDR) images captured by cameras lose detail for extreme values of light intensity (very dark or bright areas) in a given scene, which exceed the capacity of the sensors. To overcome these limitations and capture all of the visual information present in a high dynamic scene, several hardware and software techniques have been developed [1, 2]. The most common approach consists of fusing a set or "stack" of LDR images, each taken with different exposure times and each focusing on a different region of the dynamic range. In this process, images are taken based on bracketed exposure times: short exposure provides data for bright areas, while long exposure for darker areas.

One of the major obstacles for the practical application of HDR imaging is that a given scene must be completely static in order to avoid various artifacts, or distortions, specifically blurring and ghosting. Most HDR techniques rely on perfectly aligned images, but this condition is rarely met in real-world situations due to camera shake and moving objects (people, clouds, etc.). Consequently, there is a great need for methods that align images in a stack by compensating for their displacement [6]. However, a reliable estimation is difficult since the stack images are taken at varying exposures and thus yield severe brightness changes. A group of methods has been proposed that assume a scene is static and that the LDR images are registered for camera motion, only addressing the problem of scene changes. Another group of methods modifies the fusion algorithm to account for potential sources of ghosting artifacts due to any moving objects in the scene [4, 7]. A more elegant approach consists of a non-rigid transformation between LDR images to address both camera motion and scene changes at the same time [5].

In this paper I develop an HDR imaging algorithm to handle the two main artifacts, blurring and ghosting. One component is the use of the Exposure Fusion (EF) approach by Mertens et al. [3], the aforementioned method that uses a bracketed image exposure sequence and fuses the stack of images together to produce a tone-mapped LDR image. Since the design and construction of a technique using HDR imaging targets handheld cameras, which do not have the same capabilities as large CPUs or GPUs, it is also important to consider the computing power of a mobile platform.

Over the last few years, processors in mobile devices not only increased in clock speed but also took the step of becoming multicore, increasing the raw computing power available significantly. Programmable embedded GPUs also deliver to the demanding performance needs. Using this computing power, I not only develop the algorithm but also implement it on a mobile device to demonstrate its usability and extensibility over heterogeneous computing. My proposed method both produces better images and does so more efficiently than those currently available.

2. Method

My research was conducted in two phases, 1. developing the core components of the algorithm on a PC using software tools such as Matlab and OpenCV and 2. porting those components to a mobile device platform and optimizing them for mobile devices using heterogeneous computing.

2.1 Artifact Removal

The first step was to identify the problems in HDR imaging and their sources. I found that most cases suffered from camera movement, especially for handheld, mobile devices without tripods. After trying to create HDR images from image stacks available on HDR imaging websites, I realized that many of the images are not well-aligned, and blurring artifacts were common (Fig 1). Since most of the objects in the scene are unmoving, it was not movement within the scene but of the camera that was the source of the blurring artifacts. In Figure 2, the first four steps after the "Start" refer to the first stage of the algorithm in aligning images.

First, each input image is converted to grayscale to generate images that show contrast differences and then normalized. Next, using cross-correlation, each normalized contrast image is compared to its successive image to find a matching position. Given two of the images, I_1 and I_2, the cross correlation R is

$$\mathrm{R}(i,j) = \sum_{x,y}[I_1(x,y) \cdot I_2(i+x, j+y)] \qquad (1)$$

where x and y correspond to image size, and the search range covering [-10,10]. The basic concept is that by finding the cross-correlation for two different images in the input sequence, since each is an image of the same scene, it is really equivalent to auto-correlation for the same image. My algorithm applies the correlation function to normalized contrast images instead of the original images with these problems, finding a matching position more efficiently and accurately. Figure 3 shows the subroutines of the steps in Figure 2 after receiving the LDR images from existing files or a camera. Figure 4 shows the normalized contrast images in CMY (Cyan, Magenta, Yellow)

channels respectively, C (first image), M (second image), Y (third image). This corresponds to the "Show normalized contrast images as a color channel" step, and it shows the results for debugging purposes (Figure 3). Compared to the left image before adjustment, the right image shows images are aligned, with clear black lines as a result (Figure 4). The output will be described later (Figures 7 and 8).

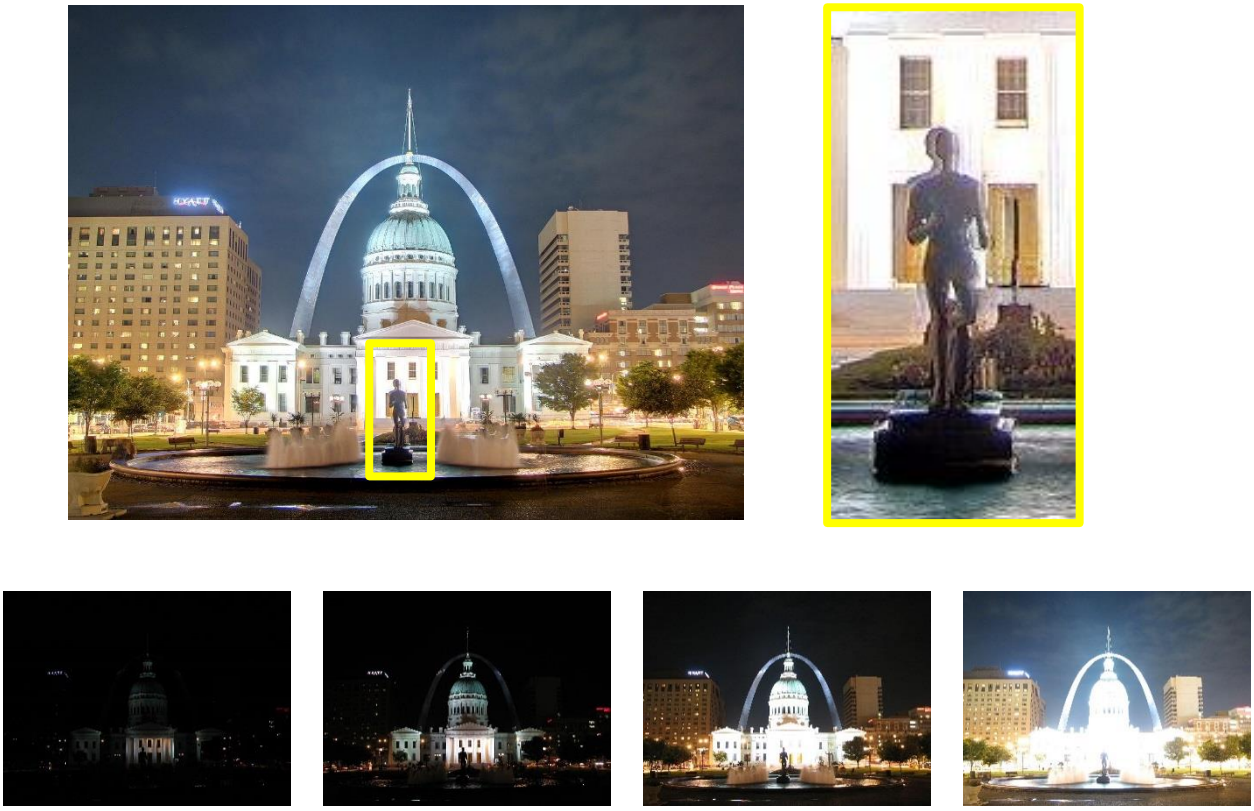


Fig. 1. The top left image is the simple HDR image using exposure fusion. The top right shows the portion of the HDR image enlarged to clearly show the blurring artifacts. The second row shows the original bracketed LDR exposures in the stack [13].

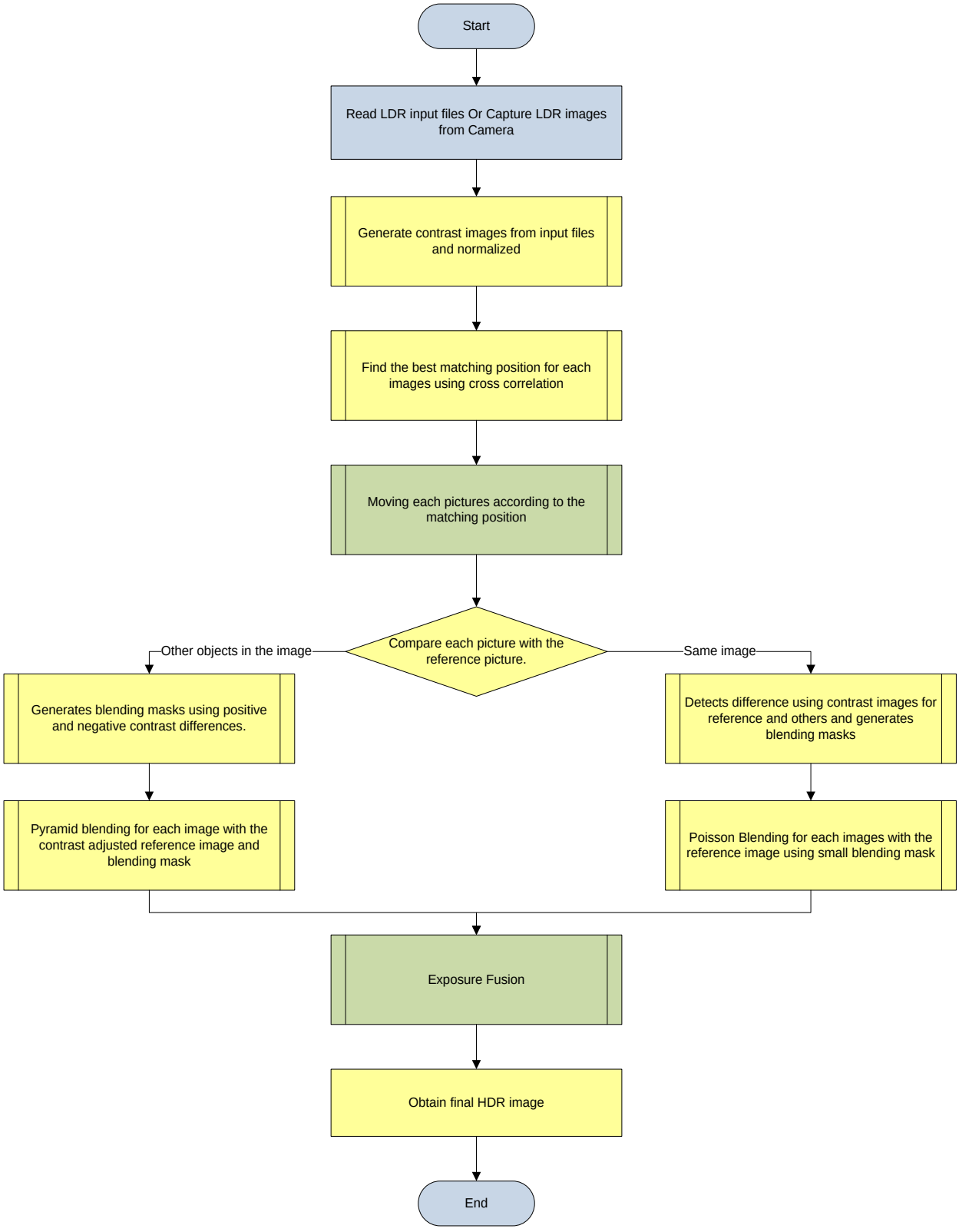


Figure 2. The flowchart of proposed method. The color of each box shows the different programming model; blue: Java, yellow: Native, Green: RenderScript

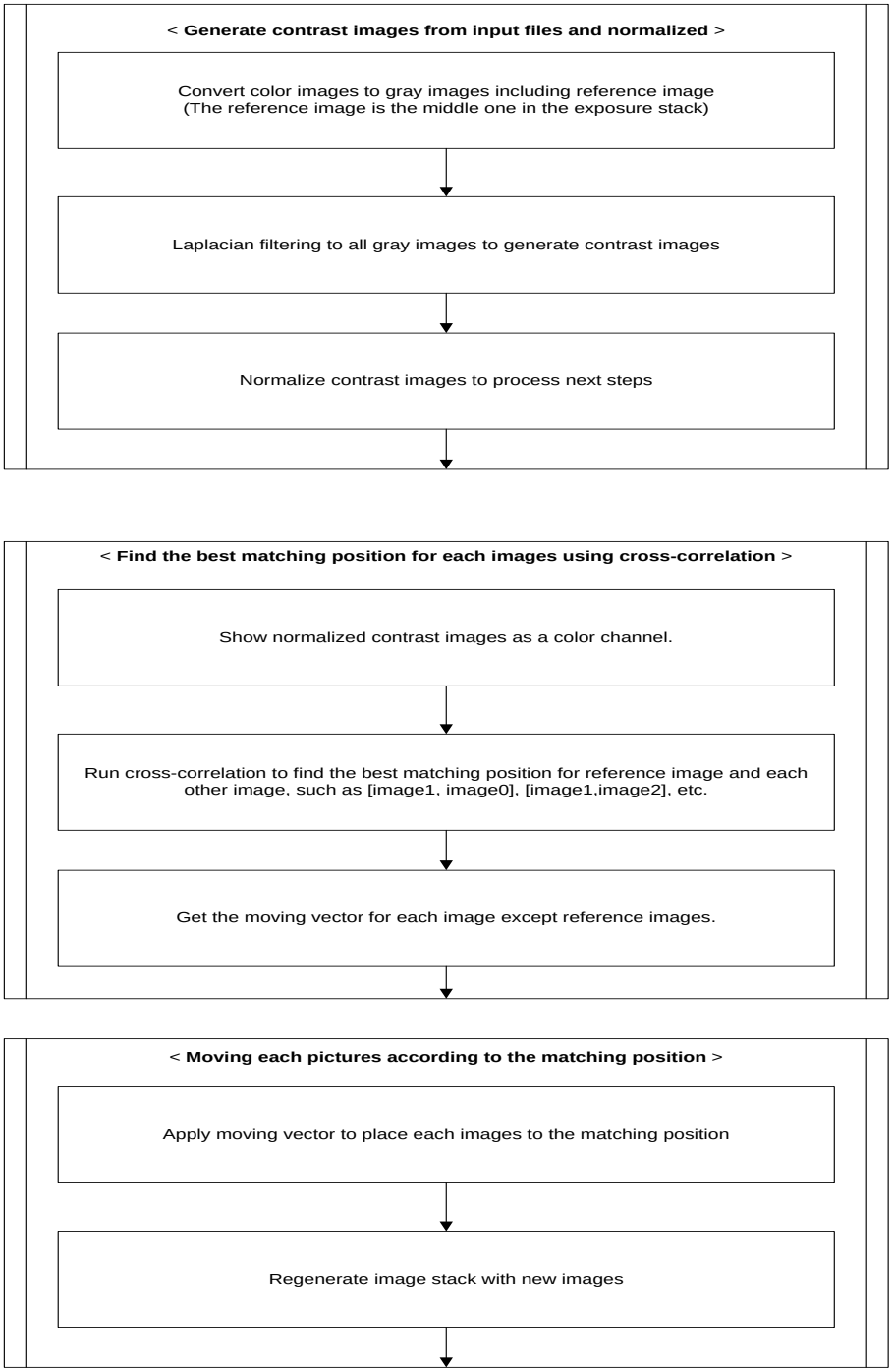


Fig. 3. The detailed flowchart of the first three steps after getting LDR images from Figure 2.

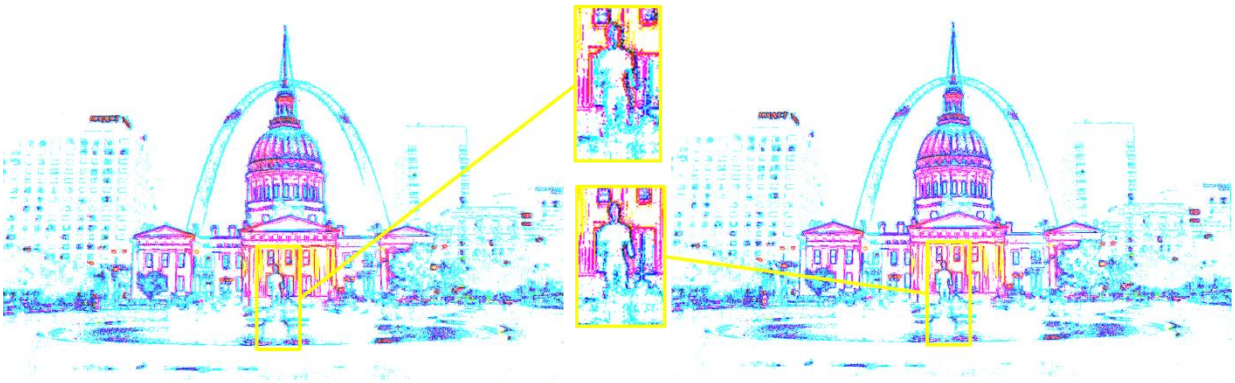


Fig. 4. A comparison between contrast images before and after alignment using the information from cross-correlation. Each color channel shows different LDR image in exposure stack. Left image shows misalignment. For the right image, the third and fourth images in the input sequence moved according to moving vectors found. The right image shows well aligned.

Ghosting artifacts are caused by moving objects in a scene found in HDR images. The last example (Figure 1,4), a static scene, showed that it is possible to produce an HDR image without major problems even if there is no deghosting procedure, which is a conditional component in my algorithm (Figure 2). With a new stack of aligned images, my algorithm compares each image with a reference image and finds if there are moving objects detected in the scene using normalized cross-correlation. There are two parameters in the formulas to check the similarity between two images. To calculate them, first the cross-correlation for position (0,0) is examined to get the normalized cross-correlation coefficient between the reference and aligned image. Then, the middle image in the stack is chosen as the reference, assuming it has a median exposure time and thus has fewer pixels with saturation problems than the other images.

It is expected that the exposure stack has balanced LDR images. If the normalized cross-correlation coefficients, R_a and R_b, are less than the thresholds, Th_a and Th_b, respectively, it indicates that the image selected is not well matched with the reference image and the chance to have moving objects in the scene is high. In this case, my algorithm takes the left path of the flowchart (Figure 2).

$$R_a = \sum_{x,y}\left(I_1(x,y) \cdot I_2(x,y)\right) / \sqrt{\sum_{x,y} I_1(x,y)^2 \cdot \sum_{x,y} I_1(x,y)^2} \quad (2)$$

$$R_b = \sum_{x,y}\left(I_1'(x,y) \cdot I_2'(x,y)\right) / \sqrt{\sum_{x,y} I_1'(x,y)^2 \cdot \sum_{x,y} I_1'(x,y)^2} \quad (3)$$

where

$$I_1'(i,j) = I_1'(i,j) - \frac{1}{width \cdot height} \cdot \sum_{x,y} I(x,y) \quad (4)$$

The left path of Figure 2 consists of two major operations, 1. generating blending masks using positive and negative contrast differences and 2. pyramid blending for each image with the contrast adjusted reference image and blending mask. The first operation includes three steps, which is focused on generating a blending mask for each image detected. The first step is generating contrast maps with positive and negative differences. In the second step, the map from the first step and the reference image are compared to create another map of positive and negative difference between them. For the third step, the positive difference values and (1 − negative difference) values are multiplied to generate a mask map. The final blending mask is created by comparing each pixel value of the mask map with the average pixel value of the mask map. The result applying this part of algorithm can be found in the result section (Figure 10).

2.2 Porting to Mobile Platform and Optimizing

Initially my algorithm was developed on a personal computer (PC) using software tools and libraries, such as Matlab and OpenCV libraries. With the goal to optimize the algorithm for mobile devices, all implementation needed to be ported on to a mobile platform. The OpenCV portion of implementation can be reused but Matlab code portion needed to be ported over OpenCV to use on mobile platform.

The Nexus 7 tablet, which runs the Android 4.4 operating system, was used as for the hardware platform, and the algorithm described in 2.1 was ported using OpenCV for Android using the Native Development Kit (NDK), which uses C/C++ through the Java Native Interface (JNI) for

better performance. To further increase the quality of performance, I incorporated the native interface for OpenCV library with ARM NEON optimization for ARMv7 and, furthermore, I also tested the heterogeneous computing model was using RenderScript [12]. Table 1 shows the available programming models on Android. RenderScript allows programmers to write kernels that automatically run in parallel on the hardware selected by the RenderScript runtime, a process not provided by OpenCV C/C++. Thus, my algorithm used the pthread library to become multithreaded for native interface to OpenCV in order to get the maximum performance for both GPUs and CPUs.

Development Env.	Language	OpenCV	Automatic Parallelism
SDK	Java	Yes	No
	RenderScript	No	Yes
NDK	C/C++	Yes	No
	OpenGL	No	Yes
	RenderScript	No	Yes

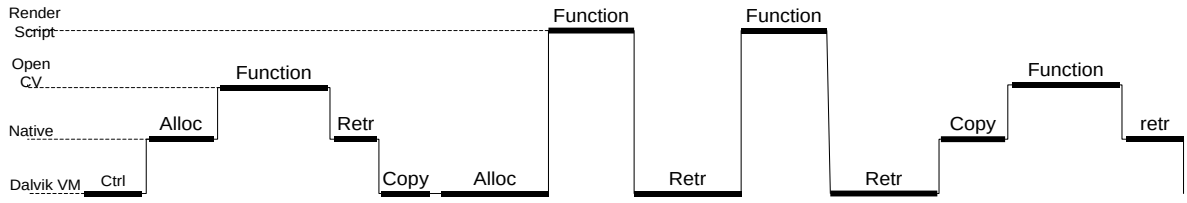
Tab 1. Programming Models on Android

To use RenderScript and OpenCV for Android SDK simultaneously in an application, OpenCV for Android SDK needs to be recompiled for use with STLport runtime. This is because the default release of OpenCV for Android SDK uses GNU STL runtime, while RenderScript uses STLport. Even though GNU STL runtime showed a better performance than STLport during testing, OpenCV for Android SDK needed to be rebuilt with STLport libraries because RenderScript does not support the version with GNU STL runtime due to licensing issues.

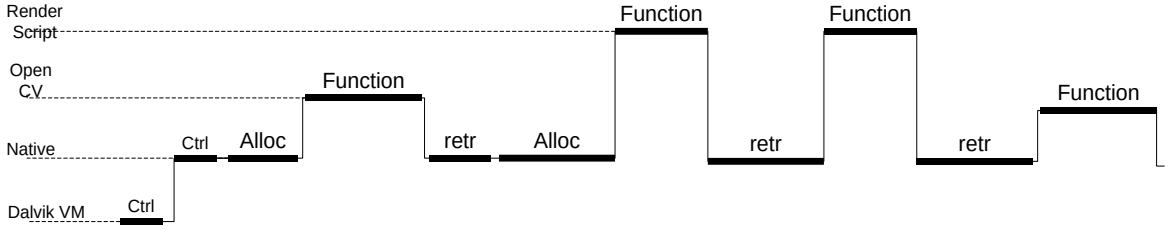
Once all implementations were correctly ported I started optimizing the code. The first native port suffered from the overhead of continuous transitions to different execution environments [11, 12]. R. Kemp et al [11] showed the possible overhead of transitions to different environments. My

algorithm minimized the number of transitions to maximize performance by reducing memory copies between CPU, GPU, and other programming domains. Although the camera capture routine could only be implemented in Java using Android SDK, all other interfaces, such as OpenCV library and RenderScript, are implemented in the native side using NDK.

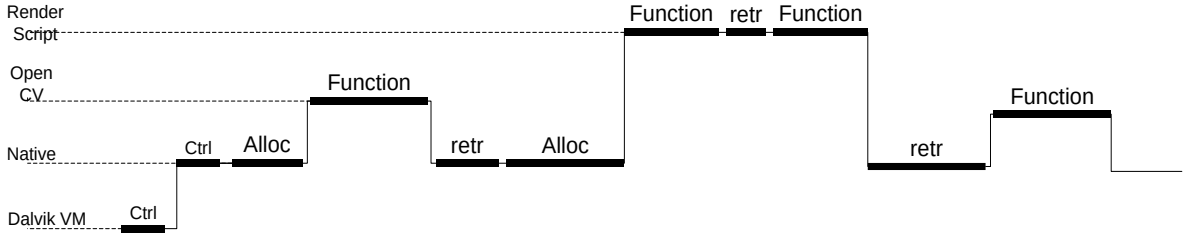
Figure 6 shows the schematic overview of the optimization process. To minimize the transition and memory copy, my algorithm evolved from Figure (a), (b), to (c). Comparing (a) to (b), there are fewer transitions between native and Java execution environments and improvement according to this change (Figure 6). Then, (c) shows even more improvement by adding control logic in the RenderScript interface, avoiding memory copies between CPU and GPU (Figure 6). OpenCL, which is the standard for heterogeneous computation but is not entirely available for Android at the moment, may provide other optional configurations to achieve better performance, like memory sharing feature by mapping [11]. However, it does reduce memory copies between GPU and CPU and reuses memory instead of retrieving the results. Since the OpenCV library does not allow automatic parallelism, I tested multithread interfaces in the Open CV library, and there was a significant improvement. Finally, my algorithm's schematic overview is shown by (d) (Figure 6). Figure 2 shows the final programming model with color indicating the execution environment: the portion using Java is blue, the Native portion is yellow, and the RenderScript is green. Using multithreading, allocation and retrieval in RenderScript can be run in parallel with other tasks as long as the output of those tasks is not needed immediately (Figure 6). Since most of the work done with RenderScript is asynchronous, it is also possible to run tasks on other threads simultaneously, contributing to the improvement.



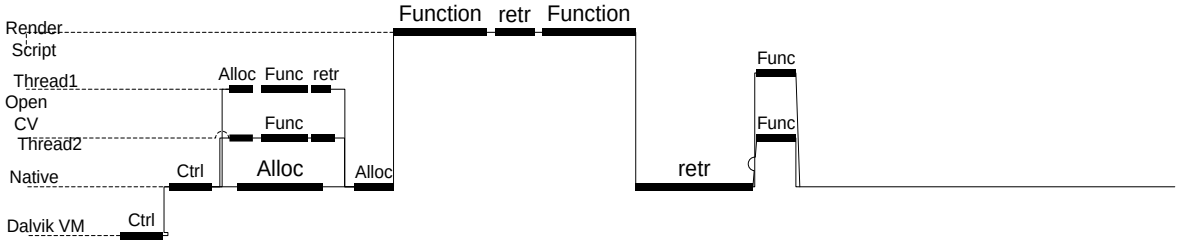
(a) Control logic and RenderScript interface in Java



(b) Control logic and RenderScript Interface in Native



(c) Control logic in Native and RenderScript and RenderScript Interface in Native



(d) Multithreaded Native interface with (c)

Fig. 6. Schematic overview of the optimization of the algorithm using OpenCV and RenderScript. Ctrl: control, Alloc: allocation, retr: retrieval of results. The vertical changes include context switching (JNI) overhead.

3. Results and Discussion

Multiple image stacks were used during testing, and to show the specific abilities of my algorithm, a few have been chosen as examples. Figure 7 compares the output HDR image without and with using my algorithm, left and right respectively, and the improvement is easily noticeable. The final result of my algorithm, exposure fusion over an aligned LDR image in an exposure stack, is shown in the second row of Figure 7.

Fig. 7. A comparison between the original method with artifacts to the left and with my algorithm to the right. The second row shows the result with images aligned using my algorithm.

Figure 8 illustrates an example of applying my algorithm to an exposure stack with a moving object. In this case, the center image of the first row is the reference image and is compared with the first and last image. The first image and the reference image match well and the normalized cross-correlation coefficients are larger than the thresholds. The last image and the reference image show a significant difference based on the normalized cross-correlation coefficient. The blending mask, the output of the step of "Generates blending mask using positive and negative contrast difference" (Figure 5), is shown in the third row of Figure 8. The last row of Figure 8 show results without and with my algorithm, left and right respectively.

Fig. 8. An exposure stack with moving objects. The first row shows the original exposures in the stack. The second row shows image after applying my algorithm with blending mask before exposure fusion. The third row shows the blending mask detected. The left of bottom shows the exposure fusion output without my algorithm; it shows the ghost of man in the right side of image. The bottom right image shows the result with my algorithm; the area where branches move severely show some improvement as well.

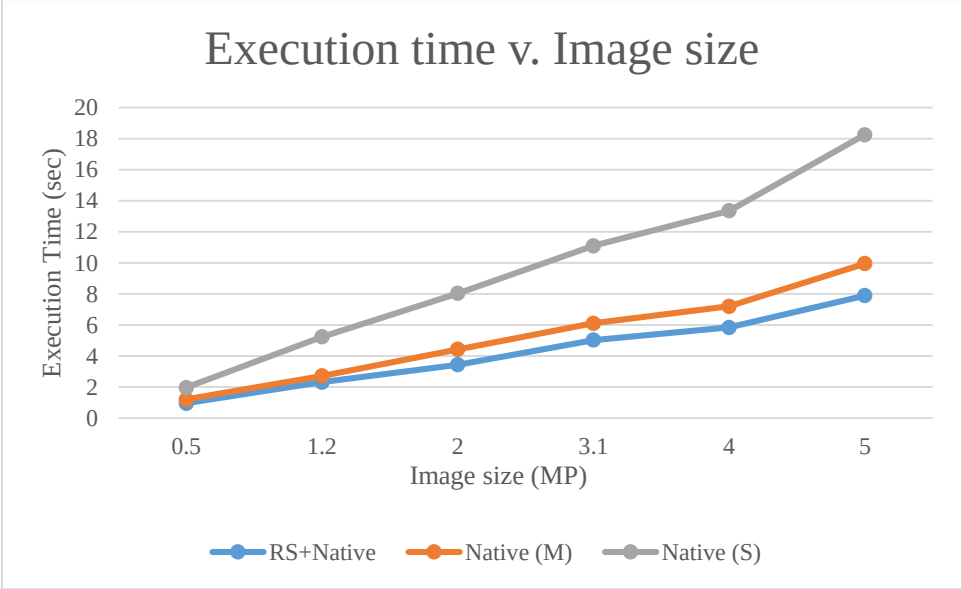


Fig. 9. Execution time comparison. The proposed method (RenderScript + Native) shows in blue line. The native implementation for multithreaded and single threaded shows in orange and gray lines respectively. Native (M) indicates native implementation using multithread and Native (S) indicates native implementation with a single thread.

As for the second phase, my algorithm was optimized on a mobile platform, Android. The method shown in Figure 6(d) is implemented using RenderScript and the multithreaded native interface. The test results for performance improvement is shown with execution time vs. different image sizes in Figure 9 The execution time has an almost linear relation with the image size for all

three implementations even though it shows different slopes. My algorithm in RenderScript and multithreaded native interface shows the best performance comparing to other implementation. Furthermore, RenderScript and multithreaded implementation does not achieve the expected performance improvement comparing to multithreaded native implementation without RenderScript, indicating that RenderScript runtime introduces overhead even with multithreading.

The proposed implementation using RenderScript and a multithreaded native interface improves the execution time by using multiple cores, including GPUs. As I described in 2.2, there may be further improvements in performance with more control logic in RenderScript and fewer transitions to the execution environment. Signal processing has potential in performing better in the native environment, using CPUs more than GPUs when it includes more than a few sequential logics. Even though my algorithm has left this portion of code in native implementation, there must be a way to implement it in RenderScript without affecting the performance.

4. Conclusion and Future Work

This paper proposes an enhanced approach of HDR imaging that can identify and remove blurring and ghosting artifacts and that has been optimized to be implemented on a mobile device within reasonable execution time and complexity constraints. HDR imaging has been not practical for use on handheld devices because of the assumption that the scene and objects within it are perfectly static, and although current methods do account for some motion, they try to tackle both types of artifacts in one step, and also the large amount of data involved prevents use on a mobile device even further. To address these challenges, this research proposes a multi-stage approach that first accounts for blurring artifacts and then conditionally detects and removes ghosting artifacts. The advantage from the first phase of the proposed algorithm is that, because it uses contrast difference maps and masks from those maps instead of the original images, it is less

complex comparing to other approaches that take few minutes to process the data but actually more effective than existing methods.

In the second phase, this paper shows how to optimize the proposed algorithm on the mobile platform using heterogeneous computation environment. This step demonstrates that the proposed algorithm is practical and efficient in a mobile environment, while other methods take more than few ten seconds to a few minutes to complete the same task. By using RenderScript and multithreading, multiple computational operations run simultaneously on CPUs and GPUs to maximize the performance of the system. Finally, the algorithm was implemented and tested on an actual device to show how HDR imaging can be practical, like a tablet, addressing common human errors like camera shake or moving objects without sacrificing the efficiency and quality necessary for a mobile device.

References

[1] P. Debevec and J. Malik, "Recovering High Dynamic Range Radiance Maps from Photographs," in SIGGRAPH, 1997.
[2] E. Reinhard, G. Ward, S. Pattanaik, and P. Debevec, "High dynamic range imaging: acquisition, display, and image-based lighting", Morgan Kaufmann, 2006.
[3] T. Mertens, J. Kautz, and F. Van Reeth, "Exposure fusion," Pacific Conf. on Computer Graphics and Applications, pp. 382–390, Jan 2007.
[4] J. Hu, O. Gallo, K. Pulli, and X. Sun, **"**HDR Deghosting: How to Deal with Saturation?" Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, June 2013.
[5] J. Hu, O. Gallo, and K. Pulli, "Exposure stacks of live scenes with hand-held cameras". In ECCV, 2012
[6] H. Zimmer, A. Bruhn, and J.Weickert. "Freehand HDR imaging of moving scenes with simultaneous resolution enhancement". Computer Graphics Forum, 2011.
[7] O. Gallo, N. Gelfand, W. Chen, M. Tico, and K. Pulli. "Artifact-free high dynamic range imaging". In ICCP, 2009.
[8] N. Gelfand, A. Adams, S. Park, and K Pulli, "Multi-exposure Imaging on Mobile Devices". Proceedings of the international conference on Multimedia, 2010.
[9] G. Salvi, P. Sharma, and S. Raman, "Efficient Image Retargeting for High Dynamic Range Scenes". arXiv:1305.4544 [cs.CV], 2013.
[10] M. Tico, N. Gelfand, and K. Pulli, "Motion-blur-free exposure fusion," in 17th IEEE International Conference on Image Processing, pp. 3321-3324, September 2010.
[11] R. Kemp, N. Palmer, T.Kielmann, and H. Bal. "Using RenderScript and RCUDA for Compute Intensive tasks on Mobile Devices: a Case Study". AM Ghuloum Software Engineering (Workshops), 305-318, 2013.

[12] R. Membarth, O. Reiche, F. Hannig, and J. Teich, “Code Generation for Embedded Heterogeneous Architectures on Android”. Design, Automation and Test in Europe Conference and Exhibition (DATE), 2014.
[13] High-dynamic-range imaging, Wikipedia http://en.wikipedia.org/wiki/High-dynamic-range_imaging
[14] K. Park, D. Park and Y. Ha, “High Dynamic Range Image Acquisition from Multiple Low Dynamic Range Images Based on Estimation of Scene Dynamic Range”. Journal of Imaging Science and Technology, vol. 53, no. 2, pp. 020505-1 - 020505-12, Mar. /Apr. 2009
[15] Sample HDR photos with easyHDR, http://www.easyhdr.com/examples.php
[16] Using the Histogram to Ensure you have covered the Dynamic Range of a Scene, http://thehdrimage.com/using-the-histogram-to-ensure-you-have-covered-the-dynamic-range-of-a-scene/