High Dynamic Range Imaging with the Android Platform

Group 8: Johan Mathe, Tim M. Wong
EE368, Stanford University

Abstract—**High Dynamic Range imaging is an technique allowing to encode a greater dynamic range of luminance in the same picture. Here we present an implementation of HDR imaging generation that works by taking different pictures with different exposure settings, aligning them, merging them into one single picure, and finally doing a tone mapping operation in order to display the HDR image on a LDR device, as the Android phone screen. This report illustrates our approach to bring HDR photography on the Android phone.**

I. Introduction

High dynamic range imaging is becoming increasingly popular in the area of amateur photography. It has interestingly not been truly implemented yet in the android platform. Here we show how we implemented such a technique on the android phone. We show here the different steps of our approach: first, since we are merging multiple pictures alltogether, we implemented an image alignment algorithm. Then we show how we implemented the high dynamic range merging, and finally we describe a tone mapping operation based on a global tone mapping operator. We will also inspect some technical details related to the tradeoff we made in order to get the best experience as possible on the handset.

II. Prior and related work

A. On Image alignment

There are Various Alignment Approaches:

1) Correlation base: This is a well developed technique for image matching, very simple to implement and can get sub pixel accuracy. However, it is very CPU intensive especially when the search range is large. Moreover, it is very sensitive to image rotation. To speed things up people often use sub-sampling to get rough alignment then before doing alignment at full resolution. Segmentation technique is often used to overcome small rotation and/or image distortion.
2) Gradient based: The gradient base method is very good at matching moving object, but it is very sensitive to contrast change, not very suitable for HDR images
3) Key points based. SIFT [7]and SURF [8] are often use for feature matching. However, they are very CPU insensitive and it is very difficult to get single pixel accuracy for HDR images.
4) Edge detection: Apply edge detection technique to the images, then compare the edges. However, for HDR image, the edge position will often shift as the contrast change so it might not get the single pixel accuracy we are looking for.

Recently, Ward[9] proposed to binarize the images use it's median gray level and they argue that the threshold location will not shift significantly due to image contrast change. They create a pyramid of median threshold bitmap (MTB), they start off with aligning the images at the bottom of the inverted pyramid. Bitwise XOR is performed between the reference image and the image under test to calculate the image offset. And use the alignment offset as the base for the next set of images up in the pyramid. This process is repeated until the top of the pyramid is reached.

In their paper they assumed that the images have linear XY translation only and no rotation between images. We found that this is not the case for the DROID phone since the phone is light and pressing the shutter button on the phone can cause the phone to rotate. We have developed a segmentation technique that takes care of small rotation and make modification to their algorithm to speed it up further.

B. On HDR

Earliest work on High Dynamic Range imaging have been found in the middle of the 19th century. In 1850, Gustave Le Gray pioneered the idea by merging different pictures altogether to end up with better dynamic range pictures. Modern HDR imaging has its roots in the 1980 and early 90's, when people started developping tone mapping ideas. HDR became increasingly popular when Paul Devebec presented a new method for merging different exposure pictures to a single HDR image[1]. The litterature is also very rich in terms of tone mapping algorithms. It goes from Global Tone Mapping operators, very efficient in terms of computation, but with a result being slighlty less impressive and less contrasted than Local Tone Mapping operators, like the gradient domain tone mapping operator [4].

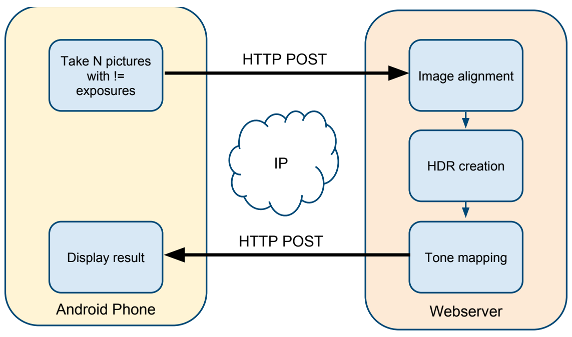
There are a couple of sotfware abailable for creating HDR pictures on PC/Mac. We focused our attention on an open source software/libray called psftools and pfscalibration[5]. This software indeed implements most of the common merging and tone mapping techniques that have been pusblished in the past 15 years.

C. On Tone Mapping

We investigated a couple of tone mapping algorithms. We can divide tone mapping algorithms in two main families: global tone mapping operators, as seen in [2] that have the same operator for every single pixel in the image, and local tone mapping operators, that use various locality properties, as the gradient algorithm, seen in [4]. Local tone mapping operators usually give more local contrasts on the image, but can produce halo effects. They are also generally fairly slower than global tone mapping operators.

III. Description the system and algorithms

The software stack of the project is a client-server application written in java on both the client side (android phone) and the server side (web server). During our experiements, we first started with a monolithic version on exclusively the android phone, but since the HDR and the alignment computation mostly happens in floating point arithmetics, we had to move the algorithm code outside of the phone (the ARM processor doesn't have any floating point extensions). It would have been possible to migrate all the code from floating point arithmetics to fixed point arithmetics, but this work would have been out of the project scope. All the steps are outlined in the algorithm pipeline picture, and are described in more details in the following sections.



Full algorithm pipeline

A. Picture taking

In order to obtain the final HDR image from standard LDR pictures, we had to take multiple pictures with multiple exposures. The android platform allows the developper to change the exposure setting since the version 2.2[1], which came out during the project. The API allows us to change the exposure value from -2EV to 2EV, with a 0.5 granularity[2]. Our application takes 4 photos, from -2EV to 1EV (-2, -1, 0, 1). We decided that 4 pictures was a good tradeoff in between the waiting time of the images upload and the amount of pictures needed to get a proper HDR picture.

B. Upload

Once images are taken and written to the sdcard of the phone, they are uploaded via a multipart[3] POST data CGI form to the Java webserver.

[1]Android 2.2 is also called FroYo

[2]These values/tests have been performed on the Nexus One phone.

[3]See RFC 2388: http://www.ietf.org/rfc/rfc2388.txt

C. Alignment

Since we are taking a series of images in a quick sequence, we are making the following assumptions:

- No FOV change i.e. no scale change
- Small delta X and Y change between 2 images so that lens distortion doesn't need to be taken into consideration
- No moving objects, currently no algo we are aware of can handle moving object very well.
- No significance motion blur.
- Rotation between images is small.
- No change in projection

We divide the images into rectangular segments for segmentation. Very often images contain large area where that has no feature at all. For example in a lot landscape pictures, the upper half of the image is blue sky and contain very little information for image alignment, image alignment on that segment might fail and worse still produce the wrong offset. To fix this we need to find a way to figure out which segment contains enough image features for alignment.

We have looked into using Harris corners and SURF keypoint detection technique to figure out which segments contain enough information for image alignment. Since we are not relying on the keypoints for image alignment, we can simply the SURF algorithm to down sample by a factor of 4 to 8 and do one or two octives only, furthermore, we don't need to do any key point descriptor. We found that this simplify SURF key point detection is even faster than the Harris corner detection in our implement.

1) Alignment Details: For image alignment, we found that it is important to have good image contrast but not too saturated image as the reference, this also improve the keypoints detection and help us to identify the right region for segmentation. On each image candidate we perform histogram on a sub-sampled image and calculate the 5 percentile low end point (LEP) and high end point (HEP) and choose the image with highest separation between HEP and LEP as the reference image.

2) MTB alignment : The MTB image alignment is a successful approximation algorithm, both the image under test and the reference images are initially sub sampled by a factor of 2N, the down sample reference is then compared with the down sampled image under test and the 8 images shifted by one pixel in all direction and compute the offset position that produce the least alignment error. For next iteration, we reduce the down sampling of the images by a factor of 2, and offset the image under test by 2x the offset calculated from the previous iteration. Repeat this process until the images are at full resolution.

Initially the reference image and the image for alignment are both down sample for a factor of 2N. The max detectable offset between the reference image and test image is therefore 2N+1-1. Since the edges of the image normally doesn't contain that much information we exclude the edge so that the center region for test is dividable of N so that we can increase the resolution by simple factor of 2 per iteration.

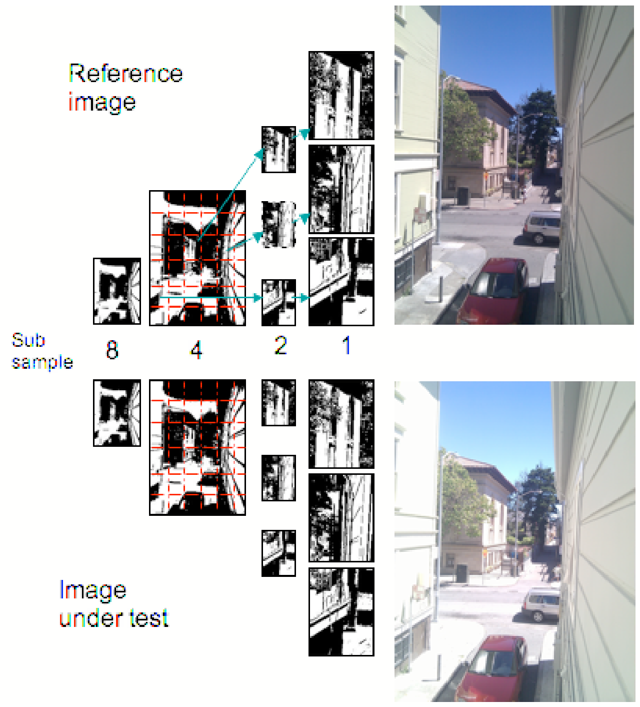


Illustration of the alignment algorithm.

To make the matching algorithm less sensitive to noise, all pixels that are within some threshold value from the median value are masked out and two binary mask images are created. In Ward's paper, they perform bitwise XOR on the two MTB images, as well as AND operation on the mask images. They then go through all the pixels and summing up all the pixels that the result of XOR is 1 and the result of the AND operation is 0. This represents the level of mismatch between the two images. This involves 4 read operations and 3 logic operations per pixel. In our implementation we encode the binary value to bit 0 and the mask to bit 2 on one data byte and perform sum operation on the reference pixel and image pixel.

0	0	0	0	0	1	0	1
					Mask bit		Data bit

Reading bit 1 of the sum data is equivalent to XOR operation on the data bit and reading bit 3 is equivalent to AND operation on the mask. In this operation, we are doing 2 read operations, 1 fixed point addition and two logic operation per pixel. That doesn't seem like a lot, but when one has 5M pixels per image and 9 such operation per image pair per iteration, the difference is small but noticeable.

To improve the matching location accuracy, we employ parabola interoperation which is commonly used in correlation based pattern matching functions.

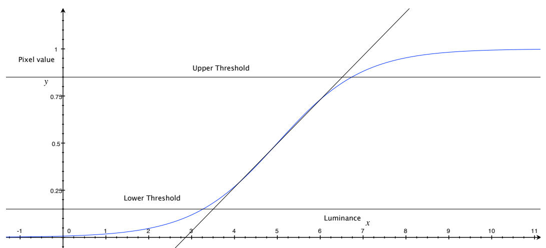
3) Segmentation : We are performing SURF key point detection on the reference image only, we found that doing it on the reference image or the image under test doesn't make any real different. Since we only need three points to measure the image rotation and translation correction, therefore, the three segments that has highest concentration of key points is used for segmentation. This cut down the image alignment time.

Very often the keypoints are concentrated in on a single region. So the three segments for alignment are very close together. This is not the best case for angle alignment, so refine the segment selection algorithm such than no two segments can be next to each other. We found that for large segment size, alignment stable is good but it is sensitive to image rotation. Smaller segment size is less sensitive to rotation and smaller compute time but more sensitive to noise. We found that dividing the image into 6x6 segments product the best result. For the first 2-3 iterations, when down sampling is large, we are not doing any segmentation, since we don't have that many pixels available for alignment to start with, segmentation will make the alignment less robust. We start to do segmentation when down sample is at 8 so max offset between segment is 32 (8 x 4). This scheme, for the worst case scenario, is capable of detecting 0.8 degree of image rotation (This is when the segments than contain the highest number of keypoints are at extreme ends)

D. Image merge

A lot of HDR merge algorithms work by estimating the luminance/pixel value response curve of the camera, and then merging the pixels regarding this response curve, as seen in [1]. This method gives very good results, but is very time consuming.

It happens that there is another much simpler method that works by assuming that the camera response curve is linear between two pixel values[3]. We discovered this method by noticing that matlab HDR merging function was very vast, and by looking at their references on their implementation[3].



Luminance/pixel value

By doing this approximation, it becomes easy to merge the images. Interestingly, for the case of the android phone, we did not need to do any gamma correction. If we consider E_i being the exposure of the ith image, we just need to merge all pixel values between two thresholds by weighting them with $\frac{1}{E_i}$. There is one big caveat in this method: we are not guaranteed to get all the pixel values if they all fall in some over-exposed or under-exposed areas. In that case, we just have to take the maximum and minimum pixel value accross all the pixels in the image for over and under-exposed pixels, respectively. This is done for each color channel (Red Green and Blue in this case). If we consider the V_i being the HDR output pixel i value and the P_{ij} being the ith pixel value for the jth input image, and N_i the number of images with correct exposures at this particular pixel, we can write:

$$V_i = \frac{1}{N_i} \sum_{j=1}^{N_i} \frac{P_{ij}}{E_j}$$

The algorithm for image merge is the following:

```
inputImages = Read(filePaths)
S = size(inputImage[0])
outputImage initOutputImage(S)
underExposed = initBooleanMap(S)
overExposed = initBooleanMap(S)
properlyExposed = initBooleanMap(S)
properlyExposedCount = initIntMap(S)

for i in size(outputImage):
  for im in inputImages:
    if IsOverExposed(im[i]))
      overExposed[i] |= true
    else if IsUnderExposed(im[i])) {
      underExposed[i] |= true
    else
      properlyExposed[i] |= true
      properlyExposedCount[i] += 1
      outputImage[i] += im/GetImageExposure(im)
    end
  end
end
for i in size(outputImage):
  outputImage[i] /= max(1, properlyExposedCount
      [i]);
  if !properlyExposed[i]:
    if underExposed[i] and !overExposed[i]:
      outputImage[i] = MinProperlyExposed(i)
    else if overExposed[i] and !underExposed[i
        ]:
      outputImage[i] = MaxProperlyExposed(i)
    else if overExposed[i] and underExposed[i]
      outputImage[i] = FindNeighborValue(
          outputImage)
    end
  end
end
```

merge.m

All the pixel values are then encoded in floating point values in a Java data structure derived form the base Image class of the JJIL library.

E. Tone mapping

In order to represent the HDR picture on a Low Dynamic Range display, we need to operate a tone-mapping operation. We decided to take a global tone mapping operator, mostly because global operators are less time consuming than other local tone mapping operators. We took the method from Reinhard and al as seen in [2]. The steps are fairly straightforward: we first compute the average log luminance by the formula (1). Here the δ prevents numerical instability for small values of L_w. Typical values ofδ are close to 10^{-4}. Here L stands for the Luminance channel of our picture. The tonemapping algorithm doesn't touch any other channel.

$$\bar{L}_w = \exp\left(\frac{1}{N}\sum_{x,y}\log(\delta + L_w(x,y))\right) \tag{1}$$

Then we normalize all the pixels of the image with this log luminance. In the following formula, a is a degree of freedom in the algorithm, it helps setting the brightness of the resulting image.

$$L(x,y) = \frac{a}{\bar{L}_w} L_w(x,y) \tag{2}$$

Finally, we operate a normalization of the luminosity between 0 and 1 with the following:

$$L_d(x,y) = \frac{L(x,y)}{1 + L(x,y)} \tag{3}$$

As we just saw here, we need to operate on the Luminance channel of the picture. In order to achieve this, we need to convert the original image from the RGB colorspace to another colorspace luminance-based. We chose the xyY colorspace. After some experimentations, it showed the biggest robustness in keeping various images contrasts. Once the luminance is remapped, we just need to convert the colors back to the original RGB colorspace.

IV. Experimental results

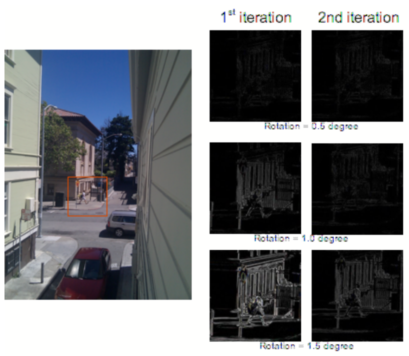
A. Image alignment

We first tested our algorithms on the phone itself, but the alignment algorithm and the merging algorithm were taking both more than 2 minutes, mostly because of the lack of a floating point unit on the ARM processors. Indeed a single division in floating point arithmetics can take up to 1ms. We can see the results taken on the android phone in the timings table. Uploading 4 images that weight around 1MB on a good wireless connection takes less than 5 seconds on the nexus one.

We found that the most time consuming part of the image alignment algorithm is the creation of keypoints for segmentation. On the AMD 2.8GHz x3 desktop computer it takes about 300ms to compute the keypoint for 5M pixel image, while it takes only 100ms to perform the MTB image alignment. There is clearly room for improvement. However, the image alignment time is short only because the keypoint detection allows us to use 3 segments out of 36 segments. If we were to use brute force approach to figure out which segments are good for image alignment, it will make much longer time. Furthermore, we are doing one keypoint detection on the reference image only; the 300ms calculation time is acceptable. We found that our alignment algorithm is pretty robust, but when the images are rotated or when the image is very noise, our algorithm might not work as good as we wish. We found that for some cases running the alignment algorithm twice can make the alignment works much better. Another possibility is to use SURF keypoint detection for coarse alignment, then we align the prealigned images. Our scheme for doing segmentation is pretty simple. We can think of a lot of ways to improve the segmentation:

- Since lens distortion normally happens near the corners, and the key features of the image is normally near to the center of the frame. We might want to weight the center segments and weight down the corner segment when we select the segment.
- We are doing 3 segments only for affine matrix calculation. One might want to do alignment on more segments and use least square fitting to get a better affine matrix. Once we got the affine matrix we can use this matrix to transform all the segment and figure out whether any of them is a outliner and automatically reject it and recalculate the affine matrix again
- Auto adjust the size of the alignment region base of the distribution of the keypoints.

To evaluate the effectiveness of our image alignment against small image rotation we use Matlab to rotate one of the image by 0.5, 1.0 and 1.5 degrees and perform alignment with the original image. The following figure shows the residual error map by subtracting the aligned image with the original image. For 0.5 degree rotation, the algorithm is doing a pretty good job of alignment and the residual size is very small and almost invisible. For 1 degree rotation, we start to see some structure in the residual signal and at 1.5 the residual is getting larger. If we apply the alignment algorithm to the aligned images, then the residual error is getting smaller.



Results analysis of the alignment.

B. HDR

There are 2 main degrees of freedom we can adjust for the HDR merge, which are the threshold for high and low pixel values that are considered as over and under exposed, respectively. When doing some tests with the usual set of HDR pictures, like the Stanford memorial church, these thresholds needed to be fairly far from the extremum values. In the case of the android phone camera, more particularly the nexus one, we noticed that we can raise these threshold, which means that the response curve of the android phone camera is linear for a very big range of pixel values. This is very coherent to the fact that digital sensors have a much more linear behavior than regular films.

C. Tone mapping

The only degree of freedom that we have in our tone mapping algorithm is the brightness setting. We found that a pretty good value to be 0.6, after a trial/errors method.

For both HDR and tone mapping algorithms, we tried both float and double arithmetics. We saw that the memory footprint resulting in using double values was too important, which is why we stayed in float data values.

D. Timing results

Here are timing measurements that show the difference between software processing on the phone and on a regular desktop PC.

	Nexus one (s)	2.6Ghz PC (s)
Image alignment	120	0.250
HDR Merging	126	0.731
Tone mapping	112	0.544

E. Picture tests

We did try a couple of cases to check how good our algorithm performs regarding the type of scene

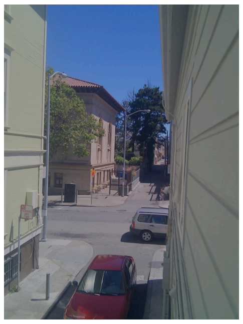
- If the image is noisy or blury (night, indoor scene), both the alignment and the HDR algorithm do not perform very good
- For an outdoor scene, without moving parts, we got excellent results, from both the alignment and the HDR part.

Image 1: EV = -2

Image 2: EV = -1

Image 2: EV = 0

Image 2: EV = 1


Result of the whole algo pipeline

V. Conclusions

We implemented a full client/server HDR processing stack for the android platform that works very well for daylight pictures. Our tests results good satisfactory results for both HDR merging and image alignment. Our experiments showed that due to the floating point arithmetics, it is impraticable to implement the alignment/HDR merge/tonemapping algorithms on the phone, but there is room for improvement by converting all the algorithm to fixed point arithmetics. In future work, we can also think about making more tone mapping operators available to the user.

VI. Bibliography

References

[1] Paul E. Debevec, Jitendra Malik, Recovering High Dynamic Range Radiance Maps from Photographs, SIGGRAPH 1997.

[2] Erik Reinhard, Michael Stark, Peter Shirley, Jim Ferwerda, Photographics Tone Reproduction for Digital Images, SIGGRAPH 2002.

[3] Reinhard, et al. "High Dynamic Range Imaging." 2006. Ch. 4.

[4] Raanan Fattal, Dani Lischinski, Michael Werman, Gradient Domain High Dynamic Range Compression, SIGGRAPH 2002.

[5] http://pfstools.sourceforge.net/

[6] Herbert Bay, Andreas Ess, Tinne Tuytelaars, Luc Van Gool, "SURF: Speeded Up Robust Features", Computer Vision and Image Understanding (CVIU), Vol. 110, No. 3, pp. 346–359, 2008

[7] SIFT algo David G. Lowe, "Object recognition from local scale-invariant features," International Conference on Computer Vision, Corfu, Greece (September 1999), pp. 1150-1157

[8] A. Tomaszewska and R. Mantiuk, "Image Registration for Multi-Exposure High Dynamic Range Image Acquisition," Proc. Int'l Conf. Central Europe on Computer Graphics, Visualization, and Computer Vision (WSCG), 2007

[9] G. Ward, "Fast, Robust Image Registration for Compositing High Dynamic Range Photographs from Handheld Exposures," J. Graphics Tools, vol. 8, no. 2, 2004, pp. 17–30.

[10] B. Girod, Lecture Notes for EE 368: Digital Image Processing, Spring 2010.

VII. Project Log

The poster and the report have been done together.

- Johan Mathe: Image merge, tone mapping, client-server stack, android app
- Tim M Wong: Image alignment, android app