HDR for Android

Dane Wallinga (dgw25), Jeffrey Davidson (jpd236)

Figure 1: Artistic HDR reconstruction from mobile phone pictures

The process of photography involves mapping radiances from a scene into pixel values in a 2D image. Naturally, the intensity of each pixel is related to the radiance in the scene - bright portions of the scene correspond with bright portions in the image. However, the map from scene radiance to image intensity is generally non-linear and dependent on the intrinsics of the camera. Generally, this map is also unknown except to the camera manufacturer.

Cameras are also limited by their dynamic range - the range of radiances that they can capture in one image. While humans can perceive vast differences in dynamic range in a scene, a camera can only capture a small subset of this range in any given image. Although it is possible to alter the exposure time of the camera, which effectively

captures a different range of radiances, it is impossible to take one image with the camera capturing the broad range of radiances that may be present in a scene.

However, by taking multiple pictures of a given scene under different exposures, it becomes possible to determine the original radiance of the scene from the images. This requires approximating the unknown radiance map from the images, and then applying the inverse of this map to each pixel in the image to determine the radiance of the scene at that point across a broader dynamic range.

This process leads to two natural problems - what happens if the content between the differently exposed pictures changes, and how do we make use of the reconstructed scene radiance to create better-looking images? The first problem is addressed by taking the pictures in rapid succession, and by using a technique called *median threshold bitmaps* to align the images. For the second problem, we use *tone mapping* techniques to compress the full range of radiances into an image, but across a wider dynamic range than the original camera could capture.

An added wrinkle in the problem arises when we attempt to apply this technique to pictures taken with a mobile phone camera. Mobile phones are hampered by lower-quality cameras and limited processing power and memory. Although we cannot directly address the quality problem, many of the pictures we calculated were improvements over the originals. To solve the limited processing power problem, we made use of an external server which performed the image manipulation and sent results back to the mobile phone.

Related Work

As an initial reference, we used an HDR assignment from a different Computer Vision course offered at the University of Wisconsin, CS 766. Our HDR application made use of the following four algorithms, which we discovered from this assignment page:

- Recovering High Dynamic Range Radiance Maps from Paragraphs, Debevec '97. This paper describes a relatively simple least-squares technique for reconstructing

the radiance map and then using this map to determine scene radiances. Although other techniques for recovering these maps exist, e.g. Robertson '03, Debevec's technique is both simple and most commonly used.

- Fast, Robust Image Registration for Compositing High Dynamic Range Photographs from Handheld Exposures, Ward '03. This paper describes a technique for aligning images that differ by a small translation. Though we have done image alignment using feature matching, edge-based features tend not to work well across different exposures. The alternative described in this paper uses median threshold bitmaps to align the images.

- A Perceptual Framework for Contrast Processing of High Dynamic Range Images, Mantiuk '06, and Display Adaptive Tone Mapping, Mantiuk '08. These papers contain two different techniques for mapping scene radiances to image intensities. We used the former technique to create artistic images, and the latter technique for realistic images. Although the focus of the latter technique was intended for different mappings for different display types, we found that the images created by this mapping operator were generally better quality than those created by other mappers listed here.

Technical Description

HDR for Android works in two phases. Phase one, which occurs on the phone, results in the pictures necessary for performing HDR calculations. Using the Android Camera API, we wrote a Java application that captured three different pictures. We set the exposure compensation on each picture to the minimum possible, zero, and the maximum possible, resulting in very dark, standard, and very light images. From the exposure compensation values, we determined approximate exposure times which would result in those images using the equation $EV = \log_2 \frac{N^2}{t}$, where EV is the negative of the exposure compensation, N is the relative aperture size, and t is the exposure time.

Assuming that the aperture size remains constant, this equation gives us the relationship between the exposure compensation we set and the resulting exposure time. We then compress the three pictures and send them, along with the determined exposure times for each picture, to a server for additional processing.

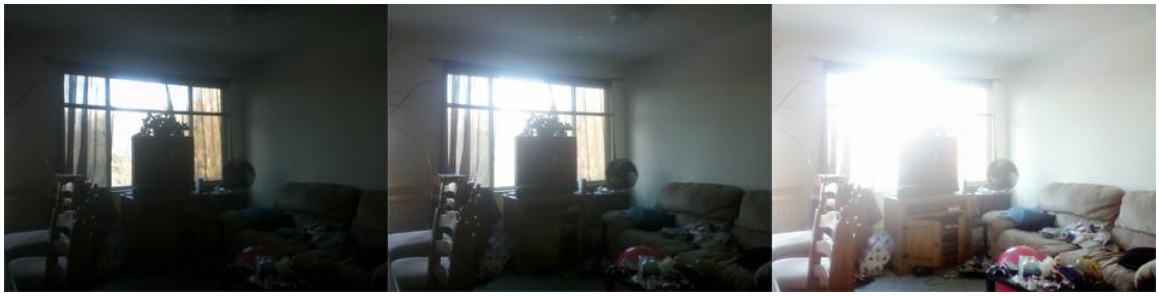

Figure 2: Three different exposures of the same scene

In phase two, we construct regular and artistic HDR images from the differently exposed inputs. This phase is performed on an Ubuntu Linux server with significantly more processing power and memory than is available on mobile phones. A PHP script drives this phase, sending and receiving the images and executing the alignment, radiance map construction, and tone mapping algorithms.

We first run our Java implementation of Ward '03 to align the input images. This algorithm makes the simplifying assumption that the only transformations that have occurred between the images are translations. This assumption tends to apply well in practice when aligning images captured by a handheld camera. The algorithm works by constructing median threshold bitmaps for each of the images - these are binary images whose pixels are 1 if the corresponding pixel in the grayscale version of the image is brighter than the median intensity, or 0 otherwise. It then slides this bitmap across a small window of possible translations, testing similarity by XORing the set of translated images together.

This process is accelerated by creating a pyramid of such binary images, and restricting translations to a single pixel at each level. For instance, if the smallest images in the pyramid are best matched by a translation of a single pixel to the left, matching at the next level of the pyramid, in which the images are twice as large, begins with a

default shift of 2 pixels to the left. In essence, matching at each pyramid level determines a single bit of the translation amount.

It is necessary to cap the number of tested translations to complete alignment in a reasonable period of time; however, we found that if the user takes pictures while trying to hold the camera steady without any support, the resulting translations could be fairly significant. Thus, we intend for the alignment algorithm to rectify slight transformations that may occur when the camera is supported but shifts slightly between pictures, and not for vastly different images that require translations of more than 16 pixels to be rectified. This part takes on the order of ten seconds to complete on the server.

Once we have aligned the images, we use our Java implementation of Debevec '97 to reconstruct the camera's radiance map and the scene radiance. This approach involves the fundamental equation $g(Z_{ij}) = \ln(E_i) + \ln(\Delta t_j)$, where $g = \ln f^{-1}$ is the log of the inverse of f, the map from scene exposure to pixel values, z_{ij} is the pixel value at location i and image j, E_i is the irradiance of the scene at location i, which we assume is constant across the different images, and Δt_j is the exposure time of image j. Given the pixel values and exposure times, we need to recover the radiance map and the irradiance values that best satisfy this equation, subject to additional smoothness constraints. This can be done by randomly sampling points in the scene and finding the least squares solution optimizing the error in the above equation.

From the radiance map, we take a weighted average of the inverse of this map across the differently exposed pictures to approximate the radiance of that point in the scene. Finally, we output the scene radiances in the Radiance RGBE .hdr file format. This format stores color values as 24-bit RGB values in addition to an 8-bit shared exponent, which allows storing values beyond the traditional 0-255 range in a reasonable amount of space. In our implementation of this algorithm, we used a third party library, ojAlgo, to perform the matrix calculations necessary to solve the least-squares system described in the paper. This part also takes on the order of ten seconds to complete.

Finally, we perform the regular and artistic tone mapping. Here, we used third-party implementations of Mantiuk's tone mapping operators, allowing us to spend more time experimenting with different tone mapping choices instead of spending all of our time implementing one choice without knowing how it would perform. These operations

take a total of approximately 45 seconds to complete. Once finished, the images are compressed and returned to the phone, where they can be viewed or offloaded to a computer.

Dane completed the initial implementations of Debevec's algorithm and the image alignment algorithm. Jeff integrated these implementations into the client-server architecture, fixing some issues with Debevec's algorithm, and wrote the Android client and PHP server code.

Experimental Results and Discussion

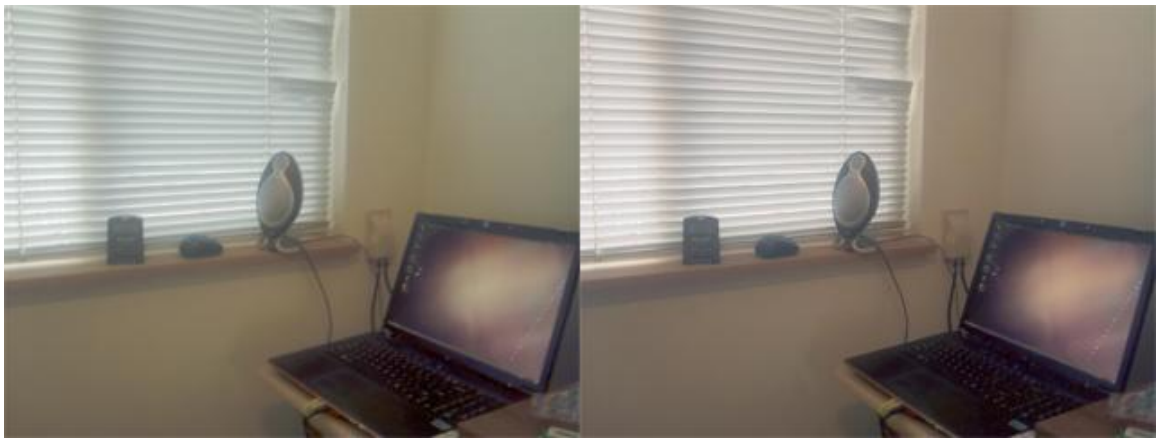

Figure 3: HDR image without alignment (left), image with MTB alignment (right)

We tested our algorithms on a number of images taken with the mobile phone. As a test of image alignment, performed the above process with and without the alignment. We found that the algorithm worked well for small translations such as the ones that occurred above. In the left image, there is visible blurring, perhaps most easily seen around the edge of the speaker in the center of the image. This is reduced in the right image, which appears much sharper.

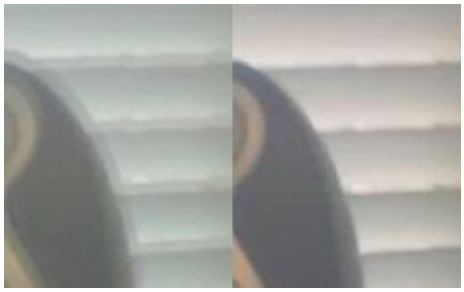

Figure 4: Close-up of speaker from Figure 3

For the rest of the HDR process, we compared the .hdr radiance maps we generated with those created by Luminance HDR, a third-party package for HDR construction and tone mapping which uses the same mapping software as we used in our program. Generally, we found that the two .hdr images resulted in similar tone-mapped images at the end of the full process, and so we are satisfied with our implementation of Debevec's algorithm. Certain pictures, especially those that contain a very bright element in an otherwise dark scene, result in significant color distortion in the .hdr file. However, we found that this distortion also existed in the .hdr files created by Luminance, albeit to a lesser degree. We believe this can be mostly attributed to the low quality of the phone camera and the noise and blooming light in these types of scenes.

Figure 5: Original image at 0 EV (left), regular tone-mapped HDR image (right)

In the above test image, the algorithm's benefits and drawbacks are well illustrated. The default exposure does well at capturing the details of the apartment building, but the bright sky causes blooming artifacts near the top edge of the building,

and the trees on the left side of the image disappear. In contrast, in the image on the right, both the building and the trees are visible, and the roof of the building is more clear, although the bricks of the building appear less vibrant than before. For this image, we found that the artistic rendering (Figure 1) was more visually appealing, as the clouds in the sky were more visible, and the higher contrast made the image appear less dull.

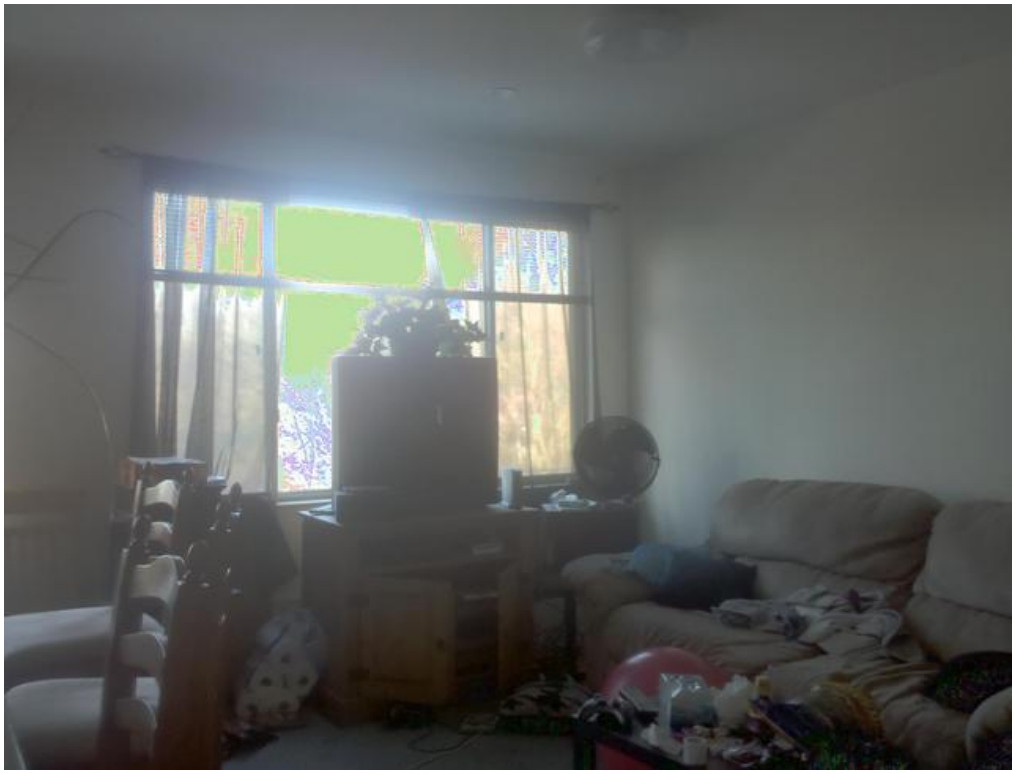

Figure 6: Radiance map errors

The above image is an example of a problem that arose in very few test images. The colors of the scene outside the window are extremely distorted, and there is seemingly random green and red noise in the bottom right corner of the image. However, when we tested the same original pictures using Luminance, we found that we encountered similar issues. Thus, we believe that the errors are caused by poor input data and not by a systematic error in our implementation, which may be rectified by choosing sample points more intelligently than by just using random selection.

Figure 7: From left to right, regular photo with 2 sec. exposure, realistic HDR rendering, artistic HDR rendering

For additional testing, we also rendered the Memorial Church test images from Debevec's paper to see how our algorithm performed on well-known, high quality images. In these tests, I disabled our alignment algorithm as the images were pre-aligned using homographies, and the fake blue background resulting from the image warps led to false realignments. Although we tuned our system for mobile phone cameras, we found that the system worked successfully on this data set, taken with a professional camera. Both the realistic and the artistic renderings are visually appealing, and both exhibit a wider dynamic range than any of the initial photos that were taken.

As a whole, we feel that our HDR implementation was a success. The process from start to finish takes slightly over one minute, which is lengthy but not unreasonable. Perhaps more importantly, the bottleneck is caused mostly by code which is not our own. We were disappointed that we could not solve all of the color distortion issues, but we found that the images we produced were generally high quality.

Future Work

We think our application could be improved in the following areas:

- *Android interface*. The application is fairly sparse - it simply takes the pictures and shows a progress bar to the user while the server is processing. We could improve it by offering the user different options to tweak the HDR generation and different tone-mapping operators, ensuring the best possible result.

- *Speed*. Although a minute is not unreasonable, we ideally want the process to take on the order of a few seconds. This may be achievable by parallelizing some work, both within the algorithms and across them (i.e. by running different tone mappers simultaneously instead of sequentially).

- *Quality*. It should be possible to work around the noise issues we were having. Debevec's paper notes that the points used to reconstruct the radiance map should be representative of the image, well spread in intensity, free of noise, and locally continuous so that the scene radiance is constant at that pixel. Unfortunately, Debevec's approach to solving this problem was to hand label about 50 points, which is obviously not an option for our automated algorithm. We chose to just choose points randomly, but a more systematic approach that examined the intensity values might solve the noise problem and lead to generally better results.