Recovering A High Dynamic Range Image and the Camera's Response Function Simultaneously

Jakob Brünker

Abstract

In this exercise, we were instructed to use the algorithm proposed by [RBS99] to take a set of images of the same scene with different exposure times and use them to recover an high dynamic range image of the scene, as well as the response curve of the used camera. As well as doing this, I have implemented a way to write the recovered image to a file in the OpenEXR format, and to display a tonemapped version of the recovered image, using the algorithm from [DMAC03].

1 Introduction

For this exercise, I first implemented the algorithm proposed in [RBS99], which can be used to take a set of spatially aligned input images of a scene, with different exposure times, and, simultaneously, iteratively recover the response curve of the camera and the luminance of the original scene. This will be discussed in section 2.

Additionally, I implemented a way for the recovered image to be saved in OpenEXR format. This will be discussed in section 3.

I also implemented the tonemapping algorithm found in [DMAC03], which makes it possible to see the entire dynamic range of a scene mapped to a regular screen. This is discussed in section 4.

Finally, in section 5, I will explain how to compile, run and use the program.

2 Recovering An HDR Image and the Response Function

The algorithm that has been used was the one found in [RBS99]. In that paper, the response function is initially assumed to be linear. This, while already producing decent results, is of course not perfectly accurate for any real camera. For this reason, [RBS99] use an algorithm that iteratively finds a good approximation of the response curve of the camera that has been used as well as a high dynamic range image of the scene.

One problem I had was that, while the proposed weight function in [RBS99] sets the weight of extreme pixel values, i. e. close to 0 or close to 255, very low, they are still not zero, the weight for a pixel value of 255, for example, is roughly 0.0183, cf. equation (5) in [RBS99]. This results in absurd looking images and response functions. An example is provided in figure 1. The reason for this is probably that the denominator in equation (8) becomes very small for high pixel values, especially because images with longer exposure times are weighted more stongly, and those are more likely to be overexposed.

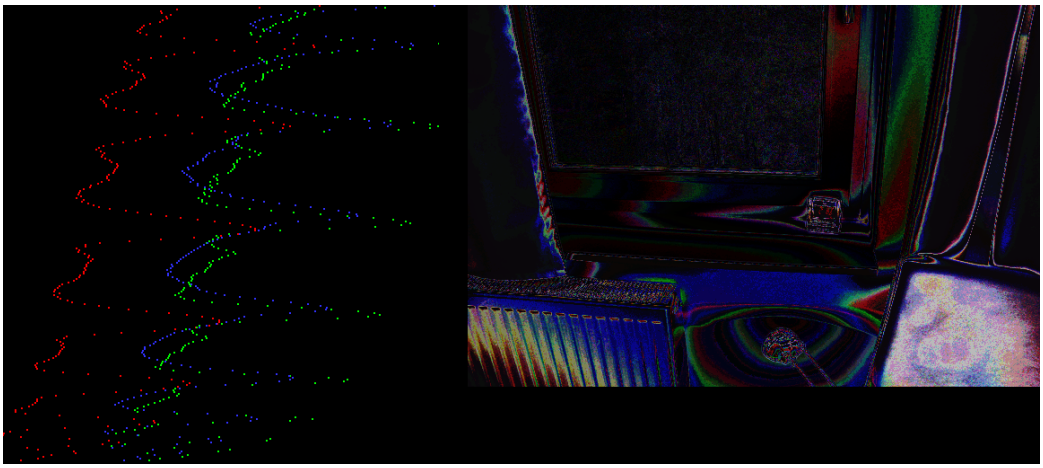

Figure 1: The image that results when the weights are not correct. For the response function on the left, the intensity goes up towards the right, and the resulting pixel value goes up towards the top. The intensity is on a logarithmic scale.

I had multiple test scenes, but the only other serious problems I encountered happened in one where the camera was pointed directly at a lightbulb, leading to parts of the image being completely overexposed even on the lowest exposure setting. The first problem is that the denominator in equation (8) becomes 0 in this case (because the weight was set to 0 due to the previously mentioned issue). The numerator is also 0, which means the result

is undefined. In subsequent iterations, this quickly leads to all values of the response function being undefined. This was fixed by setting the estimated luminance to 0 in these cases.

This, however, is obviously not really correct - the pixels were completely overexposed, and are now completely black in the final image. This can be fixed by setting them to the maximum observed exposure value, since they are at least that bright.

In figure 2 are two images that have been created by simulating a certain exposure time after creating a high dynamic range image with this algorithm, as well as the respective recovered response curves. The iterative process was executed five times for these images.

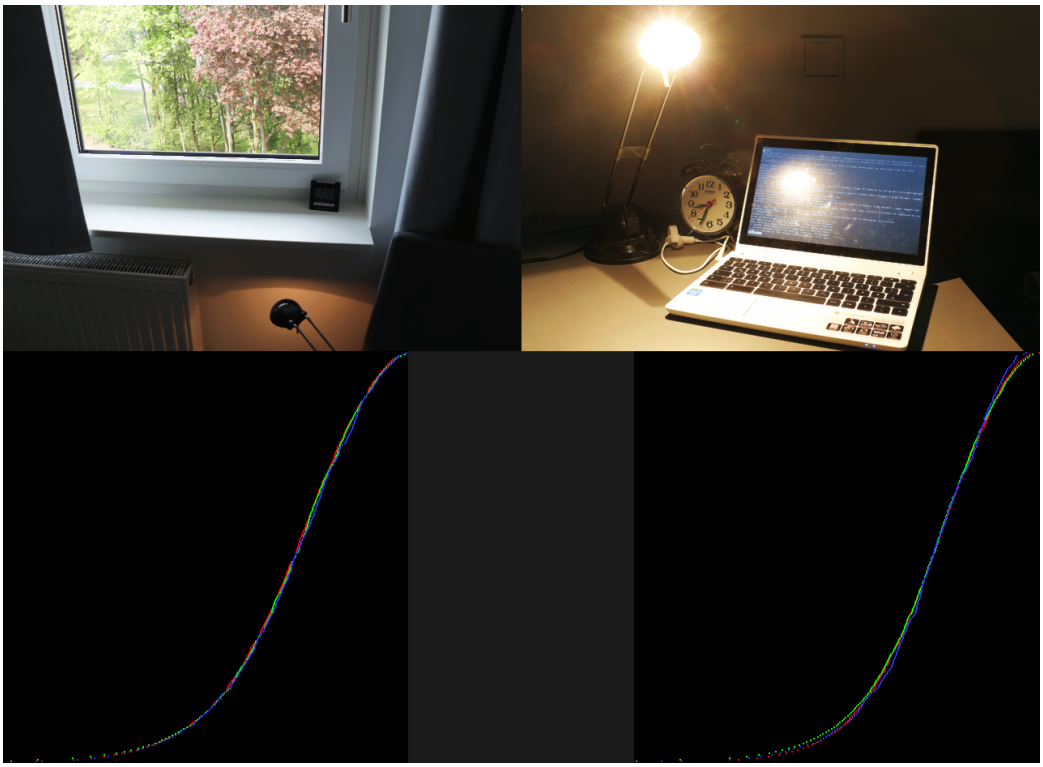

Figure 2: A window scene on the left, with simulated 1/100s exposure time, and an indoor scene on the right, looking at a lightbulb, with simulated 1/3s exposure time. Especially the blue response function appears to be slightly different for the images.

3 Writing the HDR Image to an OpenEXR File

Writing the image as a file in the OpenEXR format was fairly straightforward. The OpenEXR library has a function that one can use to simply write an array of floating point pixel values as an HDR image. I encountered essentially no problems while doing this.

4 Using Tonemapping to Display the Entire Dynamic Range

To map the high dynamic range image to the eight bit values displayable by a typical screen, I used the algorithm from [DMAC03]. This consists of several steps.

I first converted the image to the CIE XYZ colorspace. For this, I used the values provided at [Wik16]. Subsequently, I calculated the maximum luminance (Y-value) in the image. I then used equation (4) from [DMAC03] to calculate the luminance at each value. While I initally expected to only have to do this with the Y-values, this did not produce useful results. However, applying the same equation to the X and Z values resulted in pleasing images, that preserved detail from the entire dynamic range. The result is converted back to the RGB colorspace.

[DMAC03] suggest using gamma correction on the resulting image. I did implement this; however, I did not notice an improvement in the image. Unfortunately, the saturation in the tonemapped images seems quite low in bright areas. In figure 3, three scenes can be seen, after this algorithm has been used on their photographs.

5 Documentation for the Program

To run the program, you need a `config.cfg` file in the directory you are running it from, which contains the relative or absolute path to a `.hdrgen` file that specifies which images should be used. You can then run `make`. (This Makefile probably only works under UNIX-like operating systems, because it relies on X11. The source code, however, should work on Windows as well.) This will run the program after compiling it if necessary.

The program will then generate an image in the OpenEXR format and save it with the same name as the `.hdrgen` file, replacing the file ending with `.exr`. Similarly, it will write a file containing the values of the response curve

Figure 3: The tonemapped images. Note that the image on the top left is quite dark - this is because of the extremely high intensities produced by the lightbulb being part of the image. However, it is also quite apparent that details of all intensities are preserved, as evidenced by the shading inside the lampshade

in a file with the same name as the `.hdrgen` file, replacing the file ending with `.txt`. It lists the values for red pixels in the first column, the ones for green pixels in the second column, and the ones for blue pixels in the third column.

It will also open several windows to show a tonemapped version of the HDR image as well as 3 simulated exposures: 0.01 seconds, 0.05 seconds, and 0.33 seconds. It will also show the recovered response curve; the required light to reach a certain pixel value for a certain color increases towards the right, and the pixel value reached increases towards the top. The intensities are on a logarithmic scale.

To exit the program, simply close one of the windows it opens.

References

[RBS99] M. A. Robertson, S. Borman, and R. L. Stevenson. Dynamic Range Improvement Through Multiple Exposures. In *Proceedings of ICIP*, 1999.

[DMAC03] F. Drago, K. Myszkowski, T. Annen, and N. Chiba. Adaptive Logarithmic Mapping For Displaying High Contrast Scenes. In *Eurographics* 22(3), 2003.

[Wik16] Wikipedia. CIE 1931 color space. `https://en.wikipedia.org/wiki/CIE_XYZ#Construction_of_the_CIE_XYZ_color_space_from_the_Wright.E2.80.93Guild_data` (accessed April 30, 2016).