Tone Mapping

Beomsu Kim bek001@ucsd.edu

July 24, 2023

1 Task Description and Background

Tone mapping has been a challenging task since the display cannot express the extreme wide range of brightness of the light our eyes can see. To produce an image that is close to the one that is seen with human visual system has been the long time goal researchers pursued. The inevitable limitation that display devices cannot exceed a certain brightness range forced us to find another way than just to use the raw light data or luminance values. Also, if there is a huge difference in the ratio of luminance values, rendering becomes difficult. Too bright lights can dazzle the beholder resulting in inability to recognize their neighbors by increasing their luminance too much. I am sure that everyone has an experience that the display on their phone on camera becomes too dark or bright losing details when there is a sudden change of surrounding lights. Plenty of sunset photos are affected by the huge difference of luminance values. There were researches regarding the topic, tone mapping, describing how to handle this issue. There are two types of tone mapping. Global tone mapping can be thought of as wearing sunglasses. Regardless of the brightness context, a filter is applied to the whole image. On the other hand, local tone mapping responds differently to each input pixel depending on its surroundings. It is based on the fact that human vision reacts differently to the same light depending on its context. Tone mapping 1 is following a basic photographic method called Zone System to solve the issue. Secondly, tone mapping 2 is relying on a method called Bilateral filter to reduce contrast and preserve details at the same time. Lastly, log mapping is employing a fast logarithmic compression of luminance distributions and bias functions to address the problem.

2 My Approach

Before stepping into tone mapping, I rendered my scenes. First, I rendered a bathroom scene. The scene has a window so that it is easy to spot the difference between the one with a high radiance and the low radiance. I varied the radiance of the scene. The original scene has area light of radiance (30,30,30). I experimented with the radiance to (300,300,300), (1000,1000,1000), and (3000,3000,3000) to compare what it looks like after each method. Increasing

Figure 1: Max depth 3: Mirror is dark

the ray depth can be helpful for catching details of the scene. Since the mirror needs to bounce at least once more, too low depth can harm performance of the program whose scene is containing a mirror. Rendering becoming to dark like shown below in Figure 4.

Separation between the rendering of the scene and the filtering enabled me to accelerate the process. I do not have to wait for a complex scene to render every time I want to filter the image. Similar to what I did for importing and exporting Image3 type variable, I just have to load the *img* variable using torrey's built in imread method. Since the rendering takes the most of the time and has a risk of crashing if I am running too much samples for my machine, it helped me reduce the time to test the functionality of tone mapping technique.

To avoid stack overflow from killing my process, I decided to take divide and conquer strategy to achieve the goal. When I increase the number of samples beyond a certain limit, the program crashed and everything had to be done again. I figured that the result is just an average of samples why don't I take averages of small samples(noraml results of small sample) and take an average of them again to get the average of the total number. I put the result Image3 variable in a .exr format and import the file after all the small averages are done. The result was successful. I managed to make it and automate them by putting the required commands into a bash executable file. Finally, I removed the noise at last.

As a preprocess for tone mapping, I converted rgb data to luminance values. For tone map 1 and 2, I am using the same approach the Reinhard suggests, dot product rgb vector with (.27, .67, .06). As for tone map 3, I used $(\frac{20}{61}, \frac{40}{61}, \frac{1}{61})$ for my dot product. After gathering the world lumanance which was previously the $img(x, y)$ in torrey, display luminance can be obtained by either of global or local tone reproduction process. I used the result of the calculation both on the global or local as a ratio of luminance values and multiply it by the original pixel value to produce filtered value.

Figure 2: radiance (300,300,300) after tone mapping 1

3 Tonemap1

According to Reinhard, photographers have similar problems as well. Reinhard suggests global and local tone mapping can be employed to solve the tone reproduction problem[2]. Tone mapping can be done first by applying a scaling that can be thought of as similar to adjusting exposure in a camera.

Global scale method is based on an idea called Zone System. Zones are ranging from 0 to 10. Although I can use convolutions, it also can be done by using spatial approach to produce the result. The method begins as follows.

1. Get the log based average (ave_L) of the L_w for all the pixels to set a useful approximation of the scene's key value

2. Select the key value a that is related to the key of the image after the scale. Reinhard suggests trying .18, going up .36, .72, varying down to .09, and .045[2].

3. Calculate the scaled luminance L by $\frac{a}{ave_L} L_w$

4. Set L_{white} for the smallest luminance to map to pure white.

5. Get display luminance(L_d) by $\frac{L*(1+\frac{L}{L_{white}^2})}{1+L}$ for each pixel.

Unfortunately, my bath image is too big to give out a reasonable number for ave_L. It is not robust enough for a big file. If this applies, I decided to skip the global scale process.

After the global procedure has been completed, if necessary, local mapping is processed. Local scale uses local average of the pixels to find the right radius to apply dodging and burning. Both of the terms came from photographic practice. Dodging for avoiding making the resulting negative be less brighter. Burning stands for the focus of the light to give them more brightness to produce a more natural image. The local approach has some short backs in that wrong choice of sampling radius can result in the unwanted artifacts.

1. Get display luminance(L_d) by $\frac{L*(1+\frac{L}{L_{white}^2})}{1+V}$. where $V = V_1 - V_2$

The key for the algorithm is how to find the s that is the smallest such that $V = V_1 - V_2 > \epsilon$ using a center-surround function. Local sampling uses an adaptive approach to find the right sample radius in local sampling. As a result, sample radius of a pixel can be different than that of others. Although the method has some draw backs that it is slower than others in that it has

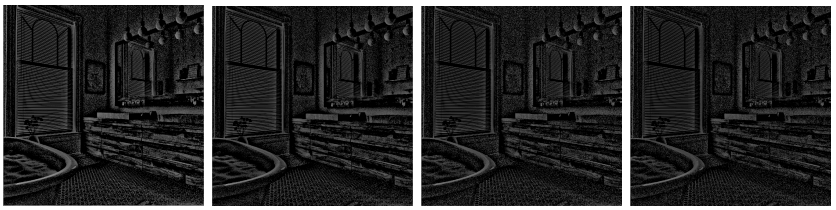

Figure 3: Detail layers from radiance 30 to 3000. Details (edges) remain still in tone map 2

to find the right sample size s for every pixel and can produce an artifact, if properly implemented, it can mimic human visual system since human vision is responding locally as well[2].

4 Tonemap2

The second approach starts with the decomposition of images depending on the frequencies of the pixel luminance distribution. The procedure filters images into the base layer and detail layer with Bilateral filter. Bilateral filter is a non-linear robust edge-preserving filter introduced by Tomasi et al in 1998[1]. It is similar to Gaussian blur but has an ability to adapt to pixels' intensity difference. The concept of bilateral filtering relies on the statistical idea that a robust model is not sensitive to outliers but responsive to similar values[1]. It blurs small variations but barely touches large ones. Although the idea of bilateral filtering is not strictly a tone reproduction in that it does not attempt to imitate human vision, the method can produce overall natural output[1]. Compared to its cousin, anisotropic diffusion, bilateral filter does not guarantee energy preservation but it is way faster[1].

Durand aimed to a fast and practical method It reduces the contrast of only the base layer to speed up the process and protect the details at the same time. The selective reproduction of the contrast can be done by a method called bilateral filtering. Similar to the first approach, this version of tone mapping uses Fourier transform on convolution to figure out the display luminance. Overall detail has been preserved after the filtering in Figure 3. Edges remained throughout the process to preserve the detail it promised.

According to the source code in the website of the paper, the pseudo code for tone mapping2 is the same as the following.

(a) Image3::img variable has the raw image information
(b) Compute an intensity layer I
(c) Compute $log(I)$
(d) Filter $log(I)$ using the bilateral filter to get $log(F)$
(e) Compute a detail channel $D = log(I) - log(F)$
(f) Compute: $\delta = max[log(F)] - min[log(F)]$
(g) Compute: $\gamma = log(constrast)/\delta$

Figure 4: Tone map 3's result of the input of radiance 3000

(h) Compute the new intensity layer: $N = 10^{\gamma log(F)+D}$
(i) Scale the RGB values by N/I
(j) Scale using scale factor

The process share parts with other techniques that a, b, c, and i are the same as what I did for other tone mappings. I choose not to use gamma correction at the last step since it gives more pleasing images without it. Although the overall tone has be shifted, the details of the image remained. Most importantly, the results for all the variation of radiance are similar. It is desirable since the input images are rendering the same scene with only difference on radiance values.

5 Tonemap3

log map is using logarithmic method to remap the light to its display values to simulate human vision. In this process, remapping of the luminance is done by log mapping and gamma correction. A power function called bias function is used here. Interpolation of the bases is done to express wide range of luminance[3]. Fortunately, I have found the source code for the implementation of it. Since the program was written in c, I had to modify it to be in c++. Down side is that it does not react robust to noise in the input data white other methods can find a way around it. However if the input is clean, it is showing an incredible consistency regardless of input radiance. The log map approach showed an astonishing consistency throughout the various radiance of the images. I used default parameters as given in the source code.

6 Works in Progress

I am planning to 1. add Perlin noise. Perlin noise is a good way to produce a nice marble pattern. I am working on implementing it on my torrey renderer. 2. add glass material Since glass is often used in real life and it will make my rendering more pleasing and compelling , I want to work on adding glass material. 3. add tennis court scene using blender I am a big fan of tennis. UCSD north court is one of my favorate place to play it. I always wanted to construct a digital

replica of a tennis court. 4. Try to reduce noise by upgrading pdfs. My solution to the rendering of scenes relies on the large number of sample counts. If I can remove unnecessary samples I can reduce the sample count to only sample those that are important for rendering. I will be able to speed up the process if I can manage to reduce the sample count without noticeable degradation of the quality of the output image. I am planning to add another pdf function than just cosine pdf or light pdf. Since my code's existing pdf function relies on just random direction, it sends the ray to no where resulting in the noise to work with. I am hoping that sending rays to where there is a not occluded shape can decrease unnecessary ray shootings. I am sure that there is a research paper I can read to get an idea. 5. Speed up tonemap 1. I am planning to make an adjustment to my code to use convolution. Tonemap1 mainly consists of element wise multiplication operations that can be done with fft. Instead of $O(n^2)$ of iterating through every pixel, the cost of calculation becomes $O(nlogn)$ overall if I use fft for the implementation.

7 Discussion and Conclusion

The use of shell file (.sh) to automate the build and debug process has been helpful for the working of the project. I put the debugging tool lldb with my corresponding arguments in a file to debug faster and more efficiently.

I pruned unused files for the project including hw1 through hw4 to make the project lighter and thus easier to navigate. Only files that will be used remained after that. Instead of putting every file on src folder, which is the way I have been using so far, I decided to create include directory to put them in good shape. I also put files in their corresponding directory depending on the purpose of them.

I also changed my CMakeLists.txt file to organize the code structure. I had worked and repeated on the cmake tutorial to organized the project to the way I am comfortable with. The thorough repetition of the cmake tutorial was helpful in that I can use ctest to find a potential bug that hinders the progress of my project. The cmake tutorial reminds me of ctests and I decided to use it to find hiding bugs efficiently. I put unit tests at the place where I cannot approach easily or it takes too much time to get to the function normally. The use of unit test reduced the time taken for me to test a function without waiting for another process to be completed. Since the project is handling big files to load, this method was so helpful in that I can skip the most time consuming step.

For the random number generator, I was using $< static\_cast < float > ()/RAND\_MAX$ to produce a number that is from 0 to 1. It was heavy but I used it because of my past experience that the recommended rng method gave me strange results. It turned out that using different seed for each sampling can solve the strange result. It was nice to learn a new material. Tailoring code is always fun.

Producing pictures that are more realistic and pleasing is so crucial. Photos are taken to record the moment that the taker wants to remain or stay. Not

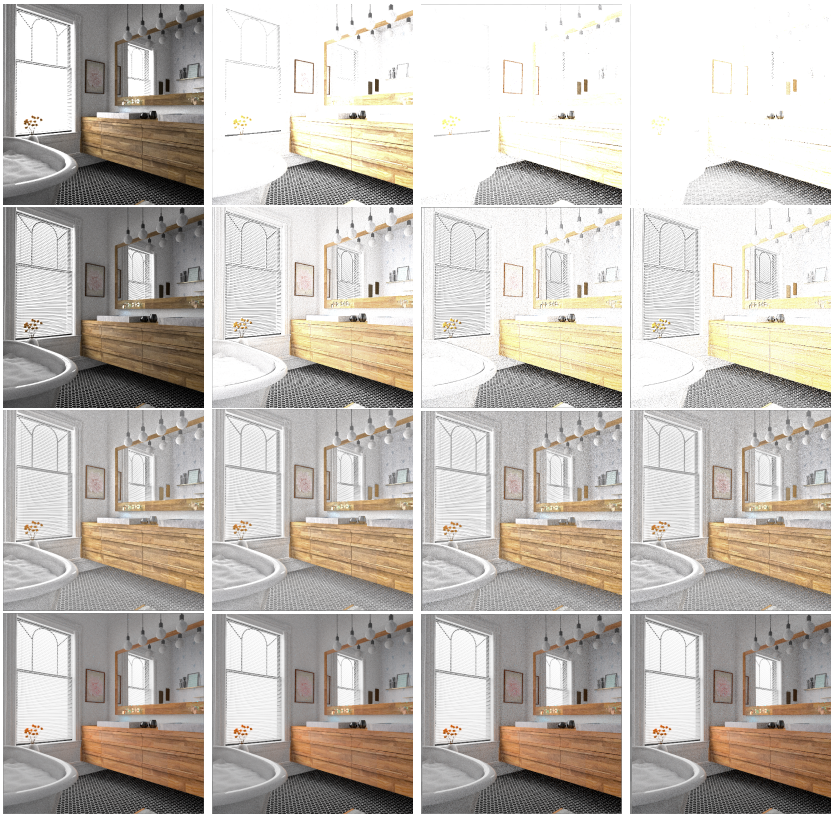

Figure 5: Rendered image of bathroom scene before filtering radiance 30, 300, 1000, 3000

Figure 6: Tone Map 1 results

Figure 7: Tone Map 2 results

Figure 8: Tone Map 3 results

every moment is captured. Only the moments or scenes that are worth the capturing are going to be recorded or synthesised. Whether the scene to be described is the rising sun at a nice beach or a peaceful moment of glaring rows of lights at the silent night, the technique of tone mapping should work out to produce a better image.

I personally like the log tone mapping because of the consistency it gave me and its results are more pleasing to watch. I put together all the raw rendering image and their corresponding results. The 0th row is the raw rendering of my torrey program of radiance from 30, 300, 1000, 3000 from left. For $i = 1, 2, 3$, the ith row is the result of the ith tone mapping method of its corresponding radiance.

References

[1] Fredo Durand and Julie Dorsey. "Fast Bilateral Filtering for the Display of High-Dynamic-Range Images". In: (2006).

[2] Peter Shirley Erik Reinhard Michael Stark and Jim Ferwarda. "Photographic Tone Reproduction for Digital Images". In: (2002).

[3] T. Anmen F. Drago K. Myszkowski and N. Chiba. "Adaptive Logarithmic Mapping For Displaying High Contrast Scenes". In: (2003).