CPSC 4040/6040
Computer Graphics Images

Assigned: Oct. 15, 2015
Due: 11:59pm, Thurs., Oct. 29, 2015

Programming Assignment 5 — Tone Mapping *(Grading: 0–10 points)*

Problem Description

In this assignment you will develop a program to tone map HDR images using the point and region processing filtering we've discussed in class. HDR images store a high dynamic range of values, typically five or more orders of magnitude. Converting them to a display space (which is limited to two orders of magnitude) AND preserving interesting features can be quite tricky.

Your code will implement two different HDR tone mapping operators, and for 6040 students they will implement a third operator. Like in Programming Assignment 03, you will also be required to produce the highest quality (in terms of realism and/or aesthetics) tone mapped output for an input image of your choice. While I have included two input HDR images, `smalldesignCenter.hdr` and `Ocean.exr`, for testing, some other places where you can download sample HDR data include:

- `http://www.cs.utah.edu/~reinhard/cdrom/`
- `http://people.csail.mit.edu/fredo/PUBLI/Siggraph2002/`
- `http://www.anyhere.com/gward/hdrenc/pages/originals.html`
- `http://www.pauldebevec.com/Research/HDR/`
- `http://www.openexr.com/downloads.html`

Your program will again work with OIIO and you will make use of OpenGL for displaying the final result. This should be built on the software you developed Programming Assignment 04 as well as earlier assignments.

Basic Requirements

Image Reading and Writing Your program should be called `tonemap`. It should take as input an HDR image and read only its first three channels (you may assume they are red, green, and blue). However, it should only read HDR formatted images, for the purposes of this assignment the formats you should support are `.hdr` and `.exr`. Both of these image formats store data using floating point formats. You should modify your reading to read data as the C++ `float` type (even if the data is, for example, `half` or `double`, you can have OIIO convert it to floats). This also means you should modify your pixmap format to store floats for its channels. Be sure to change your `TypeDesc::UINT8`'s appropriately.

One caveat I have found with experimentation is that my version of OIIO does not support striding properly when using `TypeDesc::FLOAT`. If you were using striding to flip the image directly with `ImageInput::read_image()`, you may have to instead write a small loop.

Finally, your code should support writing the tone mapped image to an filename specified as a second, optional parameter, as in Programming Assignment 03. When the user presses the `w` or `W` key, you should be able to support writing images to `.exr` and `.hdr` files. The same code should also support writing low-range formats such as `.png`.

Image Display You should display your image using OpenGL. When using `glDrawPixels`, change `GL_UNSIGNED_BYTE` to `GL_FLOAT`. OpenGL expects image data in the range $[0, 1]$, if the data is outside of its range it clamps it for you.

For this assignment, you will benefit from being able to display both the original data as well as the tone mapped data. Being able to switch between the different tone maps is useful for comparison. By pressing the `s` or `S` key, you should switch between the original image its tone mapped version. Alternatively, you may also want to consider displaying the input image in one window, and then compute and display the tone mapped image in a second window. You may find the point-spread function example from earlier in class useful for displaying multiple images.

Simple Tone Mapping Your first tone mapping operator will be to gamma correct the image, in its log space of its luminance. This is an example point-processing technique. To do so requires three steps:

1. First construct the luminance data for the image. You can use whatever operator you want (i.e. YUV, xyY, etc.). The simplest is $L_w = \frac{1}{61.0} \times (20.0R + 40.0G + B)$ for each pixel.
2. Next, compute $L_d = L_w^\gamma$ for each pixel. However, instead of using the `powf()` function, it is more efficient to do this in the log space with multiplication. To do so, compute $\log(L_w)$ for each pixel, then perform $\log(L_d) = \gamma \times \log(L_w)$. Finally compute $L_d = \exp(\log(L_d))$.
3. Finally, compute $C_d = \frac{L_d}{L_w} \times C$ for each color channel $C = \{R, G, B\}$. This scales each color of each pixel relative to the target display luminance and the input world luminance.

Try varying γ and see what you get. You should be able to improve the input over the original display, however in general there will be a number of issues which we attempt to fix.

Tone Mapping with Convolution Section 10.2 of Szeliski describes a very simple tone mapping operator which separately processes the low-pass (B) and high-pass (S) data of the $\log(L_w)$ channel. We will build these two channels using convolution.

First, you will need to implement a convolution operator, g, that performs low-pass smoothing by doing $\log(L_w) \otimes g$. g can be anything you like, but I suggest using a box filter of varying sizes (even 5×5 will improve the tone map, but up to 20×20 may do better). Later you may want to experiment with tent or Gaussian filters. The basic process you will perform is to compute a collection of intermediate images:

1. $B = \log(L_w) \otimes g$
2. $S = \log(L_w) - B$
3. $\log(L_d) = \gamma \times B + S$
4. $L_d = \exp(\log(L_d))$
5. $C_d = \frac{L_d}{L_w} \times C$

While the above steps could be completed by creating five new images, you will find that you likely do not need to store each of them separately. Instead, you could compute the necessary values in place. Also note that Szeliski, incorrectly, multiplies S by γ in step 3.

Setting γ is this situation can be tricky to understand. In general, the idea is that you want to preserve some contrast threshold c. On Durand's website (near the bottom) he suggests using

a γ, called "compression factor" set relative to the minimum and maximum of B. Specifically, $\gamma = \log(c)/(\max(B) - \min(B))$. I found that $c \in [5, 100]$ worked well. He also suggests subtracting an absolute scale from the formulation. I found that both of these changes improved my results.

Code Documentation Please see Programming Assignment 01 for the expected requirements. Your code will be evaluated based on code structure, commenting, and the inclusion of a README and build script.

Advanced Extension for 6040 students (worth 3 points)

Modify your convolution-based tone mapper to use a bilateral filter. The idea is that when you tone map with convolution, you will create halos based on how big of a window you convolve against. These halos are the result of crossing edges in the image.

To do so, you will modify your convolution to produce a non-linear operator. Durand suggests quite a few options for this, you are welcome to experiment with your own. I found that multiplying by a weight of $w = \exp(-\text{clamp}(d^2))$ where d equals the difference in $\log(L_w)$ between the center pixel of the convolution and the whatever other pixel j you are summing.

You are welcome and encouraged to try out other drop off functions. In this case, the goal is to remove halos, but otherwise your results should look fairly similar to tone mapping as Szeliski suggests. This suggested modification will be inefficient, but a fairly good approximation of Durand and Dorsey's bilateral filter. There is no need to attempt any of the optimizations they propose.

Extra Credit Opportunities

Feel free to consider implementing any other tone mapping operator of your choice, including Reinhard's global and/or local operators (discussed in class) as well as many others.

Submission

(Please read all of these instructions carefully.)

Please write the program in C or C++, and I recommend using OpenGL and GLUT graphics routines for the display. Use OIIO for image input and output. Please take extra care to make sure that your homework compiles on the School of Computing's Ubuntu Linux cluster—testing on your own home machine, even if it runs Ubuntu, may not be sufficient. Remember:

both a working build script and README must be provided

Consequently, to receive *any credit whatsoever* this code must compile on the cluster, even if it does not accomplish all of the tasks of the assignment. The grader will give a zero to any code that does not compile.

Submit using the handin procedure outline at `https://handin.cs.clemson.edu/`. You are welcome to use the commandline interface, but the web interface is sufficient. The assignment number is `pa05`.

Finally, since we are using multiple files, please only submit a single file which has aggregated everything. This file should be named `[username]_pa05.tgz` where `[username]` is your Clemson id (please remove the brackets []). For example, mine would be `levinej_pa05.tgz`. Please make sure your build script is at the top level directory.

Rubric for Grading

4040 students will be graded for the following requirements:

I. +2 Image I/O and Display

Code correctly reads, writes, and displays HDR image data. Code correctly displays the tone mapped image and allows switching between the original and tone mapped image by pressing the `s` or `S` key.

II. +2 Simple Tone Mapping

Does the code support a `-g` flag followed by an input γ? Can it correctly compute a luminance channel and correctly scale the luminances of the image in log space using γ.

III. +3 Tone Mapping with Convolution

Does code support a `-c` flag which convolves the log-space luminance? Does the code split the input log-space luminance into a blurred channel B and a sharpened channel S, and recomposes them as $\gamma \times B + S$.

IV. +2 Clarity

Does the code have good structure? Is the code commented and is a README provided? Is a working build script provided?

V. +1 Quality

Best tone mapped images using both operators submitted as `.png` files, as well as README explaining how it was created.

6040 students will receive 7 points for completing the above requirements, scaling the above by 70%. To achieve 10 points they must also complete:

I. +3 Bilateral Filtering

Code supports bilateral filtering with a switch `-b`. It implements a weighting scheme relative to the differences in image log-space luminances. Best tone mapped image using bilateral filter submitted as a `.png` file, as well as README explaining how it was created.

Going above and beyond these requirements may result in extra credit at the discretion of the instructor and grader. Please note any extra features you implement in the README.