CSE 291: Topics in Computer Science and Engineering
Computational Photography, Spring 2021 – Assignment 3

Instructor: Ben Ochoa

Due: Wednesday, April 21, 2021, 11:59 PM

Instructions

- Review the academic integrity and collaboration policies on the course website.
- This assignment must be completed individually.
- This assignment may be completed in the programming language of your choice.
- You may use third party libraries/packages for basic linear algebra, basic image processing, and image file I/O. But, you may not use third party libraries/packages that directly solve the problem. If you are uncertain about using a specific library/package, then please ask the instructional staff whether or not it is allowable.
- You must prepare a report as a pdf file. The report must describe the problems, and your solutions and results. Math must be done in Markdown/LaTeX.
- Additionally, you must create a zip file containing all of your source code, along with an automated build method (e.g., a makefile) and a `readme` file with clear and concise directions on how to build and execute your program.
- The zip file must also contain any output image files.
- You must submit both files (.pdf and .zip) on Gradescope. You must mark each problem on Gradescope in the pdf.
- It is highly recommended that you begin working on this assignment early.

Problems

Remember, the number of polynomial coefficients is ***one more*** *than the polynomial degree.*

1. (5 points) Develop a function/method named `evalPoly` that evaluates a polynomial at a given point. The function must handle an arbitrary number of polynomial coefficients (in either ascending or descending order). Develop a function/method named `polyDerivative` that calculates the derivative of a polynomial. The input is n polynomial coefficients (in either ascending or descending order) and the output is the $n-1$ polynomial coefficients (in either ascending or descending order) of the derivative of the input polynomial.

2. (25 points) Develop a function/method named `estimateCameraResponseInv` that estimates the inverse of the camera response function (modeled as a polynomial of a specified number of coefficients), and a set of nonlinear color encoded 8 or 16 bit unsigned integer per sample images and associated exposures (e.g., shutter times in seconds). The inverse of the camera response function maps the images from the nonlinear

color encoding to a linear color space. The function/method must only utilize samples within the range of specified minimum and maximum correctly exposed samples values to estimate the polynomial coefficients. Further, the function/method must scale polynomial coefficients of channels to preserve chromaticity. Download `7708.6-11.zip` from `https://www.cs.columbia.edu/CAVE/software/rascal/rrslrr.php` and use `estimateCameraResponseInv` to estimate the inverse camera response function (modeled as a polynomial with 3 coefficients) from the data set. In your report, for each channel, include the unscaled polynomial coefficients (ordered in descending powers) and (sum of absolute) error; and scale and scaled coefficients (ordered in descending powers) (hint: use `evalPoly` when computing the error and estimating the scales of the polynomial coefficients of channels.) Additionally, include a plot of the inverse camera response function for each channel (on a single plot).

3. (20 points) Develop a function/method named `calcHDR` that converts a set of nonlinear color encoded 8 or 16 bit unsigned integer per sample images and associated exposures to a (linear) 32 bit floating-point per sample high dynamic range image, given the inverse of the camera response function (hint: use `evalPoly` and `polyDerivative`). The function/method must only utilize samples within the range of specified minimum and maximum correctly exposed sample values to compute the high dynamic range image.

   Some samples will not be properly exposed in at least one standard dynamic range image. Handle these cases as follows.

   - If a sample is underexposed over all exposures, then set it to the minimum possible properly exposed high dynamic range sample value in the high dynamic range image.
   - If a sample is overexposed over all exposures, then set it to the maximum possible properly exposed high dynamic range sample value in the high dynamic range image.
   - Otherwise, the improperly exposed sample is a combination of underexposed and overexposed over all exposures. In this case, it can be set to arbitrary value, but set it to the minimum possible properly exposed high dynamic range sample value in the high dynamic range image (note: in practice, this sample would be determined using image inpainting methods).

   Create a high dynamic range image from the data set `7708.6-11.zip` (inverse camera response function modeled as a polynomial with 3 coefficients) and write the 32 bit floating-point per sample high dynamic range image to the file `7708.6-11_32f.exr`. Convert the image to a 16 bit floating-point per sample high dynamic range image and write it to the file `7708.6-11_16f.exr`. Using `linearTosRGB` from assignment 2, convert the 32 bit floating-point per sample high dynamic range image to a nonlinear sRGB color encoded 8 bit unsigned integer per sample image and write the resulting image to `7708.6-11.png`.

4. (5 points) Similarly, download `7708.24-29.zip` (inverse camera response function

modeled as a polynomial with 3 coefficients) and `7710.1-6.zip` (inverse camera response function modeled as a polynomial with 3 coefficients), and process these data sets using `estimateCameraResponseInv` and `calcHDR`, writing files `7708.24-29_32f.exr`, `7708.24-29_16f.exr`, and `7708.24-29.png`; and `7710.1-6_32f.exr`, `7710.1-6_16f.exr`, and `7710.1-6.png`. View the high dynamic range images at `https://viewer.openhdr.org/` and briefly discuss your results for these three data sets.

5. (0 points) Optionally, invert the polynomial mapping of the inverse camera response function to determine the (forward) camera response function (as a polynomial). Use the camera response function to convert the 32 bit floating-point per sample high dynamic range image to a nonlinear color encoded 8 bit unsigned integer per sample image and write the resulting images to file, comparing them with both the original images in the data sets and the high dynamic range images.