Homework Assignment 2

15-463/663/862, Computational Photography, Fall 2021
Carnegie Mellon University

Due Monday, Oct. 4, at 11:59 pm

The purpose of this assignment is to explore high dynamic range (HDR) imaging, noise calibration, color calibration, and tonemapping. As we discussed in class, HDR imaging can be used to create floating-point precision images that linearly map to scene radiance values. Noise calibration is the process of figuring out your camera's noise characteristics, and using them to improve your HDR images. Color calibration ensures that the colors you see in the image match some groundtruth RGB values. Tonemapping algorithms compress the dynamic range of HDR images to an 8-bit range, so that they can be shown on a display. To get full credit, you will need to apply these steps to both an exposure stack provided by us, and one that you capture yourselves. Finally, for extra credit, you can investigate HDR imaging using exposure brackets where you vary both the shutter speed and ISO.

Throughout the assignment, we refer to a number of key papers that were also discussed in class. While the assignment and class slides describe most of the steps you need to perform, we highly recommend that you read the associated papers.

As always, there is a "Hints and Information" section at the end of this document that is likely to help. We strongly recommend that you read that section in full before starting to work on the assignment. The Python packages required for this assignment are `numpy`, `skimage`, `matplotlib`, and `cv2` (OpenCV, to read and write HDR files), and you can use the functions provided in the `./src/cp_hw2.py` file of the homework ZIP archive.

1. HDR imaging (60 points)

For this and the following two parts (color correction and tonemapping), you will use an exposure stack we captured in Yannis' office using one of the class cameras (Nikon D3300). The image files are in the `./data/door_stack` directory of the homework ZIP archive. Figure 1 shows two exposures, as well as a (tonemapped) HDR composite.

While not particularly beautiful, the scene has a number of features that make it a good example for HDR: First, there are two areas with very different illumination and dynamic range that no single exposure can simultaneously capture correctly. Second, both areas include colorful items (Toy Story poster in the background of the bright area, Plus-Plus pieces, SIGGRAPH mugs, and book covers in the dark area) that you can use to evaluate the color rendition of your results. Third, the in-focus area has high-detail features (lettering on the book covers and lens/camera markings) that you can use to evaluate the resolution of your results. Finally, the scene includes a color checker than you can use for color calibration in bonus question.

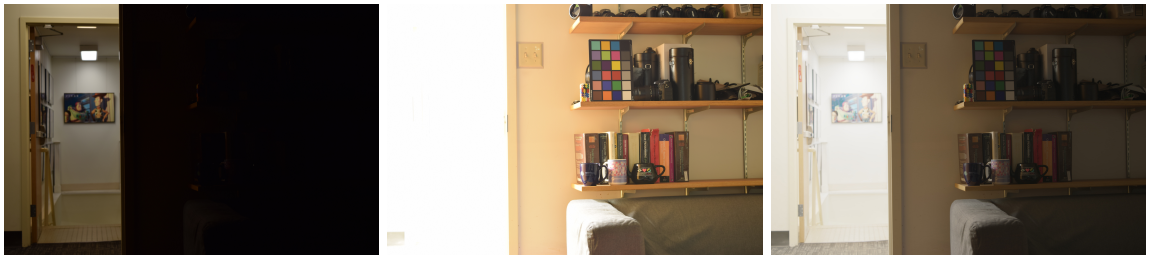

Figure 1: From left to right: Two LDR exposures, and an HDR composite tonemapped using the photographic tonemapping.

You will notice that in, the data folder, there are two sets of images, RAW (`.NEF`) and rendered (`.JPG`). As we discussed in class, the procedure for merging many low dynamic range (LDR) exposures into an HDR

image is different for RAW and rendered images. To appreciate the difference, in this assignment you will create HDR images from both sets of images.

For reference, we captured both exposure stacks with fixed aperture and ISO, and with shutter speeds equal to $\frac{1}{2048} \cdot 2^{k-1}$, where $k \in \{1, \ldots, 16\}$ is the index in an image's file name.

Develop RAW images (5 points). Use `dcraw` to convert the RAW `.NEF` images into *linear* 16-bit `.TIFF` images. For this, you should direct `dcraw` to do white balancing using the camera's profile for white balancing, do demosaicing using high-quality interpolation, and use sRGB as the output color space. Read through `dcraw`'s documentation, and figure out what the correct set of flags for this conversion are. Make sure to report the flags you use in the report you submit with your solution.

Linearize rendered images (25 points). Unlike the RAW images, which are linear, the rendered images are non-linear. As we saw in class, before you can merge them into an HDR image, you first need to perform radiometric calibration in order to undo this non-linearity. You will do this using the method by Debevec and Malik [1]. We describe how this works below, but you are strongly encouraged to read at least Section 2.1 of this paper, which explains the method.

An intensity $I_{ij}^k \in \{0, \ldots, 255\}$ at pixel $\{i, j\}$ of image k relates to some unknown scene flux value L_{ij} as

$$I_{ij}^k = f\left(t^k L_{ij}\right), \tag{1}$$

where t^k is the (known) exposure of image k and f is the unknown non-linearity applied by the camera. If we knew f^{-1}, we could convert I_{ij}^k back to linear measurements.

Instead of f^{-1}, you will recover the function $g := \log\left(f^{-1}\right)$ that maps pixel values I_{ij}^k to $g\left(I_{ij}^k\right) = \log\left(L_{ij}\right) + \log\left(t^k\right)$. This is motivated by the fact that the human visual systems responds to logarithmic, instead of linear, intensity. As the domain of g are the discrete intensity values $\{0, \ldots, 255\}$, g is essentially just a 256-dimensional vector.

Solving for these 256 values may seem impossible, because we know neither g nor L_{ij}. However, if the imaged scene remains static while capturing the exposure stack, we can take advantage of the fact that the value L_{ij} is constant across all LDR images. Then, we can recover g by solving the following least-squares optimization problem,

$$\min_{g, L_{ij}} \sum_{i,j} \sum_{k} \left\{ w\left(I_{ij}^k/255\right) \left[g\left(I_{ij}^k\right) - \log\left(L_{ij}\right) - \log\left(t^k\right) \right] \right\}^2 + \lambda \sum_{z=0}^{255} \left\{ w\left(z/255\right) \nabla^2 g\left(z\right) \right\}^2. \tag{2}$$

The first term in Equation (2) is the *data term*, and encourages values g and L_{ij} to be such that intensities in linear images would scale linearly with exposure time. As we discussed in class, the weights w have to do with the fact that the linear estimates should rely more on well-exposed pixels than on under-exposed or over-exposed pixels. See later in Problem 1 ("Weighting schemes") about what weights exactly you should use. Note that the input to w is divided by 255, because the definitions of the weights assume that the input intensities are in the range $[0, 1]$, whereas the ones you use are in the range $\{0, \ldots, 255\}$.

The second term in Equation (2) is a *regularization term*, and encourages g to be smooth by penalizing solutions g that have large second-derivative magnitudes. Given that g is discrete, the second derivative can be approximated using a Laplacian filter, that is, $\nabla^2 g\left(z\right) = g\left(z+1\right) - 2g\left(z\right) + g\left(z-1\right)$. The parameter λ controls how strongly this regularization affects the final result; it is a hyper-parameter that you will need to experiment with. Note that, when using the photon-optimal weights w_{photon} that require knowing exposure time, you can set the weights *of the regularization term only* to a constant (e.g., $w(z) = 1$).

Solve the *least-squares* optimization problem of Equation (2) by expressing it in matrix form:

$$\left\| \boldsymbol{A}\boldsymbol{v} - \boldsymbol{b} \right\|^2, \tag{3}$$

where $\boldsymbol{A}$ is a matrix, $\boldsymbol{v} = [g; \log\left(L_{ij}\right)]$ are the unknowns, and $\boldsymbol{b}$ is a known vector. Then, use one of NumPy's solvers to recover the unknowns. (See `numpy` function `numpy.linalg.lstsq`.)

While Debevec and Malik [1] recover a different g for each color channel, for this homework we recommend that you process pixels from all three channels simultaneously to recover a single g for all channels. This helps reduce color artifacts in the final HDR composite.

Plot the function g you recovered, then use it to convert the non-linear images I_{ij}^k into linear ones,

$$I_{ij,\text{lin}}^k = \exp\left(g\left(I_{ij}^k\right)\right). \tag{4}$$

Note that you will not use the values L_{ij} you recover from solving (2).

Merge exposure stack into HDR image (30 points). Now that we have two sets of (approximately) linear images, coming from the RAW and rendered files, it is time to merge each one of them into an HDR image. This part will be common for both sets of linear images. Make sure that each HDR image you create only uses images from one or the other set.

Given a set of k LDR linear images corresponding to different exposures t^k, we can merge them into an HDR image either in the linear or in the logarithmic domain. The motivation for linear merging is physical accuracy, whereas the motivation for logarithmic merging is, as mentioned above, human visual perception.

We first introduce some notation. We use $I_{ij,\text{LDR}}^k$ to refer to the intensity value of pixel $\{i,j\}$ of the k-th *original* LDR input image, read directly from either a `.JPG` or a `.TIFF` file. We use $I_{ij,\text{lin}}^k$ to refer to the intensity of pixel $\{i,j\}$ of the k-th *linear* LDR input image; this is either the intensity from Equation (4) when using `.JPG` files, or the same as $I_{ij,\text{LDR}}^k$ when using `.TIFF` files. Additionally, from this point on we assume that $I_{ij,\text{LDR}}^k \in [0,1]$. Therefore, you need to normalize the original LDR input images to the $[0,1]$ range, which you can do by dividing with 255 when using `.JPG` files, and by $2^{16}-1$ when using `.TIFF` files.

With this notation at hand, when using linear merging, we form the HDR image as:

$$I_{ij,\text{HDR}} = \frac{\sum_k w\left(I_{ij,\text{LDR}}^k\right) I_{ij,\text{lin}}^k / t^k}{\sum_k w\left(I_{ij,\text{LDR}}^k\right)}. \tag{5}$$

When using logarithmic merging, we form the HDR image as:

$$I_{ij,\text{HDR}} = \exp\left(\frac{\sum_k w\left(I_{ij,\text{LDR}}^k\right)\left(\log\left(I_{ij,\text{lin}}^k + \epsilon\right) - \log\left(t^k\right)\right)}{\sum_k w\left(I_{ij,\text{LDR}}^k\right)}\right), \tag{6}$$

where ϵ is a small constant to avoid the singularity of the logarithm function at 0. As before, the weights w in Equations (5) and (6) can be used to place more emphasis on well-exposed pixels, and less emphasis on under-exposed or over-exposed ones. See below about what weights to use.

Implement both linear and logarithmic merging for each of the two exposure stacks. Then, store the resulting HDR images as `.HDR` files, which is an open source high dynamic range file format. (See the provided function `writeHDR` in `./src/cp_hw2.py`)

Weighting schemes. There are many possible weighting scheme choices [3]. You will implement four:

$$\begin{aligned}
w_{\text{uniform}}(z) &= \begin{cases} 1, & \text{if } Z_{\min} \le z \le Z_{\max} \\ 0, & \text{otherwise} \end{cases}, \\
w_{\text{tent}}(z) &= \begin{cases} \min(z, 1-z), & \text{if } Z_{\min} \le z \le Z_{\max} \\ 0, & \text{otherwise} \end{cases}, \\
w_{\text{Gaussian}}(z) &= \begin{cases} \exp\left(-4\frac{(z-0.5)^2}{0.5^2}\right), & \text{if } Z_{\min} \le z \le Z_{\max} \\ 0, & \text{otherwise} \end{cases}, \\
w_{\text{photon}}\left(z, t^k\right) &= \begin{cases} t^k, & \text{if } Z_{\min} \le z \le Z_{\max} \\ 0, & \text{otherwise} \end{cases}.
\end{aligned} \tag{7}$$

All the above weighting schemes assume that the intensity values are in the range $z \in [0,1]$. You can experiment with different clipping values $Z_{\min}$ and $Z_{\max}$, but we recommend $Z_{\min} = 0.05, Z_{\max} = 0.95$. Unlike the other schemes, the weights w_{photon} also depend on the exposure under which a pixel was captured.

Note that, when creating an HDR image from the `.JPG` stack, you need to use the same weighting scheme in both Equations (2) (linearization) and (5)-(6) (merging).

Implement all the above weighting schemes, and use them to create HDR images. In total, you will create 16 HDR images: 2 sets of images (RAW and rendered) $\times$ 2 merging schemes (linear and logarithmic) $\times$ 4 weighting schemes (uniform, tent, Gaussian, and photon-noise optimal).

Make your pick. Select one out of the sixteen HDR images you created. You can select, for example, the one that you find the most aesthetically pleasing. Make sure to comment on why you selected the image you did. Note that, as you have not yet tonemapped your HDR images, if you display them directly they will not look very nice; see "Hints and Information".

2. Color correction and white balancing (20 points)

Figure 2: Tonemapped HDR image without (left) and with (right) color correction.

For this part, you will use the HDR image you selected at the end of Part 1. As shown to the left of Figure 2, your tonemapped images will tend to have an orange cast in the dark parts of the room. This is because the very low light inside the room and the large contrast with the light outside the room are throwing the camera's automatic white balancing off. Additionally, even if the white balancing worked perfectly, we have not been very careful about the color space the various image composites reside in.

You could apply any of the automatic white balancing algorithms we discussed in Homework Assignment 1 to ameliorate the issue. But, given that the images include a color checker (Figure 3), it is possible to do better than that and perform accurate color correction.

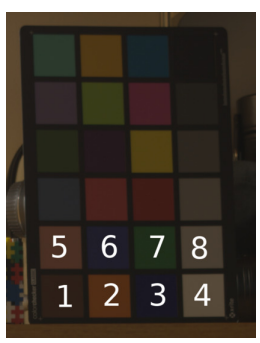

Figure 3: Color checker and patch numbering.

In particular, the color checker is created in such a way that its patches have a specific set of RGB coordinates in the (linear) sRGB color space, when the color checker is viewed under a standard illuminant (so called "D65" illumination, roughly corresponding to daylight at noon). The function `read_colorchecker_gm`, provided in the `./src/cp_hw2.py` file of the homework ZIP archive, returns these ground-truth RGB coordinate

values, with the patches numbered as shown in Figure 3.

Then, in order to have your HDR image show color correctly, you can apply a linear transform on its three channels, so that the color checker's RGB coordinates in the image match the ground-truth coordinates as closely as possible. You can do this as follows.

1. For each color checker patch, crop a square that is fully contained within the patch. (See `matplotlib` function `matplotlib.pyplot.ginput` for interactively recording image coordinates.) Make sure to store the coordinates of these cropped squares, so that you can re-use them. Use the resulting 24 crops to compute average RGB coordinates for each of the color checker's 24 patches.

2. Convert these computed RGB coordinates into *homogeneous* 4×1 coordinates, by appending a 1 as their fourth coordinate.

3. Solve a least-squares problem to compute an *affine transformation*, mapping the measured to the ground-truth homogeneous coordinates.

4. Apply the computed affine transform to your original RGB HDR image. Note that the transformed image may have some negative values, which you should clip to 0.

5. Finally, apply an additional white balancing transform (i.e., multiply each channel with a scalar), so that the RGB coordinates of patch 4 are equal to each other. This is analogous to the manual white balancing in Homework Assignment 1, where now we use patch 4 as the white object in the scene.

Store the color corrected and white balanced HDR image in an `.HDR` file. You should now have two HDR images total: The one from Part 1 that has not been color-corrected, and the one you just created. Compare the color-corrected image with the original, and discuss which one you like the best.

3. Photographic tonemapping (20 points)

Now that you have a couple of HDR images, you need to tonemap them so that you can display them. For this part, you can use whichever of the two HDR images at the end of Part 2 you liked the best.

You will implement the tonemapping operator proposed by Reinhard et al. [4], which is a good baseline to start from when displaying HDR images. We describe how to do this below, but you are strongly encouraged to read at least Sections 2 and 3 of this paper, which explain the rationale behind the specific form of this tonemapping operator, the effect of the various parameters, and its relationship to the zone system used when developing film.

Given pixel values $I_{ij,\mathrm{HDR}}$ of a linear HDR image, photographic tonemapping is performed as

$$I_{ij,\mathrm{TM}} = \frac{\tilde{I}_{ij,\mathrm{HDR}} \left(1 + \frac{\tilde{I}_{ij,\mathrm{HDR}}}{\tilde{I}^2_{\mathrm{white}}}\right)}{1 + \tilde{I}_{ij,\mathrm{HDR}}}, \tag{8}$$

where [1]

$$\tilde{I}_{\mathrm{white}} = B \cdot \max_{i,j} \left(\tilde{I}_{ij,\mathrm{HDR}}\right), \tag{9}$$

$$\tilde{I}_{ij,\mathrm{HDR}} = \frac{K}{I_{m,\mathrm{HDR}}} I_{ij,\mathrm{HDR}}, \tag{10}$$

$$I_{m,\mathrm{HDR}} = \exp\left(\frac{1}{N} \sum_{i,j} \log\left(I_{ij,\mathrm{HDR}} + \epsilon\right)\right). \tag{11}$$

The parameter K is the *key*, and determines how bright or dark the resulting tonemapped rendition is. The parameter B is the *burn*, and can be used to suppress the contrast of the result. Finally, N is the number of pixels, and ϵ is a small constant to avoid the singularity of the logarithm function at 0.

[1] Equation (11) is different from the corresponding Equation (1) in Reinhard et al. [4]. The version given here is correct, and the version in the paper is incorrect.

Implement the photographic operator and apply it to your RGB HDR images in two ways: First, apply it by tonemapping all color channels simultaneously in the same way. Second, apply it only to the luminance channel Y. For the latter, you can use the provided function `lRGB2XYZ` to convert the HDR image from RGB to XYZ, and then convert it to xyY using the definition discussed in class. While in xyY, tonemap the luminance Y while leaving the chromaticity channels x, y untouched. Then, invert the color transform to go back to RGB using the provided function `XYZ2lRGB`.

Experiment with different key and burn values. Some reasonable starting values for the parameters are $K = 0.15$ and $B = 0.95$, but to get good tonemaps you will need to explore different values. Plot representative tonemaps for both the RGB and luminance methods, and discuss your results. Make sure to mention which tonemap you like the most.

4. Create and tonemap your own HDR photo (50 points)

It is now time to apply what you implemented above to your own pictures. To create results which are clearly better than any single exposure, you should take pictures of a scene that actually has a high dynamic range! Good examples include: scenes that have both indoor and outdoor elements (a room with windows), indoor scenes with two different illuminations (like the data you used in parts 1-3), scenes with very strong backlighting, or outdoors scenes during a sunny day with strong shadows. Note that you must use the same camera settings for the data you capture in this part and Part 5; make sure to read the suggestions in Hints and Information on what camera settings to use.

Once you select the scene, capture exposure stacks in RAW and JPEG formats. We suggest using exposures that are equally spaced *in the logarithmic domain*. For example, start with some very low base exposure, and then use exposures that are 2× the base, 4×, 8×, and so on. You can either exhaust the exposure range (i.e., start from the lowest shutter speed possible, and go all the way to the maximum shutter speed in 2x steps), or select an exposure range that works for your scene.

Use the exposure stacks you captured to create two HDR images, one from the RAW and one from the JPEG images. You can use whichever of the HDR variants you implemented in Part 1 you prefer—or you can try out all of them and decide which one looks the best. Store these two images in `.HDR` format. Since you do not have a color checker, you can skip the color calibration step.

Then, process these images using the tonemapping algorithms you implemented in Part 3 (photographic, in RGB or luminance-only). Experiment with different parameters, show a few representative tonemaps, discuss your results, and determine which result you like the most.

The total number of points you will get for this part will depend on how visually compelling the final tonemapped image you create is.

5. Noise calibration and optimal weights (50 points)

As a last step in your HDR pipeline, you will attempt to further improve the fidelity of your HDR composite by implementing a simple noise calibration procedure. We will be following the noise model we discussed in class, but we will assume that the dark current term is zero and can be ignored.

To start the noise calibration procedure, you should print out a ramp intensity image as in Figure 4. You can generate such an image using the `numpy` function `numpy.tile(numpy.linspace(0, 1, 255), (255, 1))`. You should rescale the pattern as needed for it to fill out an entire page after printing.

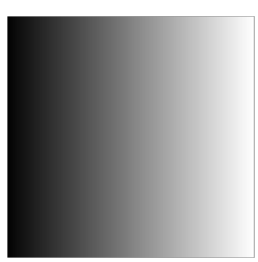

Figure 4: Ramp pattern for noise calibration.

Noise calibration (30 points). Throughout this section, you should use a single shutter speed (one where the entire ramp image is well-exposed), and the same ISO and aperture setting as you did in Part 4.

First, capture about 50 RAW images *with the lens cap on*. Convert these RAW images into *linear* 16-bit `.TIFF` images, as you did in Part 1. Then, average the images to compute the dark frame. Make sure to store the dark frame, as you will be using it shortly.

Now remove the lens cap and capture about $N = 50$ RAW images of the ramp print-out. Convert these RAW images into *linear* 16-bit `.TIFF` images, as you did in Part 1. Then, subtract from each image the dark frame you computed above. For the rest of the noise calibration procedure, you will use only the images after dark frame subtraction.

For a few pixels, plot the histogram of their values across the various images you captured. Discuss what shape these histograms approximately have, and why.

Then, for each pixel, compute its mean value and variance,

$$\mu_{ij} = \frac{1}{N} \sum_n I_{ij}^n, \tag{12}$$

$$\sigma_{ij}^2 = \frac{1}{N-1} \sum_n \left(I_{ij}^n - \mu_{ij} \right)^2 . \tag{13}$$

Round the mean to the nearest integer, which will result in several pixels having the same mean value. Calculate the average variance for this mean value.

As we discussed in class, the variance relates to the mean value as

$$\sigma_{ij}^2 = \mu_{ij} g + \underbrace{\sigma_{\text{read}}^2 g^2 + \sigma_{\text{ADC}}^2}_{\sigma_{\text{additive}}^2}, \tag{14}$$

where g is the camera gain, σ_{read}^2 is the variance of the read noise, and σ_{ADC}^2 is the variance of ADC noise. Therefore, if you plot the variance you computed above as a function of different unique mean values, the result should be approximately a straight line. Fit a line to your unique mean-variance points, and use it to estimate the camera gain g and the total additive noise variance $\sigma_{\text{additive}}^2$. Show the mean-variance plot and the fitted line, and report the estimated gain and variance.

Merging with optimal weights (20 points.) Use the RAW exposure stack you captured in Part 4 to form one last HDR image, this time using the noise-optimal weighting scheme we discussed in class.

For this, first perform dark-frame subtraction, accounting for shutter speed differences: Let's say you computed a dark frame I_{dark} by performing noise calibration using shutter speed t_{nc}. From each image I^k in your exposure stack, subtract the frame $\frac{t^k}{t_{\text{nc}}} I_{\text{dark}}$.

Then, merge the dark-frame-corrected exposure stack using the weights:

$$w_{\text{optimal}}\left(z, t^k\right) = \begin{cases} \frac{\left(t^k\right)^2}{gz + \sigma_{\text{additive}}^2}, & \text{if } Z_{\min} \le z \le Z_{\max} \\ 0, & \text{otherwise} \end{cases} . \tag{15}$$

Compare the resulting image (after tonemapping) with the best result you obtained in Part 4. Which parts of your image did the noise calibration make the biggest difference at?

6. Bonus: HDR by varying both shutter speed and ISO (100 points)

As we discussed in class, we can compose HDR images using exposure brackets created by varying either the shutter speed, or ISO, or even both. When varying both shutter speed and ISO, one needs to answer two questions: First, how do I decide what combinations of shutter speed and ISO to use? Second, how do I merge the resulting exposure stack into an HDR image? In answering these two questions, it is important to take into account the different noise characteristics of these two mechanisms for controlling exposure.

Hasinoff et al. [2] provide a detailed analysis of this kind of mixed exposure bracketing. Read this paper and try to reproduce their algorithm for capturing and merging an exposure bracket where you vary both shutter speed and ISO. For full credit, you will need to capture two RAW exposure stacks of the same scene:

One where you only vary shutter speed, and another where you vary both ISO and shutter speed. Then, you should merge each of the two stacks into an HDR image, using the procedure described in Section 4.1 of the paper, which corresponds to the noise-optimal weights in Part 5. Finally, you will need to compare the results. For a fair comparison, the total capture time for both stacks should be (approximately) the same. You do not need to implement Section 4.2: You can use either the ISO and shutter speeds reported in the paper, or ones you come up with on your own.

Note that, in order to implement dark-frame subtraction and noise-optimal mixing in the case of varying ISO, you need to separately estimate the terms σ^2_{read} and σ^2_{ADC} in Equation (14). We previously combined these in $\sigma^2_{\text{additive}}$, but this is no longer sufficient when we use ISO to vary the gain g. For full credit, you will need to think of and implement a modified noise calibration procedure that allows you to estimate the two terms. But you can get partial credit by either approximating these terms with reasonable guesses (which you should justify in your write-up), or by searching for them online.

Deliverables

When submitting your solution, make sure to follow the homework submission guidelines available on the course website (http://graphics.cs.cmu.edu/courses/15-463/assignments/submission_guidelines.pdf). Your submitted solution should include the following:

- A PDF report explaining what you did for each problem, including answers to all questions asked throughout Parts 1-5, as well as any of the bonus problems you choose to do. The report should include any figures and intermediate results that you think may help. Make sure to include explanations of any issues that you may have run into that prevented you from fully solving the assignment, as this will help us determine partial credit. The report should also explain any additional image files you include in your solution (see below).

- All of your Python code, including code for the bonus problems, as well as a `README` file explaining how to use the code.

- The HDR images that you create in parts 1 (only the one you pick at the end), 2, 4, and 5, as well as at least one RAW and corresponding `.JPG` LDR image you capture in Part 4. You can also include additional image files, LDR or HDR, for various experiments (e.g., tonemapping with different values) other than your final ones, if you think they show something important.

- If you do Bonus Part 6: Include in your PDF report a detailed description of the parts of Hasinoff et al. [2] you implemented, any issues you ran into, and any approximations or other decisions you made in reproducing their algorithm. Additionally, include the two HDR images you create, and at least one RAW image you capture for this part.

Hints and Information

- Make sure to download and install the latest version of `dcraw`. In particular, the default version that comes in older Windows versions does not support the cameras used in this class, and will produce results with a strong purple hue.

- When working with the provided and captured exposure stacks, you will notice that your algorithms will be using *a lot of memory*. This is a common issue when processing photographs captured with modern cameras, due to the very large number of pixels these cameras have. At 24 Megapixels, the Nikon D3300 used for this assignment is at the mid-range of megapixels. Still, at this resolution, a 3-channel HDR image takes up more than 0.5 GB of memory.

  This has two implications. First, you should be careful about how many of these images you create in your Python code, as otherwise you run the risk of filling up your memory and crippling your computer. Second, when processing an image, you need to make sure you use vectorized code that processes all of its pixels in parallel, as trying to process all 25 million pixels one-by-one with a double `for` loop will take ages.

In particular, when performing HDR merging, note that Equations (5)- (6) can be applied to each of the k exposure images independently. Therefore, instead of loading the entire exposure stack at once, you can load its images and process them one by one. Additionally, within each image, Equations (5)- (6) apply to each pixel in a completely parallel way. Therefore, you can process each image with a single vectorized call, instead of a double `for` loop.

One place where, no matter how careful you are, you will run out of memory is when solving the linear system (3) to recover the non-linear map g. As suggested by Debevec and Malik [1], you should greatly downsample the input images before forming the linear system. Note that you should *not* resize the image with `skimage.transform.resize`, or try to blur it before downsampling. For the purposes of inferring g, all you have to do is downsample an input image `I` with `I[::N, ::N]`, for some N. We recommend using $N = 200$.

More generally, when you are still debugging your code, we strongly recommend that you work on downsampled images to accelerate the development process. Once you know your code is correct, you can run it one more time on the full-resolution image, to produce your final results.

- When merging many LDR images to HDR ones, you may end up with pixels for which there are not any well-exposed values (i.e., the sum of weights in the denominators of Equations (5)- (6) is exactly 0). You can set those pixels to equal the maximum or minimum valid pixel value of your HDR image, respectively for problematic pixels that are always over-exposed or always under-exposed.

- Even with tonemapping, your images may appear too dark. In practice, after tonemapping, you still need to apply gamma encoding for images to be displayed correctly. As a reminder from Homework Assignment 1, gamma encoding is the following non-linear operator:

$$C_{\text{non-linear}} = \begin{cases} 12.92 \cdot C_{\text{linear}}, & C_{\text{linear}} \leq 0.0031308 \\ (1 + 0.055) \cdot C_{\text{linear}}^{\frac{1}{2.4}} - 0.055, & C_{\text{linear}} > 0.0031308 \end{cases} \tag{16}$$

  You should implement this in a script, and use it to always apply gamma encoding to tonemapped or HDR images before displaying them. Applying gamma encoding will also be helpful for displaying intermediate results (see below).

- As in Homework Assignment 1, you will likely find it helpful to display intermediate results. If you directly display the HDR images you create, they may appear very bright (potentially fully-white) or very dark (potentially fully-black). This is *not* a problem: as we discussed in class, HDR images are linear with respect to incident flux, but are scaled by a (somewhat) arbitrary scaling factor. All you have to do is multiply your image with an appropriate scaling factor of your own (smaller than 1 if the image is very bright, larger than 1 otherwise), apply gamma encoding, and then use the `clip` and `imshow` functions as in Homework Assignment 1. You will likely need to experiment with a few different values for the scaling factor you apply, until you find the one that correctly exposes your image.

- If you want to view the `.HDR` files you create, note that you *cannot* do so using a standard image viewer. Instead, you should use a dedicated viewer for `.HDR` files, such as `OpenHDR`. This viewer provides interactive sliders for controlling exposure (the scaling factor you apply to the image) and gamma encoding, making it easier to find good settings for examining your HDR image. Alternatively, you can use the function `readHDR` in the code we provide to open and load the `.HDR` in Python, then display it as we describe in the previous step.

- When applying photographic tonemapping to each RGB channel separately, you may get better results by using the same scalars $I_{m,\text{HDR}}$ and $\tilde{I}_{\text{white}}$ for all three channels. You can do this by using pixels from all three channels in Equations (9) and (11).

  Additionally, evaluating Equation (11) as written (i.e., by first computing the average of logarithms, and then exponentiating) may result in zero, NaN, or Inf values due to finite numerical precision. You may get more stable results by recognizing that Equation (11) is equivalent to computing the geometric mean of all pixels $I_{i,j,\text{HDR}}$, and changing your implementation accordingly. For more information about this type of numerical issues, look up "log-average form of geometric mean".

- When using your camera to capture your own exposure stack and noise calibration data, you should make sure to set its white balancing option to a fixed preset, as appropriate for the lighting in the scene you selected, and its output color space to sRGB. Additionally, if your camera supports this, set it to store both RAW and `.JPG` files for each image you capture (the Nikon D3300 has this option). That way, you will have perfectly paired RAW and `.JPG` exposure stacks, and you can use them to compare doing HDR with one or the other.

- While capturing your exposure stack and noise calibration data, it is critical that no camera parameters other than shutter speed change. Therefore, you should set the camera to manual mode, and disable auto-focus for the duration of the capture. If you do not take these steps, then the camera may automatically change parameters such as aperture, ISO, and focus, making your data unusable.

  Regarding aperture, you should use an aperture setting that gives you good depth of field for the scene you selected for your exposure stack.

  Regarding ISO, you should use a low ISO setting, as that makes noise calibration more reliable. In particular, make sure that your camera does not have a high ISO setting as a leftover from Homework Assignment 1, when you were building the pinhole camera.

  Regarding focusing, you can use autofocus while framing the scene you will use for your exposure stack, to make sure that your captured images will be sharp. Once the lens has been focused, you can then disable autofocus, switch to manual, and start capturing your exposure stack. You can use the same procedure to focus the lens before capturing your noise calibration data. It is fine if you need to change the lens focus between capturing the exposure stack and noise calibration data; but you should make sure to use the same settings for everything else besides focus and shutter speed (ISO, aperture, white balancing).

  Lastly, when capturing noise calibration data, you should use a shutter speed that is lower than what auto-exposure would select for the calibration target—that is, make your images somewhat darker than ideal.

- As discussed both in class and above in the assignment, it is very important that both your camera and your scene remain static while capturing your exposure stack. Given this, we strongly recommend that you mount your camera on a tripod, or at the very least on a very stable surface (e.g., a table) when taking images.

  While capturing your exposure stack, you will need to adjust the camera's shutter speed several times. Doing this manually requires touching the camera to rotate the shutter speed dial. You will also need to activate the shutter release, which means further touching the camera and pressing buttons. All of these manual actions can result in considerable camera movement, and therefore in your captured LDR images being misaligned. Using a tripod does not protect you from this type of camera motion.

  Therefore, we strongly recommend that you *tether*, i.e., connect, the camera to your laptop, so that you can control its settings and shutter release electronically, without touching the camera. Each of the class cameras comes with a USB cable you can use for this purpose.

  To control the camera, you can try using the software provided by each manufacturer on their website (here is the corresponding Nikon page for the class camera).

  As an alternative, we recommend that you try gphoto2. This is a very powerful command-line tool that can be used to script your camera and implement very complicated capture procedures. For example, the following lines auto-detect a connected camera, capture an image at shutter speed 1/2048, and then download the images from the camera to your computer and store them with filename `exposure1`. If your camera is set to capture both RAW and `.JPG`, this excerpt will download both images and store them as `exposure1.nef` and `exposure.jpg`, respectively.

```
gphoto2 --auto-detect
gphoto2 --set-config-value /main/capturesettings/shutterspeed=1/2048
gphoto2 --capture-image-and-download --filename exposure1.%C
```

If you are using a Nikon D3500 camera, please note there is a known issue with gphoto2 that results in the camera failing after a few capture commands. If you face this issue, you can solve it by killing the gphoto2 process (by pressing "Ctrl+C" twice) and running `gphoto2 --reset`. Please refer to the corresponding issue page for more details.

- When processing the data for noise calibration, it can help to remove high mean values. As at high flux regions values Poisson noise becomes dominant, high mean values have very high variance and may produce strong outliers. These outliers may skew the line you are fitting towards having a negative intercept. If that happens, you are welcome to reduce the range of mean values you use for your fit to just the lower mean values, which are more reliable for additive noise estimation.
- When performing noise calibration, you should make sure that you do not have any fluorescent lamps lighting your scene. The light output of these lamps varies with time, albeit at very high frequencies that we cannot perceive. This temporal variation may invalidate your noise calibration results.
- When using your noise calibration results to do merging with optimal weights, you should make sure to adjust gain and variance values, as appropriate to account for differences in how you normalize image data in different parts of the homework. Additionally, due to white balancing, noise calibration results will be different for each color channel, so you should make sure to use the correct values for each channel.

Credits

Some inspiration for this assignment and the write-up came from James Hays' and Oliver Cossairt's computational photography courses at Brown University and Northwestern University, respectively. The code for reading ground-truth color checker RGB values is from Ivo Ihrke's color calibration toolbox.

References

[1] P. E. Debevec and J. Malik. Recovering high dynamic range radiance maps from photographs. In *Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '97, pages 369–378, New York, NY, USA, 1997. ACM Press/Addison-Wesley Publishing Co.

[2] S. W. Hasinoff, F. Durand, and W. T. Freeman. Noise-optimal capture for high dynamic range photography. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pages 553–560. IEEE, 2010.

[3] K. Kirk and H. J. Andersen. Noise characterization of weighting schemes for combination of multiple exposures. In *BMVC*, volume 3, pages 1129–1138, 2006.

[4] E. Reinhard, M. Stark, P. Shirley, and J. Ferwerda. Photographic tone reproduction for digital images. In *Proceedings of the 29th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '02, pages 267–276, New York, NY, USA, 2002. ACM.