Homework Assignment 2

15-463/663/862, Computational Photography, Fall 2018
Carnegie Mellon University

Due Friday, Sep. 28, at 11:59pm

The purpose of this assignment is to explore high dynamic range (HDR) imaging 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. Tonemapping is the process of compressing 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, given that creating an accurate HDR image involves correctly capturing color, you can also do color calibration for extra credit.

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, it is highly recommended 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.

1. HDR imaging (70 points)

For this and the following problem (tonemapping), you will use an exposure stack we captured in Yannis' office using one of the class cameras (Nikon D3300). The image files are contained 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.

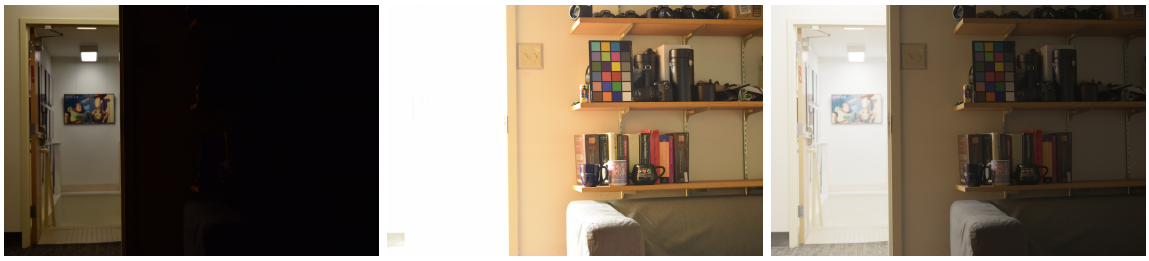

Figure 1: From left to right: Two LDR exposures, and an HDR composite tonemapped using the photographic tonemapping.

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 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, both exposure stacks are captured 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 value I_{ij}^k at pixel $\{i, j\}$ of image k relates to some unknown scene radiance 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. Since the domain of g are the discrete intensity values $\{0, \ldots, 255\}$, g is basically 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,

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

As we discussed in class, the weights w have to do with the fact that the linear estimates shoud 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 will use. The second term in Equation (2) has to do with the fact that we expect g to be smooth, and therefore we penalize 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, $\nabla^2 g\left(z\right) = g\left(z+1\right) - 2g\left(z\right) + g\left(z-1\right)$.

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 sparse matrix, $\boldsymbol{v} = [g; L_{ij}]$ are the unknowns, and $\boldsymbol{b}$ is a known vector. Then, use one of Matlab's solvers to recover the unknowns. (See help for the backslash operator \.)

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 of the 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. Linear merging is motivated by physical accuracy, while logarithmic merging, as mentioned above, is motivated by human visual perception.

When using linear merging, the HDR image is formed as:

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

When using logarithmic merging, the HDR image is formed as:

$$I_{ij,\mathrm{HDR}} = \exp\left(\frac{\sum_k w\left(I^k_{ij,\mathrm{LDR}}\right)\left(\log I^k_{ij,\mathrm{lin}} - \log\left(t^k\right)\right)}{\sum_k w\left(I^k_{ij,\mathrm{LDR}}\right)}\right). \tag{6}$$

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. Note that we use $I^k_{ij,\mathrm{LDR}}$ to refer to the pixel values of the original LDR image, which may be non-linear (when using `.JPG` files) or linear (when using `.TIFF` from RAW files).

Implement both linear and logarithmic merging for each of the two exposure stacks. Then, store the resulting HDR images as `.HDR` files. (See help for the `hdrwrite`.)

Weighting schemes. There are many possible weighting choices [4]. You will implement the following three:

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

All of the above weighting scehems assume that the intensity values are in the range $z \in [0, 1]$. So make sure to normalize your LDR images to that range (e.g., divide by 255 if they come from `.JPG` files). You can experiment with different clipping values $Z_{\min}$ and $Z_{\max}$, but we recommend $Z_{\min} = 0.01, Z_{\max} = 0.99$.

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 of the above weighting schemes, and use them to create HDR images. In total, you will create 12 HDR images: 2 sets of images (RAW and rendered) $\times$ 2 merging scehems (linear and logarithmic) $\times$ 3 weighting schemes (uniform, tent, and Gaussian).

Evaluation (10 points). One way to evaluate the results of the HDR creation process is to check its linearity. For this, you can use the color checker (Figure 2). In particular, patches 4, 8, 12, 16, 20, and 24 of the color checker are created to be near-perfectly neutral (gray) patches, with the reflectance of each patch being twice as high as the reflectance of the patch above it. Therefore, in an ideal HDR image, plotting the logarithm of the average *luminance* in each of these patches should produce a straight line.

For each of the neutral patches, crop a square that is fully contained within the patch and which can be used to compute patch averages. (See help for functions `ginput` and `impixelinfo`.) Make sure to store the coordinates of these cropped squares, so that you can re-use them.

Then, for each HDR image you create, evaluate its linearity as follows: First, convert the HDR image to the XYZ color space, and extract the Y channel (luminance). (See help for function `rgb2xyz` and make sure to use `'Colorspace'` option `'linear-rgb'`.) Second, using the cropped squares you created earlier, compute the average luminance for each of the six neutral patches. Third, perform linear regression to the *logarithms* of these six average luminances. (See again help for the backslash operator \.) Fourth and final, compute the least-squares error between the actual average luminance values and the linear fit you created.

Compute this error for each HDR image you create, and discuss how they compare to each other. Additionally, make a logarithmic plot comparing the six average luminances for the various HDR images.

Figure 2: Color checker and patch numbering.

Make your pick. Select two out of the twelve HDR images you created, one created using RAW and one created using `.JPG` LDR images. You can select either the images that have the smallest linearity error, as computed above, or the ones that you find the most aesthetically pleasing. If the images you select do not match the best in terms of linearity, comment on the differences and on why you selected the ones you did. Note that, since you have not yet tonemapped your HDR images, if you display them directly they will not appear correctly. To get a sense of what each HDR image looks like and select your favorites, before displaying them, you will need to first apply some appropriate scale to them and then gamma-correct them.

2. Color correction and white balancing (20 points)

Figure 3: Tonemapped HDR image without (left) and with (right) color correction.

For this part, you will use the two HDR images you selected from Part 1.

As shown to the left of Figure 3, your tonemapped images will tend to have an orange casts 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 Assignment 1 to ameliorate the issue. But, given that the images include a color checker, it is possible to do better than that and perform accurate color correction.

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` directory of the homework ZIP archive, returns these ground-truth RGB coordinate values, with the patches numbered as shown in Figure 2.

Then, in order to have your HDR images 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. Create 24 square crops, similar to what you did for the neutral color checker patches in Part 1. Use them to compute average RGB coordinates for each of the color checker's 24 patches.
2. Convert these computed RGB coordinates and their corresponding ground-truth values 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.
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 exactly equal to each other. This is analogous to the manual white balancing in 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 four HDR images total: The two from Part 1 that have not been color-corrected, and the two you just created. Compare the color-corrected images with the originals, and discuss which one you like the best.

3. Tonemapping (60 points)

Now that you have a bunch of HDR images, you need to tonemap them so that you can display them. You will implement two different tonemapping algorithms. For this part, you can use whichever of the four HDR images at the end of Part 2 you liked the best.

Photographic tonemapping (20 points). First, you will implement the tonemapping operator proposed by Reinhard et al. [5], 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

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

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

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

(Note that Equation (11) is different from the corresponding Equation (1) in Reinhard et al. [5]. The version given here is the correct, and the version in the paper is incorrect.) 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.

Implement the photographic operator and apply it to your RGB HDR images in two ways: First, apply it to each color channel separately. Second, apply it only to the luninance channel Y. For the latter, you can use `rgb2xyz` to convert the HDR image from RGB to XYZ, then convert the image from XYZ to xyY as we discussed in class. While in xyY, tonemap the luminance Y while leaving the chromaticities x,y untouched. Then, invert the color transforms to go back to RGB.

Experiment with different key and burn values. Some reasonable starting values for the parameterers 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.

Tonemapping using bilateral filtering (40 points). You will now experiment with tonemapping using bilateral filtering, as proposed by Durand and Dorsey [2]. As in the previous problems, you are encouraged to read through the paper, to find many implementation details and insights about the various parameters.

Given pixel values $I_{ij,\mathrm{HDR}}$ of a linear HDR image, tonemapping using bilateral filtering is performed through a sequence of steps:

1. Compute the log intensity, $L_{ij} = \log\left(I_{ij,\mathrm{HDR}}\right)$.
2. Compute the base layer using bilateral filtering, $B_{ij} = \mathrm{bilateral\_filter}\left(L_{ij}\right)$.
3. Compute the detail layer, $D_{ij} = L_{ij} - B_{ij}$.
4. Apply an offset and a scale S to the base, $B'_{ij} = S \cdot \left(B_{ij} - \max_{i,j}\left(B_{ij}\right)\right)$.
5. Reconstruct the intensity, $I_{ij,\mathrm{TM}} = \exp\left(B'_{ij} + D_{ij}\right)$.

The scale S determines the number of stops in the dynamic range of the final tonemap (e.g., 2-4 stops). For bilateral filtering, you can use any of the algorithms suggested in Section 5.1 of Durand and Dorsey [2].

As with photographic tonemapping, apply bilateral filtering both to each RGB channel separately and to luminance only. Experiment with different values of scale S and the bilateral filter parameters. Plot representative tonemaps for both the RGB and luminance methods, and discuss your results. Make sure to mention which tonemap you like the most, as well as compare with the results from photographic tonemapping.

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

Now it is 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.

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 colorchecker, you can skip the color calibration step.

Then, process these images using the tonemapping algorithms you implemented in Part 3 (photographic or bilateral, 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. 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 to I merge the resulting exposure stack into an HDR images? 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. [3] 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 both shutter speed and ISO is varied. 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, and 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 on your own. Finally, you will notice that in several places you will need to know camera-specific parameters. You can try to either approximate those yourselves with reasonable guesses (which you should justify in your write-up), or search for them online.

Deliverables

As described on the course website, solutions are submitted through Canvas. Your solution should be an archive (e.g., a ZIP file) that includes the following:

- A PDF report explaining what you did for each problem, including answers to all questions asked throughout Parts 1-4, 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 Matlab code, including code for the bonus problems, as well as a `README` file explaining how to use the code.
- HDR images that you create in parts 1, 2, and 4, 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 5: Include in your PDF report a detailed description of the parts of Hasinoff et al. [3] you implemented, 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.

Please organize your solution submission using the following file structure:

```
.zip
├── .pdf ............................................................ The PDF report.
├── src/ ............... Contains all Matlab M-files and the README file explaining how to use the code.
└── data/ ........................................... Contains all image, video, and other data files.
```

Hints and Information

- 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 very cautious about how many of these images you create in your Matlab 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 of them 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 in [1], you should greatly downsample the input images before forming the linear system. Note that you should *not* resize the image with `imresize`, 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(1:N:end, 1:N:end)`, for some N. We recommend using $N = 200$.

- 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 correction for images to be displayed correctly. As a reminder from Assignment 1, gamma correction is the following non-linear operator:

$$C_{\text{non-linear}} = \begin{cases} 12.92 \cdot C_{\text{linear}}, & C_{\text{linear}} \leq 0.0031308 \\ (1 + 0.055) \cdot C_{\text{linear}}^{\frac{1}{2.4}} - 0.055, & C_{\text{linear}} \geq 0.0031308 \end{cases} \tag{12}$$

  You should implement this in a script, and use it to always apply gamma correction to tonemapped or HDR images before displaying them.

- When using your camera to capture your own exposure stack, you should make sure to set its white balancing option to AUTO, 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, it is critical that no camera parameter other than shutter speed changes. 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 parameters such as aperture, ISO, and focus may be automatically changed by the camera, making your captured exposure stack unusable.

  In practice, you can use autofocus while framing your scene, 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.

- As discussed both in class and above in the assignment, it is very important that both your camera and your scene remains 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
```

Credits

Some inspiration for this assignment and the write-up came from James Hays' computational photography course at Brown University. 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] F. Durand and J. Dorsey. Fast bilateral filtering for the display of high-dynamic-range images. In *Proceedings of the 29th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '02, pages 257–266, New York, NY, USA, 2002. ACM.

[3] 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.

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

[5] 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.