MIT EECS 6.815/6.865: Assignment 5: High-Dynamic-Range Imaging and Tone Mapping

Due Monday March 12 at 9pm

1 Summary

- merge N bracketed images into an HDR image
- tone mapping with Gaussian blur
- tone mapping with bilateral grid

6.865 students get a break this week. No extra work.

2 HDR merging

In this assignment, we will only work with images that are linear, static , and where the camera hasn't moved. The image files are provided with gamma 2.2 encoding but `imread` will do the right thing, decode gamma and provide you with linear values.

Consult the course slides for the overall merging approach. We will first compute weights for each pixel and each channel to eliminate values that are too high and too low. We will then compute the scale factor between the values of two images. Finally, we will merge a sequence of images using a weighted combination of the individual images according to the weights and scale factors.

2.1 Weights

Write a function `computeWeight(im, epsilonMini=0.002, epsilonMaxi=0.99)` that returns an image that is equal to 1.0 when the corresponding pixel value of `im` is between `epsilonMini` and `epsilonMaxi`, and 0.0 otherwise.

Use `numpy`'s logical indexing to avoid loops. See the lecture slides.

2.2 Factor

Now that we know which pixels are usable in each picture, we can compute the multiplication factor between a pair of images. Write a function `computeFactor(im1, w1, im2, w2)` that takes two images and their weights computed according to the above method, and returns a single scalar corresponding to the median value of `im2/im1` for pixels that are usable in both `im1` and `im2`.

Again, we encourage the heavy use of numpy methods. A combination of logical indexing, `flatten` and the built-in `sort` function should help.

2.3 Merge to HDR

Use the above functions to merge a sequence of images. You can assume that the first one is the darkest one, and that they are given in order of increasing exposure.

Write a function `makeHDR(imageList, epsilonMini=0.01, epsilonMaxi=0.99)` that takes a sequence of images as input and returns a single HDR image.

Do not forget the special case for the computation of the weight of the darkest and brightest images. You should only threshold in one direction.

To test your method, you can write the output scaled by different factors.

If you want to save your HDR images, do not use imread because it would quantize them to 8 bits per channel, which defeats the purpose. You can use the `save` and `load` functions from `numpy`.
See e.g. `http://docs.scipy.org/doc/numpy/reference/generated/numpy.save.html`.

3 Tone mapping

Now that you have assembled HDR images, you are going to implement tone mapping. Your tone mapper will follow the method studied in class. The function will be called `toneMap(im, targetBase=100, detailAmp=3, useBila=False)`. It takes as input a HDR image, a target contrast for the base, an amplification factor for the detail, and a Boolean for the use of the bilateral filter vs. Gaussian blur.

First decompose your image into luminance and chrominance using your code from assignment 1. That is, compute a black-and-white version of your image to get the luminance, and deduce the chrominance by normalizing the original pixels by this luminance. The luminance we used for problem set 1 has three channels, which is wasteful, but we won't worry about it.

Next, compute a log version of the luminance. Add a small constant to the luminance to avoid problems at 0. Use numpy tricks to extra the smallest non-zero value and set the zero pixels to it. No loop allowed. Stick to logical indexing, `flatten` and `min`.

You are ready to compute the *base* We will implement two versions: Gaussian blur and bilateral filtering, which will be chosen based on the value of the input parameter `useBila`. In both cases, we will use a standard deviation for the spatial Gaussian equal to the biggest dimension of the image divided by 50. The range standard deviation for the bilateral filter should be 0.4 (in log10).

You are welcome to use your own implementation of these methods, but you can also faster versions. For Gaussian blur, you can use `scipy`'s method `ndimage.filters.gaussian_filter`. Be careful: you should pass it one standard deviation per dimension, and specify that the channel dimension shouldn't be blurred.

For fast bilateral filtering, we provide you with an implementation of the bilateral grid described at
`http://groups.csail.mit.edu/graphics/bilagrid/`

with more mathematical justifications at
`http://people.csail.mit.edu/sparis/publi/2009/ijcv/Paris_09_Fast_Approximation.pdf`

Create a bilateral grid object using `bila=bilaGrid(ref, sigmaS, sigmaR)`, where `ref` is a 1-channel image. In your case, you can just pass one of the channels of your log luminance. Once you have created the grid, use it to perform bilateral filter by calling `out=bila.doit(im)` where im is now a 3-channel image, the full 3-channel version of the luminance.

Given the base, compute the detail by taking the difference from the original log luminance.

You are finally ready to put everything together. Compute the scale factor in the log domain that brings the dynamic range of the base to the given target (that is, the range in the log domain should be `log10(targetBase)`. Make sure to add an offset that ensures that the highest base value is 1 once mapped back to the linear domain[1]. Multiply the details (in the log domain) by `detailAmp`, add the resulting output base and detail, and put them back to the linear domain. This should give you an output luminance, and reintegrating the chrominance should be easy.

Enjoy your results and compare the bilateral version with the Gaussian one. We recommend the following parameters:

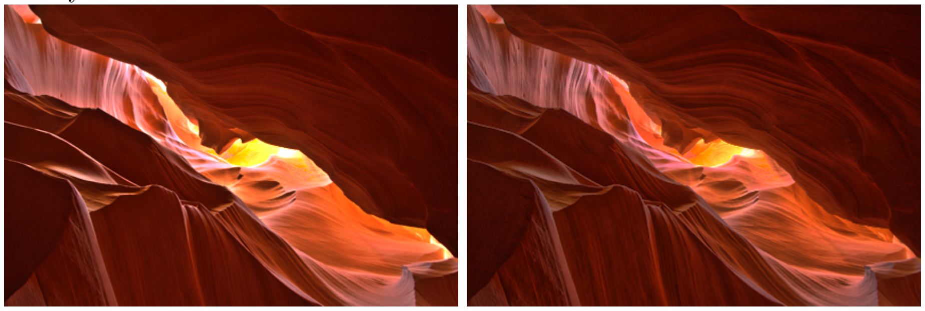
```
toneMap(im, 100, 1, False)
toneMap(im, 100, 3, True)
```

Although you might want to tone down the detail amplification for the `vine` image.

4 Putting it all together

Run your full pipeline (merging + tone mapping) on at least the `vine, design,` and `ante2` sequences. Use both the Gaussian and bilateral version.

My results on ant2 are below:



[1]Bell/whistle: add a parameter so that the user can choose to make it a little less than 1 to leave room for the detail.

5 Extra credits

Deal with image alignment (e.g. the `sea` images. We recommend Ward's median method, but probably single-scale to make life easier yet slower. Alternatively, you can simulate the clipping in the darker of two images.

Derive better weights by taking noise into account. You can focus on photon noise alone. This should give you an estimate of standard deviation (or something proportional to the standard deviation) for each pixel value in each image. Use a formula for the optimal combination as a function of variance to derive your weights. This should replace the thresolding for dark pixels, but you still need to set the weight to zero for pixels dangerously close to 1.0.

Write a function to calibrate the response curve of a camera. See `http://www.pauldebevec.com/Research/HDR/`

Figure out why the scale factors between images is not exactly the same for all three channels :-(I am personally not sure. It could be a white balance problem.

Hard: deal with moving objects.

6 Submission

Turn in your 6 images, python files, and make sure all your functions can be called from a module `a5.py`. Put everything into a zip file and upload to Stellar.

Include a `README.txt` containing the answer to the following questions:

- How long did the assignment take?
- Potential issues with your solution and explanation of partial completion (for partial credit)
- Any extra credit you may have implemented
- Collaboration acknowledgement (but again, you must write your own code)
- What was most unclear/difficult?
- What was most exciting?

Include tone mapping results for the three required sequences.