6.815/6.865 Digital & Computational Photography

Problem Set 2: Tone Mapping and Poisson Image Editing

Due Tuesday, March 11 at 7:00pm

Introduction

Sometimes we want to stretch the limits of traditional photography. Modern image editing programs provide a number of tools that allow users to create fantastic imagery. In this assignment, we will be exploring two cutting-edge techniques that have barely had the time to make it into commercial software products.

As with the previous problem set, make sure you check the labels for each problem, since some of them will be for 6.865 students only. Submission is the same as before: a ZIP file that include MATLAB code and a PDF with written answers and supporting images.

Tone Mapping

Human visual systems are remarkably adept at seeing large differences in brightness. We can sit in a classroom on a sunny day and see everything, from the bored looks on the faces of our classmates to the fluffy clouds outside. To take a photograph of this particular scene would result in disaster, as digital sensors simply do not have the ability to capture such large variations in brightness.

There are a number of ways to *capture* the high dynamic ranges common in natural scenes, but it is still necessary to *display* them. Computer monitors aren't particularly good at this, and printed images are even worse. Thus, it's often necessary to compress the dynamic range of the image.

Tone Mapping describes a set of techniques that does just that. Here, you will implement a simplified version of Durand and Dorsey's 2002 algorithm: `http://people.csail.mit.edu/fredo/PUBLI/Siggraph2002/`. This algorithm works by reducing the large-scale variation of luminance while preserving local detail. It does this using the bilateral filter on log-domain luminance values.

The bilateral filter blurs an image while preserving strong edges. One can think of it as an extension of the standard Gaussian blur filter, except with additional terms to handle edges. You can see a similar effect in the latest version of Photoshop as the "Surface Blur" filter. The latest version of `dcraw`, which you may have experimented with in the previous assignment, also includes a bilateral filtering option for smoothing high-ISO images.

Mathematically speaking, the output of the bilateral filter computes the output value of a pixel s as a weighted average of its neighboring pixels $p \in N_s$, where the weights depend on both the spatial distance and intensity difference:

$$J_s = \frac{1}{k(s)} \sum_{p \in N_s} \exp\left\{ -\frac{(p-s)^2}{2\sigma_d^2} \right\} \exp\left\{ -\frac{(I_p - I_s)^2}{2\sigma_r^2} \right\} I_p.$$

In this expression, s and p are coordinates of an image (like $[200, 250]$; I_s and I_p are the pixel intensities at those coordinates. There are three parameters to bilateral filtering:

- The width parameter w defines the size of the neighborhood N_s. Bilateral filtering looks at the pixels $(2w-1) \times (2w-1)$ square neighborhood centered at s.
- The spatial standard deviation σ_d defines the influence of neighboring pixels according their distance to s in the image lattice.
- The range standard deviation σ_r defines the influence of neighboring pixels according to their intensity difference from s.

Finally, the normalization term $k(s)$ is defined as follows:

$$k(s) = \sum_{p \in N_s} \exp\left\{ -\frac{(p-s)^2}{2\sigma_d^2} \right\} \exp\left\{ -\frac{(I_p - I_s)^2}{2\sigma_r^2} \right\}.$$

This just ensures that all the weights sum to 1.

The bilateral filter can be used for tone mapping using the following pseudocode (given input image channels `[R_in,G_in,B_in]`):

```
intensity_in = 1/61 * (20*R_in + 40*G_in + B_in)
log(base) =  bilateral(log(intensity_in))
log(detail) = log(intensity_in) - log(base)
compress_factor = log(range_out) / max(log(base) - min(log(base)))
log(offset) = -max(log(base)) * compress_factor
log(intensity_out) = log(base) * compress_factor + log(offset) + log(detail)
[r,g,b] =  [R_in,G_in,B_in] / intensity_in
[R_out,G_out,B_out] = [r,g,b] * exp(log(intensity_out))
```

The main parameter for this code is the output range (`range_out`) which defines the amount of remaining large-scale contrast desired in the output. In practice, values of 10 to 30 work well.

Problem 1 (6.815/6.865)

Implement a MATLAB function `bilateral(I,w,sigma_d,sigma_r)` where the parameters are as described before. Try, as much as possible, to avoid four nested `for`-loops, as they are quite inefficient in MATLAB). Two nested loops should be reasonable. Select a noisy grayscale image and experiment with parameters until you find numbers that produce good results. For efficiency, you will probably want to stick with relatively low-resolution images. You might select an photo you took taken at a high ISO value, or alternatively, you can use an image from the previous assignment with added noise.

In your writeup: Show before-and-after results of your experiments and note the parameters that you used. Specifically, display your result for the ideal values of σ_d and σ_r as well as some different values, and describe the qualitative effects of changing these parameters.

Problem 2 (6.815/6.865)

The supplemental materials for this assignment include an HDR image (`vinesunset.hdr`) as well as a function to load it into MATLAB (`read_rle_rgbe.m`[1]) Load the HDR image into MATLAB (you can also view it from Photoshop CS2 and perhaps earlier). Then, do the following:

- Implement tone mapping, as described earlier, using a Gaussian blur filter instead of bilateral filtering. Use a filter size of 21 and a standard deviation of 8:

  `imfilter(I,fspecial('gaussian',21,8))`.

  Use an output range of 30.

- Now, replace the Gaussian blur filter with the bilateral filter that you implemented for Problem 1. Use $w = 10$, $\sigma_d = 8$, and $\sigma_r = 0.2$.

You should implement these as two separate MATLAB functions `tonemap_gaussian` and `tonemap_bilateral` (yes, it's just a copy-and-paste job, but it'll make grading easier).

In your writeup: Display the results of tone mapping for both Gaussian and bilateral filtering. Describe the unsightly artifacts in the Gaussian-filtered version, and explain how the edge-preserving nature of the bilateral filter manages to resolve these issues.

Poisson Image Editing

Chances are that, at some point in your life, you've played around in Photoshop or some other image-editing software and tried to paste one part of an image into another. Perhaps you were trying to put your friend's head on the body of someone else who is doing something embarrassing. In any case, doing a naive copy-and-paste will often yield very unattractive results. Poisson Image Editing, as described by Pérez et al. at SIGGRAPH 2003, proposes one method to simplify this task. For this problem, begin by reading their paper: http://research.microsoft.com/vision/cambridge/papers/perez_siggraph03.pdf. 6.815 students can focus on Section 2 and the first part of Section 3.

Problem 3 (6.815/6.865)

Implement the image cloning technique, as described in Equations 7–11 of the Pérez et al. paper, and use it to paste `bear.bmp` (masked by `mask.bmp`) into `waterpool.bmp`. You can see the desired result in Figure 3 of the paper. For this problem, you don't have to write a general-purpose function. If it's easier, just do everything in a MATLAB script. Either way, name it `poisson.m`.

[1] Borrowed from http://www.cis.rit.edu/mcsl/icam/hdr/.

This is not easy! There are a lot of little details that you'll have to figure out, such as identifying boundary pixels given the mask, and so on. One of the biggest components of this problem is figuring out how to initialize and solve the large linear system that arises. We recommend that you use MATLAB sparse matrices (`help sparse`) and solve the system using the conjugate gradient method (`help cgs`).

In your writeup: Show the results of your code for pasting the bear image into the water image. Provide at least two images, one in which the bear is at the top, and one in which the bear is at the bottom.

Problem 4 (6.865 only)

The Pérez et al. paper describes a number of extensions to the basic technique that you implemented: texture flattening, local illumination/color changes, and seamless tiling (these are in Section 4). Implement one of these extensions. The local illumination technique may be of particular interest, as it has connections to an alternative tone mapping method to the one you implemented for this assignment[2].

In your writeup: State the extension that you chose to implement, briefly describe how you did it, and show your results.

Submission

Like the previous assignment, you should assemble a ZIP file that is named after your Athena login. Make sure this file contains:

- A PDF file with answers to your written questions and your results. *In general, you should try to make this file as self-sufficient as possible.* In other words, we shouldn't have to look at your code to evaluate your results. Please don't tell us to run something unless it's absolutely necessary. We will look at your code to make sure you did the work and assign partial credit if you did something wrong.
- Your MATLAB code (we've provided stub methods with the method signatures we expect for grading):
  - `bilateral.m`
  - `tonemap_gaussian.m`
  - `tonemap_bilateral.m`
  - `poisson.m`
  - Code for Problem 4 (6.865 only)
- Any images (other than the provided ones) that might be necessary to run your code.

All submissions are due on the Stellar website by March 11 at 7pm.

[2] http://www.cs.huji.ac.il/~danix/hdr/hdrc.pdf