6.815/6.865 Digital & Computational Photography

Problem Set 3: Poisson Image Editing

Due Friday, March 4 at 7:00pm

Introduction

Again in this assignment, we will be exploring a cutting-edge technique that allows users to create fantastic imagery and has barely had the time to make it into commercial software products.

As with the previous problem sets, 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 the previous problem set: a ZIP file that includes MATLAB code, and a PDF with written answers and supporting images.

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://www.irisa.fr/vista/Papers/2003_siggraph_perez.pdf`. This paper is also available under the materials section of the course website. 6.815 students can focus on Section 2 and the first part of Section 3.

Problem 1 (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`.

Hints

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`). This is faster than using dense matrices and e.g. the \ operator, but it also allows you not to worry as much about which pixels are known or unknown.

One challenge is to keep track of the indices of pixels between the original image representation (a 2D matrix) and the linearized form on which you will set up and solve your linear system (a vector). Another, related challenge is how to maintain neighborhood information: up/down/left/right neighbor relationships are obvious in a 2D grid, but not in a flattened vector (especially when it is sparse, only containing items within the mask). You may want to use arrays to track the correspondence between pixels in the source image coordinates and the elements in the linear system you set up, and to track the neighborhood relationships between elements.

Some useful functions to investigate:

- `sub2ind/ind2sub` - map back and forth between linear subscripts and nD array indices.
- `logical` - convert numerical values (e.g. the `mask.bmp`) to logical (true/false) values. You can also directly use logical selection expressions like `I(mask == 1)` to extract a flat vector of all elements in matrix `I` where `mask` is `1`.
- `spy` - visualize the sparsity pattern of a sparse matrix. This can be both fun and interesting!

A simple, inefficient way to implement the algorithm is to use for loops over the image, and treat *all* pixels as unknowns (with a trivial equation for pixels outside the mask). This makes neighborhood indexing easy, but it is slow.

A more efficient approach is to use logical selection (as described for `logical` above) to extract the unknown region into a vector. You can track indices by creating a 2D matrix of the same size as the image, containing the pixel coordinates or indices as its values, and then extracting the masked region of this into a parallel vector. You can also build indexing vectors for the 4 neighbors of each unknown pixel by shifting this indexing matrix by 1 in each direction (`help circshift`). Given this information, you should be able to set up and solve the desired linear system, and then map the results back into the output image matrix.

You may assume that the mask is entirely at least 1 pixel inside the image boundary, so no unknown pixels or their neighbors will be at undefined locations.

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 2 (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 the previous assignment[1].

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. More importantly, including some explanation about what your code does and how it does it will be very helpful in evaluating 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):
  - `poisson.m`
  - Code for Problem 2 (6.865 only)
- Any images (other than the provided ones) that might be necessary to run your code.

All submissions are due by March 4 at 7pm.

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