Mini-project 2

CMPSCI 670, Fall 2016, UMass Amherst
Due: October 14, 5:00 PM
Instructor: Subhransu Maji
TA: Tsung-Yu Lin

Guidelines

Submission. Submit a *single pdf document* via moodle that includes your solutions, figures and printouts of code. For readability you may attach the code printouts at the end of the solutions. You could have 24 hours late submission with a 50% mark down. Late submission beyond 24 hours will not be given *any* credits.

Plagiarism. We might reuse problem set questions from previous years, covered by papers and webpages, we expect the students not to copy, refer to, or look at the solutions in preparing their answers. We expect students to want to learn and not google for answers.

Collaboration. The homework must be done individually, except where otherwise noted in the assignments. 'Individually' means each student must hand in their own answers, and each student must write their own code in the programming part of the assignment. It is acceptable, however, for students to collaborate in figuring out answers and helping each other solve the problems. We will be assuming that you will be taking the responsibility to make sure you personally understand the solution to any work arising from such a collaboration.

Using other programming languages. All of the starter code is in Matlab which is what we expect you to use. You are free to use other languages such as Octave or Python with the caveat that we won't be able to answer or debug non Matlab questions.

1 Hybrid image [20 points]

A hybrid image is a combination of a low-pass filtered (i.e. blurry) image and a high-pass filtered (i.e. sharp) image. Recall that one can obtain a sharp image by subtracting the blurry version of an image from itself. Mathematically this can be written as $I = blurry(I) + sharp(I)$. Thus a hybrid image of I_1 and I_2 can be obtained as:

$$I_{hybrid} = blurry(I_1, \sigma_1) + sharp(I_2, \sigma_2) = I_1 * g(\sigma_1) + I_2 - I_2 * g(\sigma_2). \quad (1)$$

Here, $g(\sigma_1)$ and $g(\sigma_2)$ are Gaussian filters with standard deviations σ_1 and σ_2 and * denotes the filtering operator. Figure 1 shows the result of filtering with Gaussians of two different σ values.

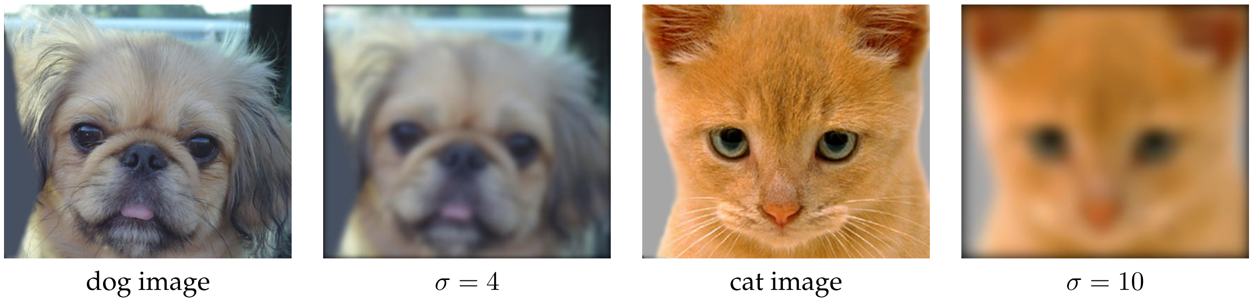

Figure 1: Effect of filtering with a Gaussian. The bigger the sigma the more blurry it is.

Implement the function `hybridImage(im1, im2, sigma1, sigma2)` that computes the hybrid image using Equation 1. Use your code to generate at least one example of a hybrid image (see examples here `http://cvcl.mit.edu/hybrid_gallery/gallery.html`). You will have to tune the `sigma1` and `sigma2` to make it work on specific images. In order to visualize the hybrid image it is sometimes useful to show multiple copies of the image at different resolutions. You can use the function `vis_hybrid_image.m` included in `p2_code.zip` which can be used to create such a figure. Include the code, the two source images, the hybrid image displayed in the format below, as well as the parameters that worked for you. *The three most creative submissions will be given a prize.*

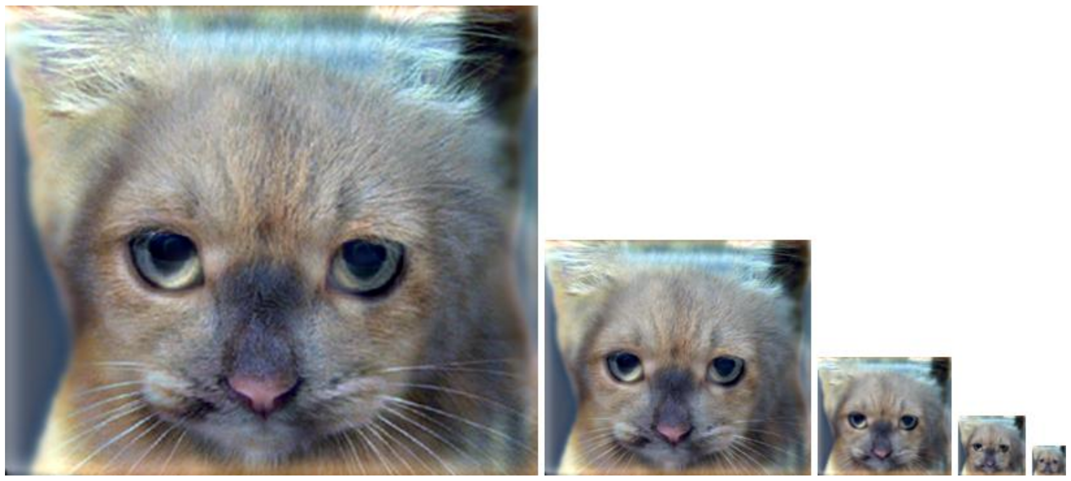

Figure 2: Hybid image of the dog and cat. The large image looks like the cat while the small image looks like the dog. The image was created with $\sigma_1 = 4$ and $\sigma_2 = 10$.

2 Photometric stereo [50 points]

In this part you will implement a basic shape from shading algorithm as described in Lecture 5. This is also described in shape from shading section (Sec 2.2) in Forsyth and Ponce book (pdf link for this section). The input to the algorithm is a set of photographs taken in known lighting directions and the output of the algorithm is the albedo (paint), normal direction, and the height map (Figure 3).

If you haven't done so already, download the `p2_code.zip` and `p2_data.zip` files from moodle. The data directory consists of 64 images each of four subjects from the Yale Face database. The light source directions are encoded in the file names.

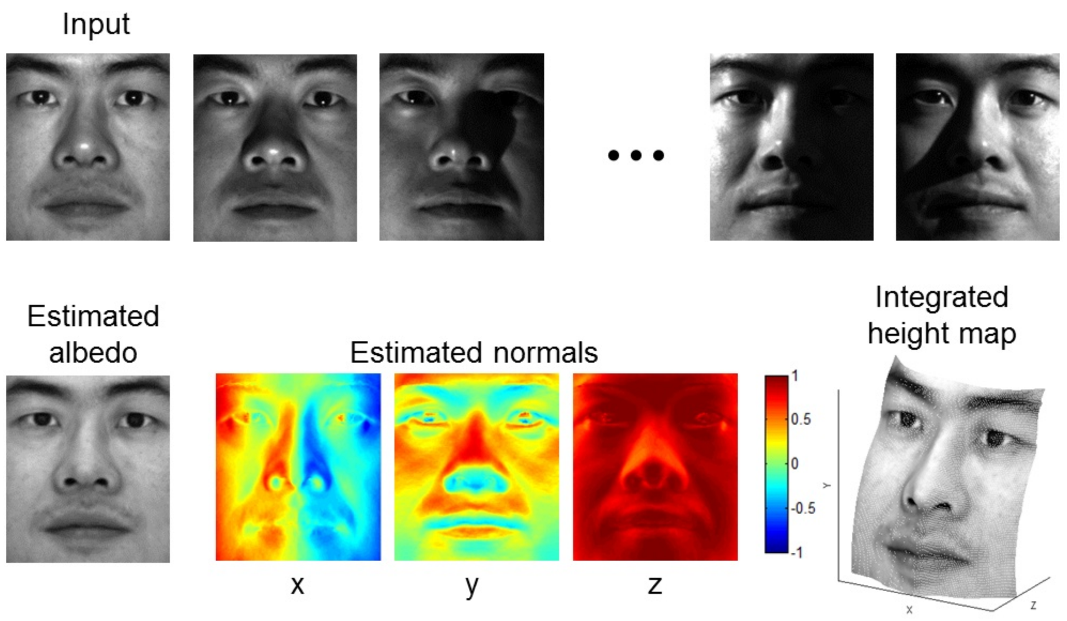

Figure 3: Top row: Input images. Bottom row: Estimated albedo, normals, and depth map

Below is an outline of the steps you need to implement the functions. The entry code for this part is `evalCode.m`. You will add code to files `prepareData.m`, `photometricStereo.m` and `getSurface.m` as described below. The remaining files are for processing the input data and displaying the output.

1. **Read images and light sources.** For each subject (sub-directory in the `data` folder), read in the images and light source directions. This is accomplished by the function `loadFaceImages.m` (which, in turn, calls `getpgmraw.m` to read the PGM files). The function `loadFaceImages()` returns the images for the 64 light source directions, an ambient image (i.e., image taken with all the light sources turned off), and the light directions for each image.

2. **[5 points] Preprocess the data.** We need the images under the illumination of only the point light source. Fortunately, due to linearity of light we can do this by subtracting the ambient image from all the images. Also, set any negative values to zero and rescale the resulting intensities to between 0 and 1 by dividing by 255. Implement this in the `prepareData.m` file.

3. **[20 points] Estimate the albedo and normals.** Implement the function in `photometricStereo.m` which takes as input the stack of images and the matrix of the light source directions, and returns an albedo image and normal directions. The normal directions should be encoded as a three dimensional array of size $h \times w \times 3$ where the third dimension corresponds to the x-, y-, and z-components of the normal. To solve for the albedo and the normals, you will need to set up a linear system as shown in slide 80 of Lecture 5.

   To get the least-squares solution of a linear system, use MATLAB's backslash operator. That is, the solution to `Ax = b` is given by `x = A\b`. If you directly implement the formulation of slide 80 of the lecture, you will have to loop over every image pixel and separately solve a linear system in each iteration. There is a way to get all the solutions at once by stacking the unknown g vectors for every pixel into a $3 \times n$ pixels matrix and getting all the solutions with a single application of the backslash operator. You will most likely need to reshape your data in various ways before and after solving the linear system. Useful MATLAB functions for this include `reshape` and `cat`. You may also need to use element-wise operations. For example, for two equal-size matrices $\mathbf{X}$ and $\mathbf{Y}$, $\mathbf{X}.{*}\ \mathbf{Y}$ multiplies corresponding elements, and $\mathbf{X}.\hat{}2$ squares every element. As before, `bsxfun()` can also be a very useful function here.

4. **[20 points] Compute the surface height map by integration.** The method is shown in slide 83 of Lecture 5, except that instead of continuous integration of the partial derivatives over a path, you will simply be summing their discrete values. Your code implementing the integration should go in the `getSurface.m` file. As stated in the slide, to get the best results, you should compute integrals over multiple paths and average the results. You should implement the following variants of integration:
   - Integrating first the rows, then the columns. That is, your path first goes along the same row as the pixel along the top, and then goes vertically down to the pixel. It is possible to implement this without nested loops using the `cumsum()` function.
   - Integrating first along the columns, then the rows.
   - Average of the first two options.
   - Average of multiple random paths. For this, it is fine to use nested loops. You should determine the number of paths experimentally.
5. Display the results using `displayOutput` and `plotSurfaceNormals` functions.

Hint: You can start by setting the `subjectName='debug'` which creates images from a toy scene with known geometry and albedo. You can debug your code against this before you try the faces.

2.a What to submit

To get full credit for this part you have to

- Include your implementation of `prepareData.m`, `photometricStereo.m`, and `getSurface.m`. Also, include the visualization of albedo image, estimated surface normals, and recovered surface of four subjects. It is sufficient to simply show the output of your best method. For the 3D screenshots, be sure to choose a viewpoint that makes the structure as clear as possible (and/or feel free to include screenshots from multiple viewpoints)
- **[5 points]** Discuss the differences between the different integration methods you have implemented. Specifically, you should choose one subject, display the outputs for all of a-d (be sure to choose viewpoints that make the differences especially visible), and discuss which method produces the best results and why. Also, discuss how the Yale Face data violate the assumptions of the shape-from-shading method covered in the slides.

3 Extensions

On this assignment, there are not too many opportunities for ”easy” extra credit. This said, here are some ideas for exploration:

- Generate synthetic input data using a 3D model and a graphics renderer and run your method on this data. Do you get better results than on the face data? How close do you get to the ground truth (i.e., the true surface shape and albedo)?
- Investigate more advanced methods for shape from shading or surface reconstruction from normal fields.
- Try to detect and/or correct misalignment problems in the initial images and see if you can improve the solution.
- Using your initial solution, try to detect areas of the original images that do not meet the assumptions of the method (shadows, specularities, etc.). Then try to recompute the solution without that data and see if you can improve the quality of the solution.

If you complete any work for extra credit, be sure to clearly mark that work in your report, explain it and include the code.