Assignment 4: Photometric Stereo
Vision and Image Processing

Søren Olsen, Esben Plenge and François Lauze

January 9, 2016

This is the fourth mandatory assignment on the course Vision and Image Processing. The goal is to implement some basic Photometric Stereo.

This assignment must be solved in groups. We expect that you will form small groups of 2 to 4 students that will work on this assignment. You have to pass this and the other 3 mandatory assignments in order to pass the course.

The deadline for this assignment is Monday 23/1, 2016 at 20:00. You must submit your solution electronically via the Absalon home page. For general information on relevant software, requirement to the form of your solution including the maximal page limit, how to upload on Absalon etc, please see the first assignment.

Photometric Stereo

The goal of this assignment is to implement basic photometric stereo. For that you will use two datasets provided to you on Absalon in the `Assignment 4 Code and data` folder, `Beethoven.mat` and `Buddha.mat`. The first one consists of 3 images, and is a synthetic example, while the second one consists of 10 images and is a real one.

Note on datasets and softwares. Datasets are stored in Matlab mat-files. Each file contains the following variables:

- a 3D array `I` of size (m, n, k) where (m, n) is the size of each image and k is the number of views, i.e., view i corresponding to lighting $\mathbf{s}_i$ is `I(:,:,i)`.
- a 2D binary array `mask` of size (m, n). Pixels with values 1 (`true`) indicate positions where intensity data has been recorded. Photometric Stereo should only be solved at these points.
- an array `S` of light vectors, of size $(k, 3)$, where line i represents the directional light $\mathbf{s}_i$ that was used to obtain image `I(:,:,i)`.

Software for integration of the normal field and surface display is provided both for Matlab and Python. For Matlab, 3 functions are provided:

- `function z = unbiased_integrate(n1, n2, n3, mask)` computes a depth map for the normal field given by $(n1, n2, n3)^T$ only within the mask using a so-called "Direct Poisson Solver". The resulting array `z` has the same size as `mask`. Values that correspond to pixel locations where `mask == 0` are set to `nan` (Not a Number).
- `function z = simchony_integrate(n1, n2, n3, mask)` computes a depth map for the normal field given by $(n1, n2, n3)^T$ a Fourier-Transform based solver. As for `unbiased_integrate`, the resulting array `z` has the same size as `mask`. Values that correspond to pixel locations where `mask == 0` are set to `nan` (Not a Number).
- `function display_depth(z)` displays the obtained depth map `z` as a 3D graph.

For Python, a module called `ps_utils.py` is provided. It contains functions similar to the Matlab ones and an extra one.

- `unbiased_integrate(n1, n2, n3, mask)`, works as the Matlab one,
- `simchony_integrate(n1, n2, n3, mask)`, works as the Matlab one.
- `display_depth(z)`, works more or less as Matlab one. BEWARE: it requires Python's module/package `mayavi`.
- `read_data_file(filename)` reads a dataset Matlab mat-file and returns `I`, `mask` and `S`.
- It also contains a few extra functions that should not necessarily be of interest to you.

For both `Beethoven` and `Buddha` datasets, data manipulation and reshaping is very important to maintain good performances. Both Matlab and Python/Numpy allow you to extract a subarray as a list of value, and affects a list of value to an image subarray. You may want to look at integration source code to see how it can be done.

1 Beethoven Dataset

`Beethoven` is a synthetic and clean dataset, with exactly 3 images. If nz is the number of pixels inside the non-zero part of the mask, You should create an array J of size/shape $(3, nz)$ and obtain the albedo modulated normal field as $M = S^{-1}J$. With it, extract the albedo within the mask, display it as a 2D image. Then extract the normal field by normalizing M, extract its components $n1$, $n2$, $n3$. Solve for depth and display it at different view points.

2 Buddha Dataset

`Buddha` is real dataset, with exactly 10 images. If nz is the number of pixels inside the non-zero part of the mask, You should create an array J of size/shape $(10, nz)$ and obtain the albedo modulated normal field as $M = S^{\dagger}J$ (the pseudo-inverse). With it, extract the albedo within the mask, display it as a 2D image.

Then extract the normal field by normalizing M, extract its components $n1$, $n2$, $n3$. Solve for depth and display it art different view points.

The result might be disappointing! Suggest other possibilities for handling this dataset (as mentioned in the lectures). Feel free to implement them:-)