Vision and Image Processing
Assignment 3: Photometric Stereo

François Lauze and Søren Olsen

December 7, 2020

This is the third 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 mandatory assignments in order to pass the course.

The deadline for this assignment is Friday 18/12, 2020 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 and a bit less basic. Don't be surprised when your results are disappointing, Photometric stereo is not always robust. This is why RANSAC and normal field smoothing are introduced in this assignment.

For that you will use several datasets provided to you on Absalon in the `Assignment 3 Code and data` folder, they are

1. `Beethoven.mat`, 3 images, (semi-)synthetic (comes from a real model, but processed).
2. `mat_vase.mat`, 3 images, synthetic.
3. `shiny_vase.mat`, 3 images, synthetic.
4. `shiny_vase2.mat`, 22 images, synthetic.
5. `Buddha.mat`, 10 images, real.
6. `face.mat`, 27 images, real.

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)`.

Python software for reading `.mat` files, integration of the normal field, surface display is provided and smoothing/regularization of normal fields in a module called `ps_utils.py`, written for Python 3.

- `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).
- `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).
- `display_surface(z, albedo=None)` renders a 3D surface which height is coded in z. it requires Python's module/package `mayavi` which can be installed via `Anaconda` and `pip`.
- `read_data_file(filename)` reads a dataset Matlab mat-file and returns `I`, `mask` and `S`.
- `ransac_3dvector(data, threshold, max_data_tries=100, max_iters=1000, p=0.9, det_threshold=1e-1, verbose=2)` is a specialised implementation of RANSAC which returns an hopefully robust solution to the estimation of a 3D vector m from measurements I_i which should be obtained as $I_i = s_i^T m$, with $s_1, \ldots, s_K$ known vectors (light sources to us).
- `smooth_normal_field(n1, n2, n3, mask, iters=100, tau=0.05, verbose=False)` smoothes a normal field via a numerical partial differential equation.
- It also contains a few extra functions that should not necessarily be of interest to you.

Each of these functions are reasonably well documented using docstrings, so, after importing `ps_utils.py`, you can invoke `help(ps_utils)` for the entire documentation, or for a specific function such as `help(ransac_3dvector)` etc...

For both all datasets 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. Woodham's estimation can be vectorised. You may want to look at integration source code to see how Python allows for these type of manipulations.

When using RANSAC, however, an estimation of albedo and normal will have to be performed one pixel at a time.

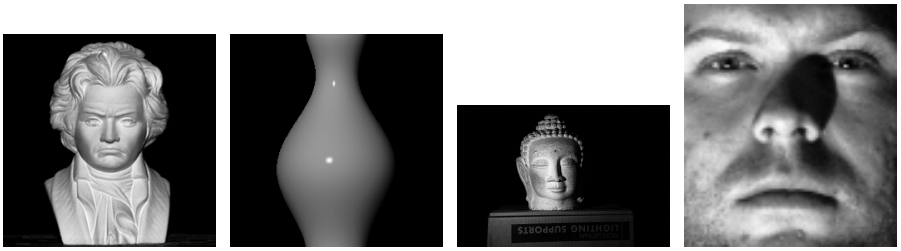

Figure 1: A few images out of the different datasets

1 A bit of modelling

You shoud provide brief answers to the following points. No need to elaborate too much.

- Write for a generic pixel, light source and normal, Lambert's law in its simplest form.
- How is Lambert's law modified to deal with self shadows.
- What about cast shadows? Comment on the difference between the two.
- Comment on the modelling limits of Lambert's law.
- How can we obtain an estimate of albedo and normals in Woodham's approach to Photometric Stereo. Write the equation.
- What should be done if one uses RANSAC. Please describe. It will help you when implementing the RANSAC based estimation.

2 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.

3 mat_vase Dataset

`mat_vase` 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.

4 shiny_vase Dataset

`shiny_vase` is a synthetic and clean dataset, however not respecting Lambert's law, which cannot cope with specularities. It also consists of 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. Comment on what happens here.

Do you think that RANSAC could provide a better estimation of normals? Explain. You should try and replace Woodham's first step (via inverse/pseudoinverse) with RANSAC estimation. The `threshold` parameter in `ransac_3dvector()` should no more than be 2.0. After the estimation for each pixel, extract the normals and the albedo. Display and comment on the results. Do they differ from Woodham's estimation? Try then to make the estimated normal field smoother using the `smooth_normal_field()` function. You may experiment with the `iters` parameter.

5 shiny_vase2 Dataset

`shiny_vase2` is a synthetic and clean dataset, however not respecting Lambert's law, which cannot cope with specularities. It consists of exactly 22 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 $(22, nz)$ and obtain the albedo modulated normal field as $M = S^{\dagger}J$. 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. Comment on what happens here.

Do you think that RANSAC could provide a better estimation of normals? Explain. You should try and replace Woodham's first step (via inverse/pseudoinverse) with RANSAC estimation. The `threshold` parameter in `ransac_3dvector()` should no more than be 2.0. After the estimation for each pixel, extract the normals and the albedo. Display and comment on the results. Do they differ from Woodham's estimation? Try then to make the estimated normal field smoother using the `smooth_normal_field()` function. You may experiment with the `iters` parameter.

6 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 at different view points.

You should try and replace Woodham's first step (via inverse/pseudoinverse) with an estimation using RANSAC. The `threshold` parameter in `ransac_3dvector()` should be **at least** 25.0 Experiment with it. After the estimation for each pixel, extract the normals and the albedo. Display and comment on the results. Try then to make the estimated normal field smoother using the `smooth_normal_field()` function. You may experiment with the `iters` parameter.

7 face Dataset

`Buddha` is real dataset, with exactly 27 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 $(27, nz)$ and obtain

the albedo modulated normal field via RANSAC with a threshold of 10.0 Try then to make the estimated normal field smoother using the `smooth_normal_field()` function. You may experiment with the `iters` parameter. Report your results.