Assignment 1: Photometric Stereo & Colour

Computer Vision 1
University of Amsterdam

Due 11:59pm, September 19, 2021 (Amsterdam time)

General guidelines

Your code and report must be handed in together in a zip file (`ID1_ID2_ID3.zip`) before the deadline by submitting it to the Canvas Lab 1 Assignment. For full credit, make sure your report follows these guidelines:

- Include an introduction and a conclusion to your report.
- The maximum number of pages is 10 (single-column, including tables and figures). Please express your thoughts concisely. The number of words does not necessarily correlate with how well you understand the concepts.
- Answer all given questions (in green boxes). Briefly describe what you implemented. Blue boxes are there to give you hints to answer questions.
- Try to understand the problem as much as you can. When answering a question, give evidences (qualitative and/or quantitative results, references to papers, figures etc.) to support your arguments. Note that not everything might be explicitly asked for and you are expected to think about what might strengthen you arguments and make the report self-contained and complete.
- Analyze your results and discuss them, e.g. why algorithm A works better than algorithm B in a certain problem.
- Tables and figures must be accompanied by a brief description. Do not forget to add a number, a title, and if applicable name and unit of variables in a table, name and unit of axes and legends in a figure.

Late submissions are not allowed. Assignments that are submitted after the strict deadline will not be graded. In case of submission conflicts, TAs' system clock is taken as reference. We strongly recommend submitting well in advance, to avoid last minute system failure issues.
Plagiarism note: Keep in mind that plagiarism (submitted materials which are not your work) is a serious crime and any misconduct shall be punished with the university regulations.

Contents

1 Photometric Stereo (50pts)

In this part of the assignment, you are going to implement the photometric stereo algorithm as described in Section 5.4 (Forsyth and Ponce, *Computer Vision: A Modern Approach*). The chapter snippet can be found in the course materials.

Following this instruction, you will have to edit and fill in your code in the files **estimate_alb_nrm.py**, **check_integrability.py**, and **construct_surface.py**. The main script **photometric_stereo.py** is provided for reference and should not be taken as is. Throughout the assignment, you will be asked to perform different trials and experiments which will require you to adjust the main code accordingly, this also shows how well you can cope with the materials.

Include images of the results into your report. For 3D models, make sure to choose a viewpoint that makes the structure as clear as possible and/or feel free to take them from multiple viewpoints.

1.1 Estimating Albedo and Surface Normal

Let us start with the grayscale sphere model, which is located in the `SphereGray5` folder. The folder contains 5 images of a sphere with grayscale checker texture under similar lighting conditions with the one in the book. Your task is to estimate the surface reflectance (albedo) and surface normal of this model. The light source directions are encoded in the image file names.

Hint

To get the least-squares solution of a linear system, you can use `numpy.linalg.lstsq` function.

Question - 1 (15-*pts*)

1. Complete the code in **estimate_alb_norm.py** to estimate albedo and surface normal map for the `SphereGray5` folder. What do you expect to see in albedo image and how is it different with your result?

2. In principle, what is the minimum number of images you need to estimate albedo and surface normal? Run the algorithm with more images by using `SphereGray25` and observe the differences in the results. You could try all images at once or a few at the time, in an incremental fashion. Choose a strategy and justify it by discussing your results.

3. What is the impact of shadows in photometric stereo? Explain the trick that is used in the text to deal with shadows. Remove that trick and check your results. Is the trick necessary in the case of 5 images, how about 25 images?

1.2 Test of Integrability

Before we can reconstruct the surface height map, it is required to compute the partial derivatives $\frac{\partial f}{\partial x}$ and $\frac{\partial f}{\partial y}$ (or `p` and `q` in the algorithm). The partial derivatives also give us a chance to double check our computation, namely the test of *integrability*.

Question - 2 (10-*pts*)

1. Compute the partial derivatives (`p` and `q` in the algorithm) by filling in your code into **check_integrability.py**.
2. Implement and compute the second derivatives according to the algorithm and perform the test of integrability by choosing a reasonable threshold. What could be the reasons for the errors? How does the test perform with different number of images used in the reconstruction process in Question-1?

1.3 Shape by Integration

To reconstruct the surface height map, we need to continuously integrate the partial derivatives over a path. However, as we are working with discrete structures, you will be simply summing their values.

The algorithm in the chapter presents a way to do the integration in *column-major* order, that is you start at the top-left corner and integrate along the first column, then go towards right along each row. Yet, it is also noticed that it would be better to use many different paths and average so as to spread around the errors in the derivative estimates.
Note: By default, Numpy used row-major operations. So if you are unrolling an image to linearize the operation, you will end up with a row-major representation. Numpy can be configured to be column-major. Otherwise, if you are using the double for-loops without an unrolling operation, then this concern doesn't apply.

Question - 3 (10-*pts*)

1. Construct the surface height map using column-major order as described in the algorithm, then implement row-major path integration. Your code should now go to **construct_surface.py**. What are the differences in the results of the two paths?
2. Now, take the average of the results. Do you see any improvement compared to when using only one path? Are the construction results different with different number of images being used?

Hint

You could further inspect the shape of the objects and normal directions by using `matplotlib.pyplot.quiver` function. You will have to choose appropriate sub-sampling ratios for proper illustration. You code goes to the `show_results` function in **utils.py**.

1.4 Experiments with different objects

In this part, you will try to run the photometric stereo algorithm in various number of scenarios to see how well it can be generalized.

Question - 4 (5-*pts*)

Run the algorithm and show the results for the `MonkeyGray` model. The albedo results of the monkey may comprise more albedo errors than in case of the sphere. Observe and describe the errors. What could be the reason for those errors? You may want to experiment with different number of images as you did in Question-1 to see the effects. How do you think that could help solving these errors?

So far, we have assumed that albedos are 1-channel grayscale images and that input images are also 1-channel. To work with 3-channel images, a simple solution is to split the input image into separate channels and treat them individually. Yet, that would generate a small problem while constructing the surface normal map if a pixel value in a channel is zero.

Question - 5 (5-*pts*)

Update the implementation to work for 3-channel RGB inputs and test it with 2 models `SphereColor` and `MonkeyColor`. Explain your changes and show your results. Observe the problem in the constructed surface normal map and height map, explain why a zero pixel could be a problem and propose a way to overcome that.

Now, it is the time to try the algorithm on real-world datasets. For that purpose, we are going to use the *Yale Face Database*[1].

Hint

For proper computation of albedo and surface normal, you may want to suspend the shadow trick described in the text, and use the original formula

$$\boldsymbol{i} = \mathcal{V}\boldsymbol{g}(x, y)$$

[1] `http://cvc.cs.yale.edu/cvc/projects/yalefaces/yalefaces.html`

Question - 6 (5-*pts*)

Run the algorithm for the Yale Face images (included in the lab material). Observe and discuss the results for different integration paths. Discuss how the images violate the assumptions of the shape-from-shading methods. Remember to include specific input images to illustrate your points. How the results would improve when the problematic images are all removed? Show the results in your report.

2 Colour Spaces (20pts)

In this part of the assignment, you will study the different colour spaces for image representations and experiment how to convert a given RGB image to a specific colour space.

RGB Colour Model (3-*pts*)

Why do we use RGB colour model as a basis of our digital cameras and photography? How does a standard digital camera capture the full RGB colour image?

Colour Space Conversion (10-*pts*)

Create a function to convert an RGB image into the following colour spaces by using the template code you are provided **ConvertColourSpace.py** and other sub-functions. Visualize the new image and its channels separately in the same figure. That is, for example, in the case of HSV colour space, you need to visualize the converted HSV image, and its Hue, Saturation and Value channels separately (4 images, 1 figure). Do not change the already given code.

Opponent Colour Space

$$\begin{pmatrix} O_1 \\ O_2 \\ O_3 \end{pmatrix} = \begin{pmatrix} \frac{R-G}{\sqrt{2}} \\ \frac{R+G-2B}{\sqrt{6}} \\ \frac{R+G+B}{\sqrt{3}} \end{pmatrix}$$

Normalized RGB (rgb) Colour Space

$$\begin{pmatrix} r \\ g \\ b \end{pmatrix} = \begin{pmatrix} \frac{R}{R+G+B} \\ \frac{G}{R+G+B} \\ \frac{B}{R+G+B} \end{pmatrix}$$

HSV Colour Space

Convert the RGB image into HSV Colour Space. Use OpenCV's built-in function *cv2.cvtColor(img, CV2.RGB2HSV)*.

YCbCr Colour Space

Convert the RGB image into YCbCr Colour Space. Use OpenCV's built-in function *cv2.cvtColor(img, CV2.RGB2YCrCb)*. Note, you need to arrange the channels in

Y, C_b and C_r order.

Grayscale

Convert the RGB image into grayscale by using 3 different methods mentioned in `https://www.johndcook.com/blog/2009/08/24/algorithms-convert-color-grayscale/` In the end, check and report which method OpenCV uses for grayscale conversion, include it as well, and visualize all 4 in the same figure.

Colour Space Properties (5-*pts*)

Explain each of those 5 colour spaces and their properties. What are the benefits of using a different colour space other than RGB? Provide reasons for each of the above cases. You can include your observations from the visualizations.

More on Colour Spaces (2-*pts*)

Find one more colour space from the literature and simply explain its properties and give a use case.

3 Intrinsic Image Decomposition (15pts)

Intrinsic image decomposition is the process of separating an image into its formation components, such as reflectance (albedo) and shading (illumination).[2] Then, under the assumption of body (diffuse) reflection, linear sensor response and narrow band filters, the decomposition of the observed image $I(\vec{x})$ at position $\vec{x}$ can be approximated as the element-wise product of its albedo $R(\vec{x})$ and shading $S(\vec{x})$ intrinsics:

$$I(\vec{x}) = R(\vec{x}) \times S(\vec{x}). \tag{1}$$

In this part of the assignment, you will experiment with intrinsic image components to perform one of the computational photography applications; material recolouring. For the experiments, we will use images from a synthetic intrinsic image dataset.[3]

Other Intrinsic Components (4-*pts*)

What other components can an image be decomposed other than albedo and shading? Give an example and explain your reasoning.

Synthetic Images (2-*pts*)

If you check the literature, you will see that almost all the intrinsic image decomposition datasets are composed of synthetic images. What might be the reason for that?

Image Formation (4-*pts*)

Show that you can actually reconstruct the original *ball.png* image from its intrinsics using *ball_albedo.png* and *ball_shading.png*. In the end, your script should output a figure displaying the original image, its intrinsic images and the reconstructed one. Name your script as **iid_image_formation.py**.

Recoloring

Manipulating colours in photographs is an important problem with many applications in computer vision. Since the aim for recolouring algorithms is just to manipulate colours, better results can be obtained for such a task if the albedo image is available as it is independent of confounding illumination effects.

[2] H. G. Barrow and J. M. Tenenbaum. Recovering intrinsic scene characteristics from images. Computer Vision Systems, pages 3-26, 1978.

[3] `http://www.cic.uab.cat/Datasets/synthetic_intrinsic_image_dataset/`

Recoloring (5-*pts*)

Assume that you are given the *ball.png* image and you have access to its intrinsic images *ball_albedo.png* and *ball_shading.png*.

1. Find out the true material colour of the ball in RGB space (which is uniform in this case).
2. Recolour the ball image with pure green (0,255,0). Display the original ball image and the recoloured version on the same figure. Name your script as **recoloring.py**.
3. Although you have recoloured the object with pure green, the reconstructed images do not seem to display those pure colors and thus the colour distributions over the object do not appear uniform. Explain the reason.

Note that this was a simple case where the image is synthetic, object centered and has only one colour, and you have access to its ground-truth intrinsic images. Real world scenarios require more than just replacing a single colour with another, not to mention the complexity of achieving a decent intrinsic image decomposition.

4 Colour Constancy (15pts)

Colour constancy is the ability to perceive colors of objects, invariant to the colour of the light source. The aim for colour constancy algorithms is first to estimate the illuminant of the light source, and then correct the image so that the corrected image appears to be taken under a canonical (white) light source. The task of the automatic white balance (AWB) is to do the same in digital cameras so that the images taken by a digital camera look as natural as possible.

In this part of the assignment, you will implement the most famous colour constancy algorithm; *Grey-World Algorithm*.

Grey-World Algorithm

The algorithm assumes that, under a white light source, the average colour in a scene should be achromatic (grey, [128, 128, 128]).

Grey-World (15-*pts*)

1. Create a function to apply colour correction to an RGB image by using Grey-World algorithm. Display the original image and the colour corrected one on the same figure. Name your script as **AWB.py**. Use *awb.jpg* image to test your algorithm. In the end, you should see that the reddish colour cast on the image is removed and it looks more natural.

   Note: You do not need to apply any pre or post processing steps. For the calculation or processing, you are not allowed to use any available code or any dedicated library function except *standard Numpy functions*.

2. Give an example case for Grey-World Algorithm on where it might fail. Remember to include your reasoning.

3. Find out one more colour constancy algorithms from the literature and explain it briefly.

Hint

Check the von Kries model for the colour correction step.