Computational Photography - Programming exercise 2

Due: March 21, 2024

The purpose of this assignment is to explore lightfields and focal stacks. As we discussed in class, having access to the full lightfield of a scene allows creating images that correspond to different viewpoint, aperture, and focus settings. We also discussed how having a focal stack of a scene allows creating an all-in-focus image and a depth map using depth from defocus.

Here, you will combine the two into a single pipeline: Instead of capturing a focal stack, you will synthesize one from a lightfield image captured with a plenoptic camera. Then, from the synthetic focal stack, you will compute an all-in-focus image and a depth map.

You are strongly encouraged to read the handheld plenoptic camera paper by Ng et al. [2] that we discussed in class. Sections 3.3 and 4, in particular, will be very helpful for understanding how to solve this homework assignment.

1. Lightfield rendering and focal stacks

For the first part of the homework, you will use a lightfield image of a chess board scene, obtained from the Stanford Light Field Archive [1]. (We also used this scene for related examples during the lightfield lecture.) The lightfield is available as file `./data/chessboard_lightfield.png` in the homework ZIP archive. This image file is formatted in the same way as images captured by a plenoptic camera, with the pixels under neighboring lenslets corresponding to neighboring patches in the image. This format is described in detail in [2]. Figure 1 shows a crop from the center of the lightfield image, as well as a regular pinhole-camera view of the chessboard scene.

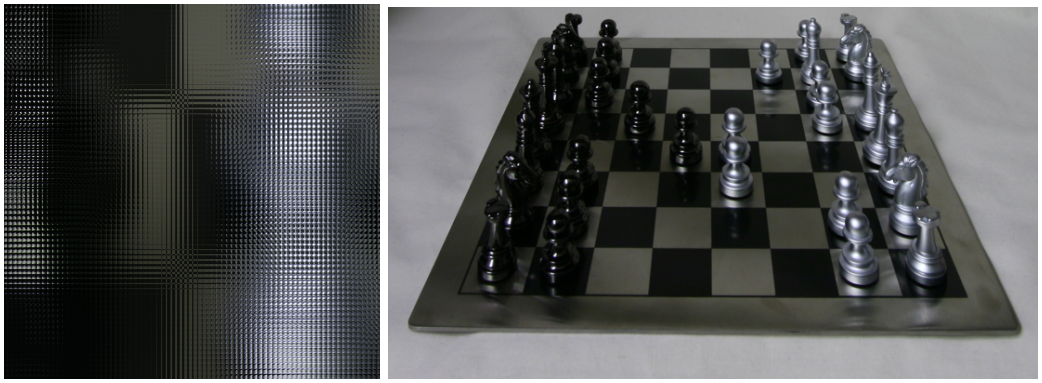

Figure 1: The chessboard scene lightfield. Left: Crop of the lightfield image. Right: A pinhole camera view of the scene.

Initials: Load the lightfield image, and create from it a 5-dimensional array $L(u, v, s, t, c)$. The first four dimensions correspond to the 4-dimensional lightfield representation we discussed in class, with u and v being the coordinates on the aperture, and s and t the coordinates on the lenslet array. The fifth dimension $c = 3$ corresponds to the 3 color channels. When creating this structure, you can use the fact each lenslet covers a block of 16×16 pixels.

Submit a function implementing this with the format: **LF=arrangeLF(I,u_max,v_max)**, where I is the plenoptic image, u_max, v_max the number of u, v samples under each lenslet (16 for the data we provide), and LF is the resulting 4D lightfield.

Sub-aperture views: As we discussed in class, by rearranging the pixels in the lightfield image, we can create images that correspond to views of the scene through a pinhole placed at different points on the camera aperture (a "sub-aperture"). This is equivalent to taking a slice of the lightfield of the form

$L(u = u_o, v = v_o, s, t, c)$, for some values of u_o and v_o corresponding to the point on the aperture where we place the pinhole. For the chessboard lightfield, we can generate 16×16 such images.

Create all of these sub-aperture views, and arrange them into a 2D mosaic, where the vertical dimension will correspond to increasing u values, and the horizontal dimension to increasing v values. Figure 2 shows the expected result. Submit the mosaic with your solution. Use a variable named $ImArray1$ to store the array. Save the image array as a .png image called $ex1\_q1.png$, and add it to the report.

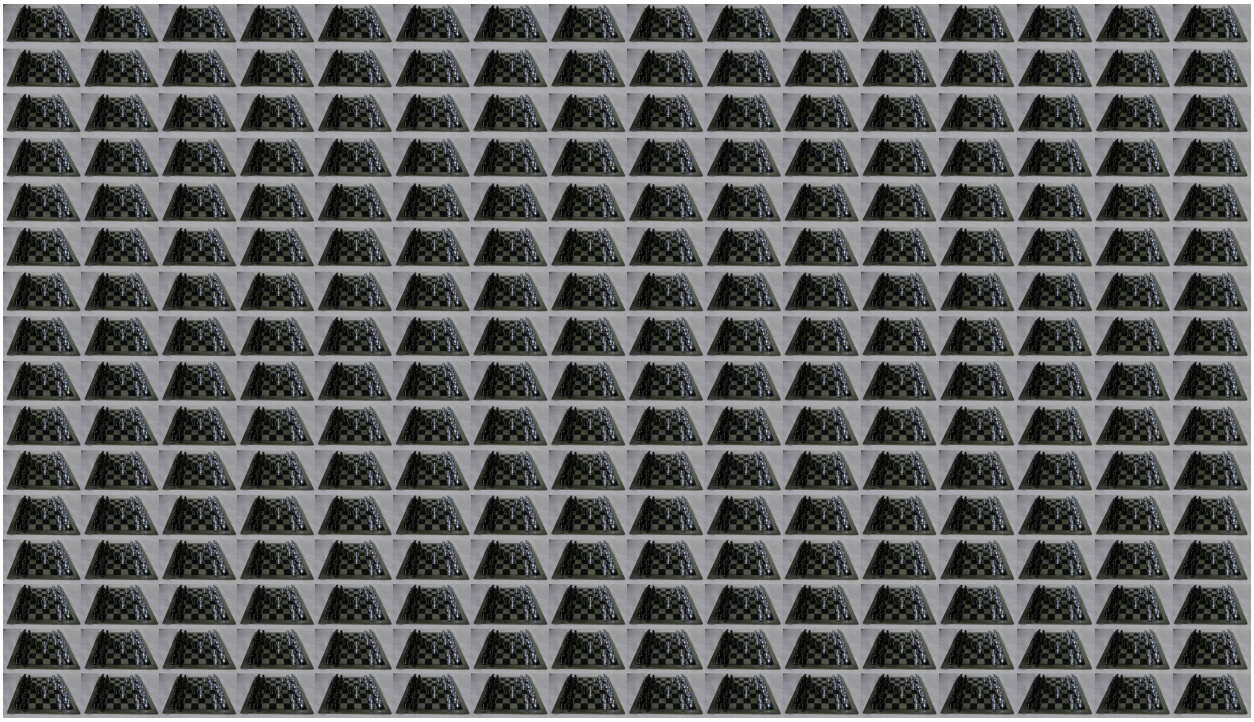

Figure 2: Mosaic of sub-aperture views.

Refocusing and focal-stack generation: A different effect that can be achieved by appropriately combining parts of the lightfield is refocusing at different depths. In particular, averaging all of the sub-aperture views results in an image that is focused at a depth near the top of the chess board. This corresponds to creating an image as:

$$\int_u \int_v L(u, v, s, t, c)\, \mathrm{d}v\, \mathrm{d}u. \tag{1}$$

The resulting image is shown to the left of Figure 3. As we discussed in class and explained in detail in Section 4 of [2], focusing at different depths requires shifting the sub-aperture images before averaging them, with the shift of each image depending on the desired focus depth and the location of its sub-aperture. More concretely, to focus at depth d represented by lightfield slope s, we need to combine the sub-aperture images as:

$$I(s, t, c, s) = \int_u \int_v L(u, v, s + su, t + sv, c)\, \mathrm{d}v\, \mathrm{d}u. \tag{2}$$

For $s = 0$, the image we obtain is the same as in Equation (1). Note that s here is a slope in the light field. There is a linear mapping between it and the actual depth at which you focus, but you are not requested to find it, you can assume you are getting an input a slope. (Eq.6 in [2] is doing also a zoom, but you are requested to implement Equation (2), which is simpler). Figure 3 shows refocused images for two more settings of s.

To handle a general aperture mask m you should sum

$$I(s,t,c,s) = \int_u \int_v m(u,v) \cdot L(u,v,s+su,t+sv,c)\, \mathrm{d}v\, \mathrm{d}u. \tag{3}$$

You should implement Equation (3), and submit a function with the format: **I=refocus(LF,s,mask)**, where LF is a 5D light field you generated above, a s is the target refocusing slope, and mask is an aperture mask. Note that s may be a fractional number and you need to implement bilinear interpolation.

Use it to generate a focal stack $I(s,t,c,s)$ for a range of values s. Make sure that your focal stack is long enough so that each part of the scene is in focus in at least one image in the stack. In your solution, show 3 different refocusings (covering the full depth range of the scene), using each of the 4 aperture masks provided in the file masks.mat. Discuss the influence of the different aperture masks in your report.

Use a variable named $ImArray2\_j$ (for $j = 1 \ldots 4$) to store a 1×3 array of 3 refocused images with each aperture mask.

Save these image array as a .png image called $ex2\_q2\_mj.png$, and add it to the report.

Figure 3: Refocusing at different depths. The left image corresponds to using Equation (2) with $s = 0$ (or equivalently, using Equation (1)).

2 All-focus image and depth from defocus

As we saw in class, when we have access to a focal stack, we can merge the images into a new images where all of the scene is in focus. In the process of doing so, we also obtain depth estimates for each part of the scene, a procedure that is known as *depth from defocus*.

To merge the focal stack into a single all-focus image, we first need to determine per-pixel and per-image weights. The weights in this case correspond to how "sharp" the neighborhood of each pixel is at each image in the focal stack.

There are many possible sharpness weights. Here you will implement the following:

1. For every image in the focal stack, first convert it to the XYZ colorspace, and extract the luminance channel:
$$I_{\text{luminance}}(s,t,d) = rgb2gray\left(I(s,t,c,d)\right). \tag{4}$$
2. Create a low-frequency component by blurring it with a Gaussian kernel of standard deviation σ_1:
$$I_{\text{low-freq}}(s,t,d) = G_{\sigma_1}(s,t) * I_{\text{luminance}}(s,t,d). \tag{5}$$
Note: the filter G_{σ_1} should be normalized such that it sums to 1.
3. Compute a high-frequency component by subtracting the blurry image from the original:
$$I_{\text{high-freq}}(s,t,d) = I_{\text{luminance}}(s,t,d) - I_{\text{low-freq}}(s,t,d). \tag{6}$$
4. Compute the sharpness weight by blurring the *square* of high-frequency component with another Gaussian kernel of standard deviation σ_2:
$$w_{\text{sharpness}}(s,t,d) = G_{\sigma_2}(s,t) * \left(I_{\text{high-freq}}(s,t,d)\right)^2. \tag{7}$$

Note that the weights are the same for each of the color channels.

Once you have the sharpness weights, you can compute the all-focus image as:

$$I_{\text{all-focus}}(s,t,c) = \frac{\sum_d w_{\text{sharpness}}(s,t,d)\, I(s,t,c,d)}{\sum_d w_{\text{sharpness}}(s,t,d)}. \tag{8}$$

In addition, you can create a per-pixel depth map by using the weights to merge depth values instead of pixel intensities, that is:

$$\text{Depth}(s,t) = \frac{\sum_d w_{\text{sharpness}}(s,t,d)\, d}{\sum_d w_{\text{sharpness}}(s,t,d)}. \tag{9}$$

Figure 4 shows the all-focus image and depth map resulting from one set of σ_1 and σ_2 values used for sharpness evaluation. You should experiment with different values and report which ones work best, as well as show the corresponding all-focus image and depth map.

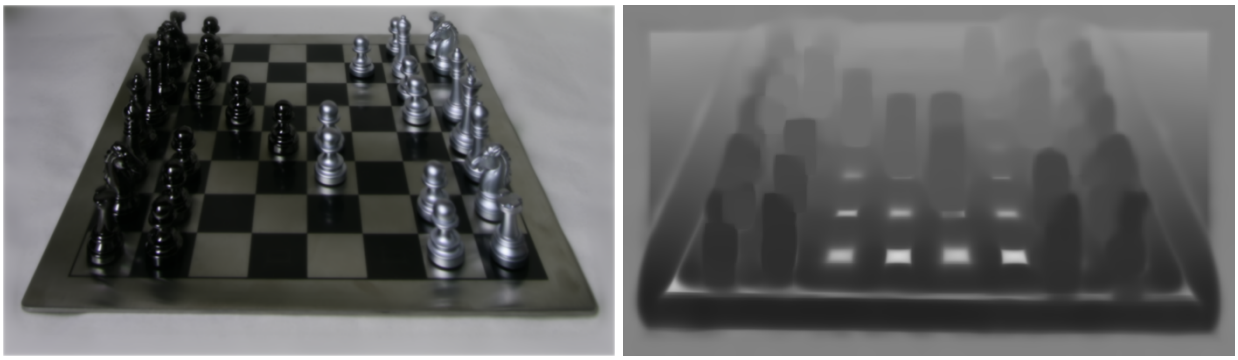

Figure 4: Left: All focus image for one set of σ_1 and σ_2 values. Right: Corresponding depth map.

Implementing the above algorithm, you should submit a function with the format:
[all_focus_I,depth_I]=refocusAll(focal_stack), where **focal_stack** is a 4D array of the format $focal\_stack(s,t,c,d)$.

Note: 1) To create the focal stack for this part, run the function from part 1, with the widest aperture mask (that is a 16×16 mask of all ones). 2)In this assignment you can treat the light field slope as depth d. (In practice to convert slope to depth you need some additional information such as the focal length of the main lens).

Deliverables

Your solution should be a ZIP archive that includes the following:

- 3 Phython functions (.py files), the first one loading the camera data and arranging it into a light field, the second one refocusing, and the 3rd one extracting an all-in-focus image plus depth map, as described above.
- The viewpoint collage you create, as well as 4 sets of refocused images.
- An all-in-focus image and the equivalent depth map, generated with a good selection of the σ_1, σ_2 parameters.
- One .py script named MainEx2.py which will go through all the subsections.
- A written PDF named studentID1_studentID2.pdf (write your ID numbers instead of studentID1, studentID2) report with the analysis requested above including the image array for each subsection.
- Wrap all the py files, PDF and images into a single zip file name: studentID1_studentID2.zip (again use your IDs here).

Hints and Information

- to convert an RGB image into a grayscale one use

```
def rgb2gray(rgb):
 return np.dot(rgb[...,:3],[0.2989, 0.5870,0.1140])
```

- To implement the shifts described in various parts of the homework, you can use

  `scipy.interpolate.interp2d,`

  with the linear interpolation option.

- To implement the above functions efficiently you should use matrix operations. Implementations using 4 nested loops will not be accepted.

- When refocusing in problem 1, it will be helpful to express the coordinates u and v relative to the center of each lenslet. You can do this as follows:

```
lensletSize = 16;
maxUV = (lensletSize - 1) / 2;
u = (1:lensletSize) - 1 - maxUV;
v = (1:lensletSize) - 1 - maxUV;
```

References

[1] The (new) stanford light field archive, 2008. http://lightfield.stanford.edu/.

[2] R. Ng, M. Levoy, M. Brédif, G. Duval, M. Horowitz, and P. Hanrahan. Light field photography with a hand-held plenoptic camera. *Stanford Computer Science Technical Report*, 2005.