Programming Assignment 2: Light Fields

The purpose of this assignment is to explore light fields, focal stacks, and depth from focus.

Problem 1:

Task1: Initialization

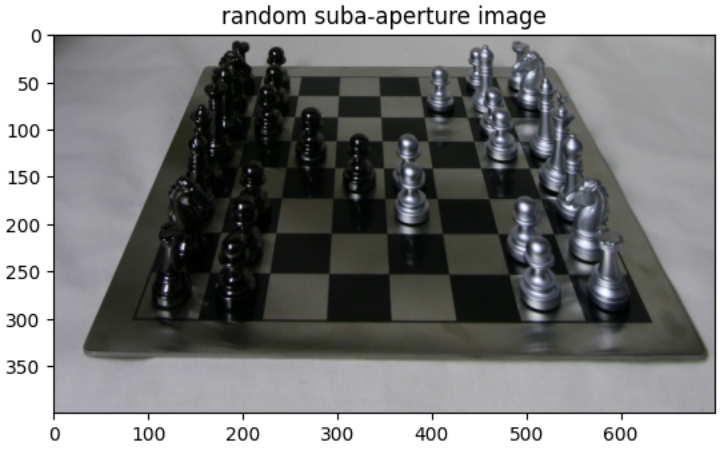
Load the light field image in Python, and create from it a 5-dimensional array L (u, v, s, t, c). The first four dimensions correspond to the 4-dimensional light field representation we discussed in class, with u and v being the coordinates on the aperture, and s and t the coordinates on the lenslet 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.

Solution:

```
#Task1: Initials

lensletSize = 16 #lenslet covers a block of 16 x 16 pixels
aperture  dim = [lensletSize, lensletSize]
lenslet_array_dim = [img.shape[0]//aperture_dim[0], img.shape[1]//aperture_dim[1]]

# creating the 5-dimensional array L(u; v; s; t; c).
img2 = img.reshape([lenslet_array_dim[0], aperture_dim[0], lenslet_array_dim[1], aperture_dim[1], 3])
print(img2.shape)
L = img2 = img2.transpose((1, 3, 0, 2, 4))
print(L.shape)

# display a random suba-aperture image to check the generation of  L
show_image(img=L[2,1], title='random suba-aperture image')
```

Result:(400, 16, 700, 16, 3)

(16, 16, 400, 700, 3)



Task2: Sub-aperture views

Discription:

According to the discussion of the 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 = u0; v = v0; s; t; c), for some values of u0 and v0 corresponding to the point on the aperture where we place the pinhole. For the chessboard lightfield, we can generate 16 x 16 such images.

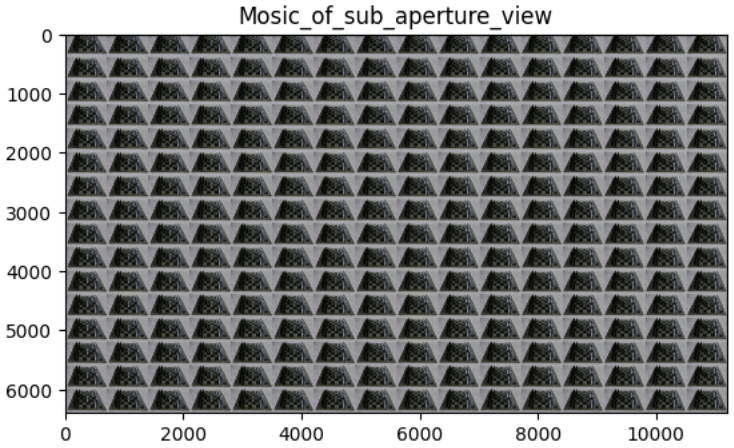
Task:

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. Submit the mosaic with your solution.

Solution :

Code

```
#  Task2: Sub-aoerture views
#rearranging the pixels in the lightfield image
#Creating all of the sub-aperture views and the arrangement of the 2D mosaic
#the vertical dimention is corresposing to increasing u values
sub_aperture_view = img3 = L.transpose((0, 2, 1, 3, 4)).reshape(img.shape)
show_image(sub_aperture_view, 'Mosic_of_sub_aperture_view')
```



Task3: Refocusing and focal-stack simulation

Description: 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:

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

focusing on 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 on depth d, we need to combine the sub-aperture images as:

$$I\left(s, t, c, d\right) = \int_u \int_v L\left(u, v, s + d \cdot u, t + d \cdot v, c\right)\, \mathrm{d}v\, \mathrm{d}u. \qquad (2)$$

Task:

Implement Equation (2), and use it to generate a focal stack I (s, t, c, d) for a range of values d. 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, make sure to show at least five different re focusings.

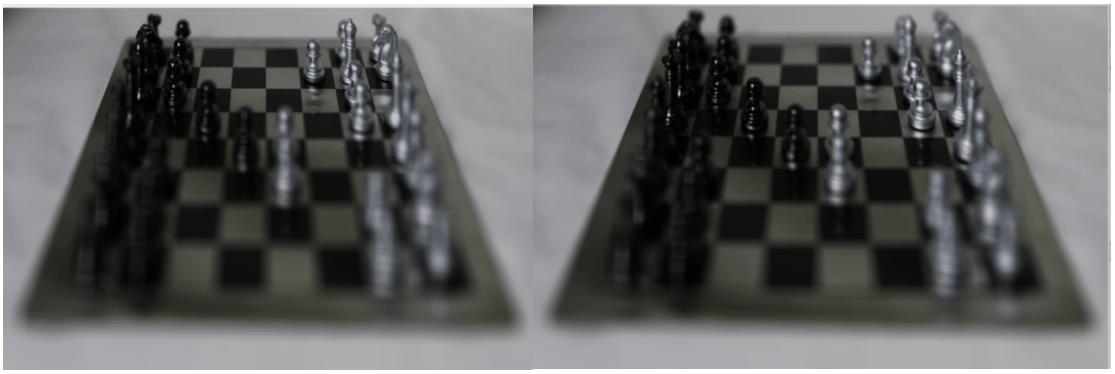
Solution:

Code:

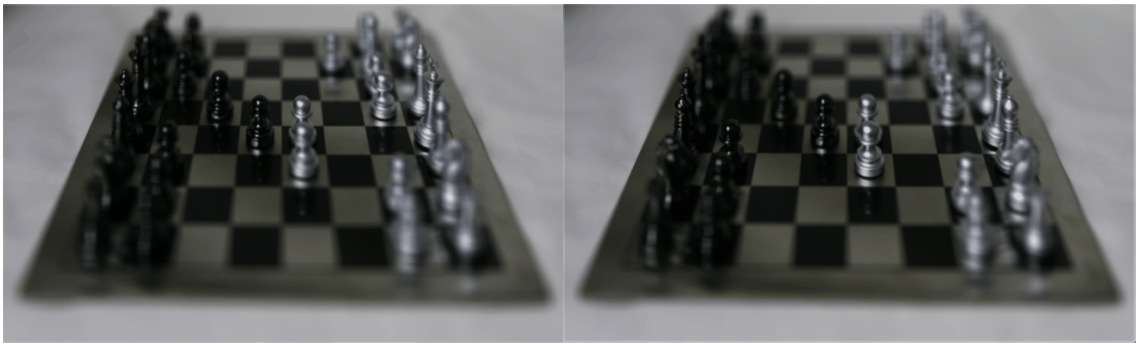
**** The code is in the attached Python file**

Output: The gif images

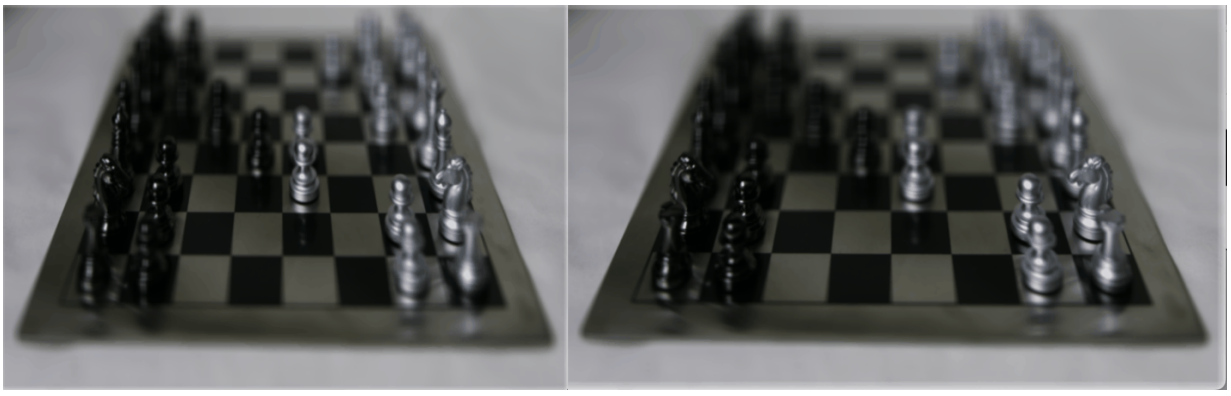
Focal stakes:



Focus :1 Focus:2



Focus:3 Focus:4



Focus:5 Focus:6

Focus:7

Focus:8

Focus:9

Focus:10

Task4:

Description: we can merge the images into a new image 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 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, In particular, 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.

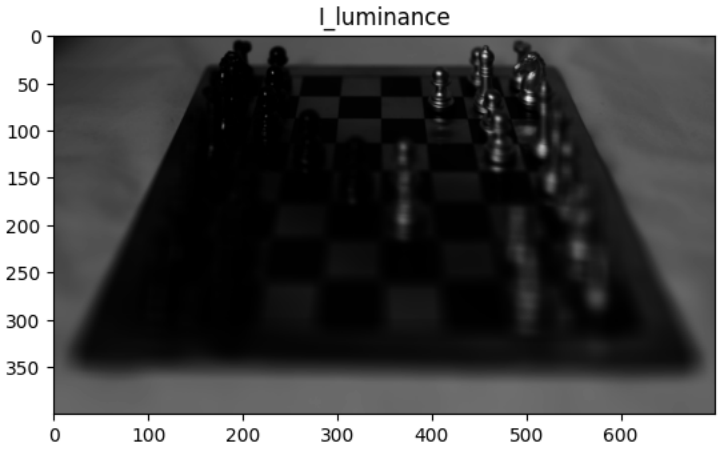
Task:

Here is the implementation the following:

1. For every image in the focal stack, first convert it to the XYZ colorspaceand extract the luminance channel (hint: import skimage.color):

$$I_{\text{luminance}}(s,t,d) = \text{get\_luminance}\left(\text{rgb2xyz}\left(I(s,t,c,d)\right)\right). \tag{3}$$

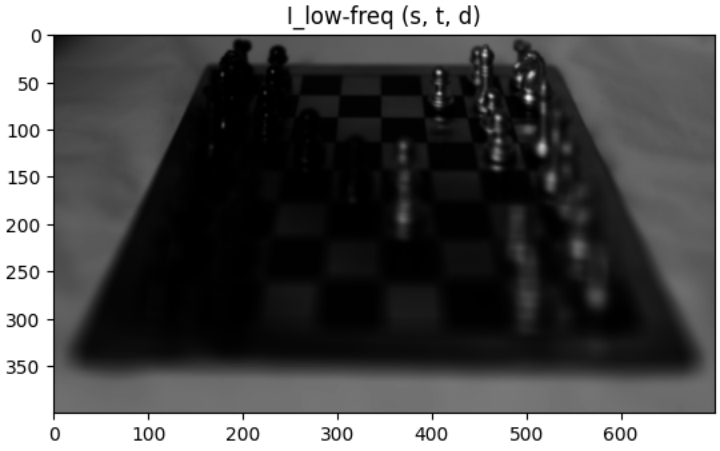
Result: I_luminance range: 0.0002575428991028399 0.9992252663626364



2. Create a low-frequency component by blurring it with a Gaussian kernel of standard deviation σ1:

$$I_{\text{low-freq}}(s,t,d) = G_{\sigma_1}(s,t) * I_{\text{luminance}}(s,t,d)\,. \tag{4}$$

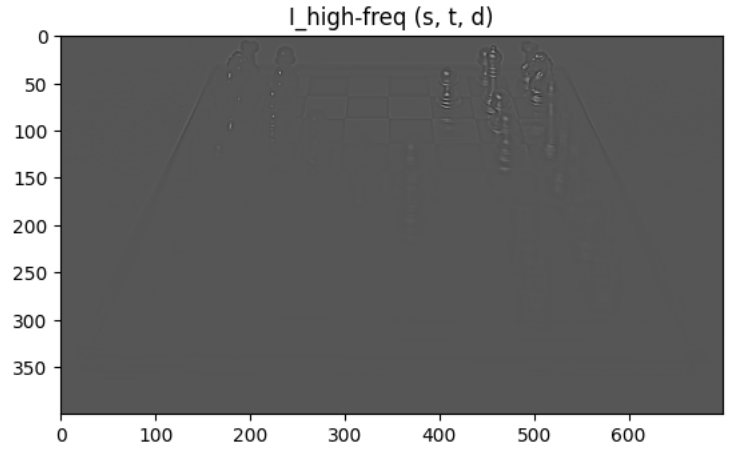
Result: sigma1 =0.9



3. Compute a high-frequency component by subtracting the blurry image from the original:

$$I_{\text{high-freq}}(s,t,d) = I_{\text{luminance}}(s,t,d) - I_{\text{low-freq}}(s,t,d)\,. \tag{5}$$

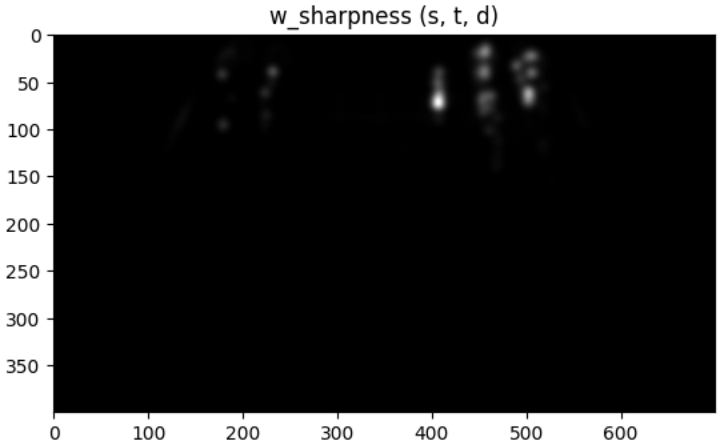
Result:



4. Compute the sharpness weight by blurring the square of high-frequency component with another Gaussian kernel of standard deviation σ2:

$$w_{\text{sharpness}}(s,t,d) = G_{\sigma_2}(s,t) * \left(I_{\text{high-freq}}(s,t,d)\right)^2. \tag{6}$$

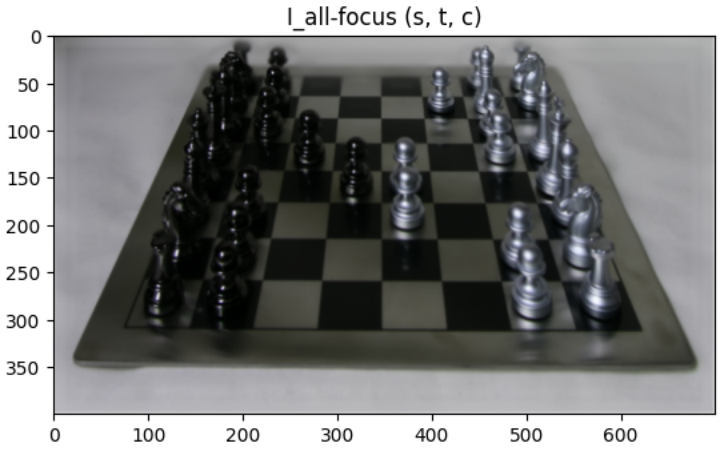
Result: sigma2: 4.5



5. Note that the weights are the same for each of the color channels. Once we have the sharpness weights, we can compute the all-focus image as:

$$I_{\text{all-focus}}(s,t,c) = \frac{\sum_d w_{\text{sharpness}}(s,t,d)\, I(s,t,c,d)}{\sum_d w_{\text{sharpness}}(s,t,d)}. \tag{7}$$

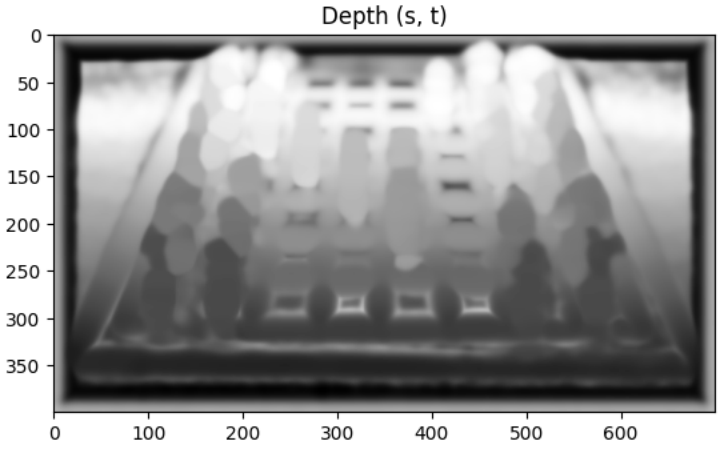
Result:



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

$$\text{Depth}\,(s,t) = \frac{\sum_d w_{\text{sharpness}}\,(s,t,d)\,d}{\sum_d w_{\text{sharpness}}\,(s,t,d)}. \tag{8}$$

Result: Depth range: 0.002241072629485325 1.9093179706396999



Discussion :

The depth map appears smoother in this iteration. However, there are some inaccuracies in depth estimation, particularly noticeable in the squares of the checkerboard. The issue arises where the white squares in the foreground and black squares in the background have been incorrectly estimated, contrary to their expected positions (similar to the chess pieces). This discrepancy may be attributed to the inherent smoothness of the image, making Gaussian filtering less effective in correcting these areas.

As observed, the inaccurately estimated depth of the squares had negligible impact on their appearance. This is likely due to their smooth texture and uniform color, which means any defocus blurring effect within those squares wouldn't noticeably affect the image.