MIT EECS 6.815/6.865: Assignment 10:

Light fields, refocusing andy focal stacks

Due Monday April 23 at 9pm

1 Summary

- light field loader
- aperture and epipolar slice visualizations
- image translation
- Light field refocusing
- Sharpness evaluation
- Focal stacks blending
- Focal stack index visualization
- Full-focus image from a light field.

2 Light fields

We will represent light fields as 4D arrays of color, encoded as 5D `numpy` array. The first two dimensions index the aperture vertical and horizontal coordinates, and the remaining 3 dimensions are regular y, x, channel images.

Light fields can represent huge amounts of data, which is why we provided you with downsampled versions indicated by name postfixes. We encourage you to use the `tiny` versions for debugging but you should turn in results for the "small-full-" versions.

2.1 Light field loader

We provide you with light fields organized into discrete picture files named as `name-XXX.png` where `name` is the base name (e.g. `chess` or `jewel`) and `XXX` is a three-digit picture index starting at 000. The pictures correspond to views from points on the aperture, organized from top to bottom and left to right. There are always NxN files (the two dimensions of the aperture are sampled equally).

Write a function `loadLF(path='chess/chess-tiny-')` that takes as input a base path and outputs a 5D light field array.

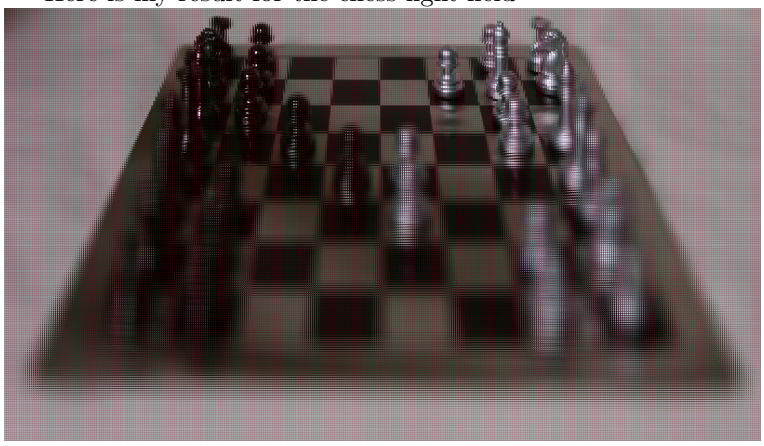
2.2 Visualizations

Write a function `apertureView(LF)` that outputs an image visualizing the light field as a set of aperture views. Assuming the size of the y and x dimensions of the light field are h and w, the aperture visualization should be an Nh×Nw picture composed of h×w sub-pictures representing the value of each pixel in each of the $N \times N$ views.

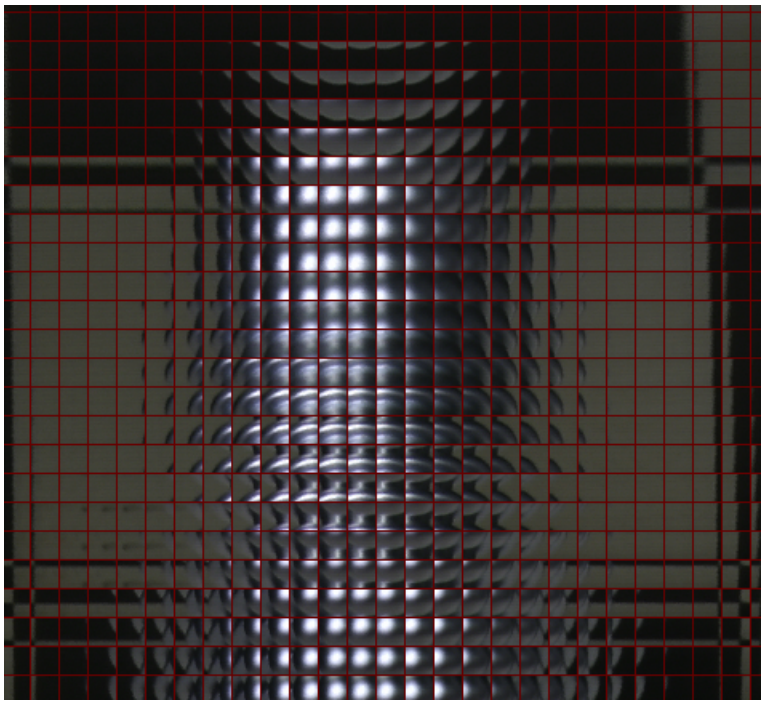
Turn in your output for the lego light field.

Bonus: add a 1-pixel boundary between the aperture views.

Here is my result for the chess light field



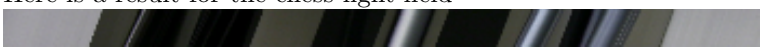
and a close up to see the sub-apertures



Write a function `epiSlice(LF, y)` that visualizes a horizontal slice of the light field corresponding to a fixed y and v. Use the v in the center of the aperture. The images will be wide and not tall because the sampling in x is much denser than in u.

Turn in your results for the small lego light field for $y = 100$.

Here is a result for the chess light field



2.3 Image shifting

In order to refocus light field, we need to sum over the individual images translated by an amount that depends on the focusing distance. For this, implement a function `ytranslateIm(im, dy, dx):` that takes as input an image and shifts it by floating point coordinates `dy` and `dx`. We will need to use this function many times, so make it fast using either `scipy.ndimage.map_coordinates` or `scipy.ndimage.interpolation.affine_transform`.

2.4 Refocusing

Write a function `refocusLF(LF, maxParallax=0.0, aperture=17)` that takes as input a light field and outputs a focused image by summing over u and v. Only `aperture` $\times$ `aperture` of the light field images should be used, which simulates a finite aperture. Each image is shifted along x and y by an amount proportional to its distance from the center of the aperture. The resulting shift

should be `maxParallax` for images at the boundary of the full aperture (e.g. 0 and 16 for the full light fields) regardless of the fraction of the aperture actually used.

Be careful, the u dimension is not in the same orientation as the x one and the sign should be flipped.

2.5 Rack focus

Write a function `rackFocus(LF, minmaxPara=-7, maxmaxPara=2.0, step=0.5, aperture=17)` that writes to disk images focused between `minmaxPara` and `maxmaxPara` in steps of `step`.

Generate a rack focus sequence for the lego light field with a full aperture (17) and with an aperture of 5.

3 Focal stack

We will generate all-focus images from a sequence of images focused at different depth. This is particularly useful for macro photography where depth of field is a challenge. We will first evaluate the sharpness of each pixel in the various input images, and then will combine them with weights depending on sharpness.

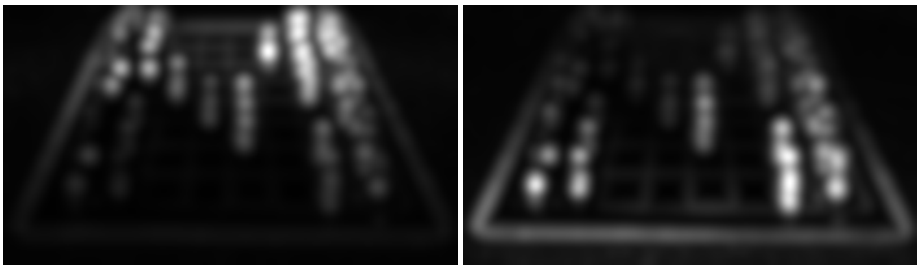
3.1 Sharpness evaluation

Write a function `sharpnessMap(im, exponent=1, sigma=1)` that returns a 3-channel image that measures the amount of high frequencies in the neighborhood of each pixel.

We will work in the luminance domain (I use my usual weights of 0.3, 0.6, 0.1) to make sure that the weights are the same for all three channels. Compute a high pass by subtracting a Gaussian blur with standard deviation `sigma`. Then to compute the local high frequency energy, we will take a local weighted average of the square of the high pass. Otherwise, the negative and positive components of the high pass would cancel each other. So take the square of your high pass and apply a Gaussian blur of standard deviation 4 `sigma`.

The additional parameter `exponent` is to increase the contrast of the sharpness maps and make sure that the sharpest image has a much higher value than the other ones. Raise the result of the second Gaussian blur to this exponent.

Here are two example of sharpness map for the chess sequence. I used `exponent=1` because it is easier to bug.



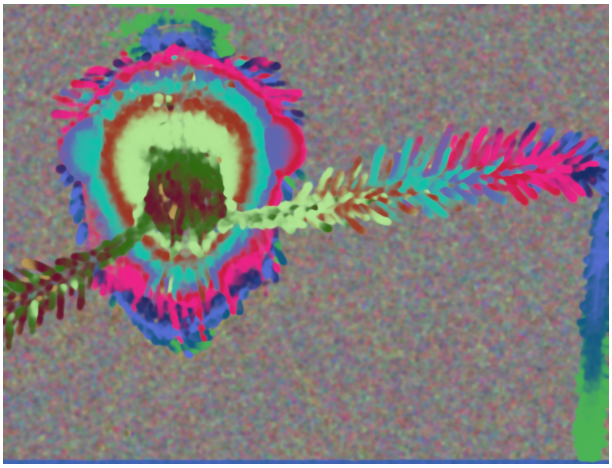
3.2 Compositing

Write a function `fullFocusLinear(imL)` that takes a sequence of images as input and uses the above sharpness computation to generate an all-focus images. The output should be an image, the sum of the input images weighted by their sharpness maps.

Turn in your results for the chess and bug sequences.

3.3 Index visualization

To better understand the algorithm, create a visualization of which image influences each pixel in the full-focus image. Here is one example I created where each image index is represented by a random color.



3.4 Putting it all together

Use your light field refocusing and focal stack blending to create an all-focus image of the lego dataset, using the full aperture data to generate an image as sharp everywhere as the image from the center of the aperture.

The advantage of this approach is that the resulting image has lower noise than the center image.

4 Extra credits

Virtual shutter: use the same principle as the light field to control motion blur. Start from a movie sequence and "focus" it at different velocities by shifting the individual images. You can use alignment (brute force or with your features) to automatically tune to the appropriate velocity.

Tilt lens effects and Scheimpflug photography from a light field (see e.g. `http://en.wikipedia.org/wiki/Scheimpflug_principle`

Generate an arbitrary view outside the plane of the aperture from a light field.

Explore different ways to select pixels, e.g. using the max of sharpness, or use a cross bilateral filter to get a better sharpness map and improved behavior at occlusion boundaries.

5 Submission

Turn in your images, python files, and make sure all your functions can be called from a module `a10.py`. Put everything into a zip file and upload to Stellar.

Include a `README.txt` containing the answer to the following questions:

- How long did the assignment take?
- Potential issues with your solution and explanation of partial completion (for partial credit)
- Any extra credit you may have implemented
- Collaboration acknowledgement (but again, you must write your own code)
- What was most unclear/difficult?
- What was most exciting?

Images:

- Aperture image for the lego light field
- Epipolar slice for the lego light field
- Rack focus set of PNGs for the lego light field for both apertures.
- Sharpness map for the bug images
- Index visualization
- Full focus image for the bug and chess sequences.