MIT EECS 6.815/6.865: Assignment 10:

Light fields, refocusing andy focal stacks

Due Monday April 23 at 9pm

1 Summary

- aperture and epipolar slice visualizations
- image translation
- Light field refocusing
- Sharpness evaluation
- Focal stacks blending
- Depth map calculation
- Full-focus image from a light field.
- Lytro fun!

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.

We will represent focal stacks as 3D arrays of color, encoded as 4D `numpy` arrays. The first dimension is correlated with the focal distance and the remaining 3 are regular images.

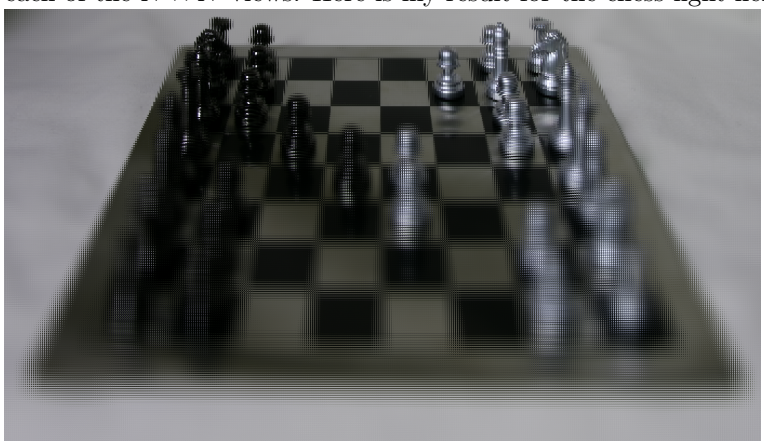
Light fields can represent huge amounts of data, which is why we provided you with downsampled versions indicated by name postfixes. We encourage you to use the smaller (`eg 'chess3x3.npy'`) versions for debugging but you should test on the larger datasets before turning your assignment in.

NOTE: For various legal and technical reasons we are still in the process of making the Lytro cameras available. We will send out an announcement with a link/instructions soon.

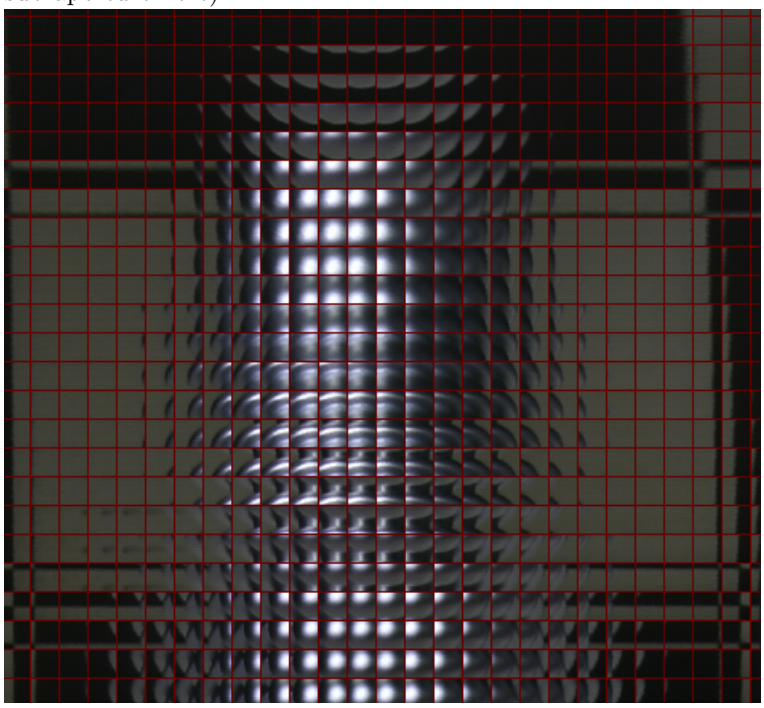
2.1 Visualizations

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. Here is my result for the chess light field



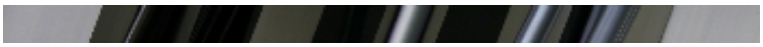
and a close up to see the sub-apertures (I've added borders between each sub aperture here)



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



2.2 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 `shiftFloat(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`.

2.3 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 (so an image at the center is not shifted at all). The difference in shift between the leftmost and rightmost images (or, alternatively, the top and bottom) should be `maxParallax`, regardless of how big the aperture is.

Be careful, the u dimension is not in the same orientation as the x one and so the sign of your shifts should be flipped. Small (or negative) `maxParallax` values should be focused close to the camera.

Your results with the `tiny` dataset will have ghosting artifacts in the out of focus regions because that down sampled data contains only a subset of the original u,v values. Don't worry too much about it.

2.4 Rack focus

Write a function `rackFocus(LF, aperture=8, nIms = 15, minmaxPara=-7.0, maxmaxPara=2.0)` that returns a focal stack in the form of a numpy array `stack[image, y, x, color]`. There should be `nIms` images, the first one should start with `minmaxPara` as its `maxParallax`, and the remaining `maxParallax` values should be uniformly distributed up to `maxmaxPara` (for the last image).

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.

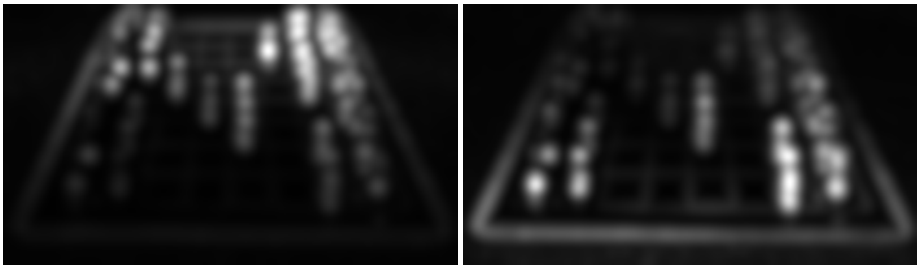
3.1 Sharpness evaluation

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 (use the 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, 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 debug, then I multiplied the result by a big constant (something on the order of 100) so the numbers would be big enough to visualize.



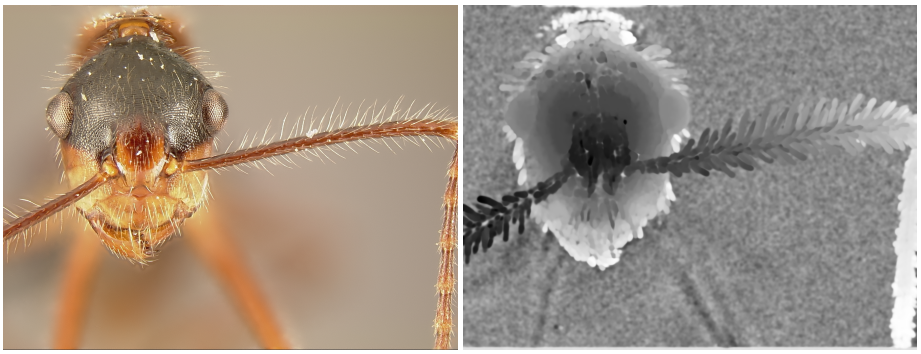
3.2 Compositing

Write a function `fullFocusLinear(stack, exponent=1.0, sigma=1.0)` that takes a 4D numpy array [image, y, x, color] as input and uses the above sharpness computation to generate an all-focus image. The output `out` should be an image - the sum of the input images weighted by their sharpness maps. (NOTE: this function will also return another image, see below)

3.3 Depth from defocus

In theory, our composite should weigh pixels that are in focus more heavily than pixels that are out of focus. And since each image in our focal stack is focused at a different depth, our weights should be correlated with depth. We will use this fact to visualize a *depth map* - an image where the intensity of each pixel is correlated with the distance from the camera to the point that pixel images. We'll assume that our n images are focused at n uniformly distributed depths ranging from [0.0,1.0]. Use the same weights you used to compute your all-focus

image, but this time average the depths corresponding to each of your images instead of their pixel values. Finally, take the square of this weighted average (to give your map a little more contrast). Have your `fullFocusLinear()` function return your depth map as `zmap`. Below are my results for the `BugStack.npy` using `exponent=3, sigma=1`.



3.4 Denoising with light fields

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.

Include your image in your submission as `'denoisedLego.png'` NOTE: The lego data set is very big, you will recieve instructions on how to obtain it soon.

4 Lytro

Lytro has generously donated a number of Lytro cameras for us to use in this class. Soon we will be splitting you up into small groups and you will use these cameras to capture your own light fields data. We're currently setting up a server that will let you convert lytro files into our focal stack numpy arrays. The last part of this assignment will involve taking a photo with the Lytro and using it as input to your `fullFocusLinear()` function. More details and instructions to follow.

5 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.

6 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?