# Exercise 1 Extensions

## Extension 1: ESLF Files

Download the ESLF sample at http://dgd.vision/LF2018/flowers.eslf.6334.zip and inspect the file IMG_6334.eslf.png. The eslf format was introduced by the Lytro Power Tools Beta and is in common use in large datasets online. The Lytro tool applies debayering, undoes hexagonal lenslet packing, applies rectification, and applies devignetting and colour correction. The format is relatively efficient and easy to load.

Write a tool to load this file format into a 4D structure (hint: the lenslet images are 14x14 pixels wide). Try some basic filtering on the loaded file. If you get stuck there's a working example for loading ESLF files at http://dgd.vision/LF2018/LFXReadStanfordIllum.m .

Questions:

1. ESLF files are organized in lenslet order (a k,l tiling of i,j images). They are also compressed using the PNG format, which exploits similarity of adjacent pixels to losslessly compress images. What would happen if the ESLF were organized in image order instead (a i,j tiling of k,l images)?

2. Could the PNG format be extended to exploit similarities both between adjacent lenslet images (similarity along k,l) as well as within lenslet images (along i,j)? How much impact do you expect this to have on the compression rate?

## Extension 2: Super-Resolved Refocus

Extend the shift-and-sum filter by adding a capability for super-resolution. Either build your own function from scratch, or modify the existing function `LFFiltShiftSum`.

An easy approach is to upsample every 2D slice as you shift it, prior to adding the slices together.

Questions:

1. At which slopes are there appreciable improvement in resolution? At which slopes is there not?
2. This method operates only for objects at one depth. How could you extend it to work on scenes occupying a range of depths?

# Extension 3: Augmenting the renderer

Augment the interpolating renderer with one or more of the following:

- Create a loop to animate the camera's pose and generate animated output
- Run it on different input light fields
- Create the "vertigo" effect by translating the camera towards or away from the scene while adjusting its focal length to keep the apparent size of the subject fixed
- Add depth of field effects (finite aperture)
- Simulate motion blur
- Use depth information to improve rendering quality

Question: Describe how you augmented your renderer and any problems you encountered.

# Extension 4: Slope and Depth Estimation

Load one of the sample light fields and write a local slope estimator. Start with Matlab's gradient function, `[Li,Lj,Ll,Lk] = gradient(LF)`, taking care to get the output order correct.

Combine local slope estimates by taking the average in neighbourhoods. Use the magnitude of the slope to weight the mean.

For a light field with a calibrated intrinsic matrix, derive the metric depth from the estimated slope.

If you get stuck there's a working example at http://dgd.vision/LF2018/LFXSlopeDepth.m .

Questions:

1. For both gantry-based and lenslet-based light fields, lines of constant value tend to align closely with the i and j axes. Given this, which is more numerically stable as a measure of the line's slope, Li/Ll or its inverse? Why?
2. How can you combine information from multiple colour channels?
3. What happens if you apply a volumetric focus filter to a depth estimate?