Edward Murphy
Refocusing an Image

☺ **ATTENTION: Every time you use open this notebook, evaluate the whole notebook first. Every time you run another notebook (especially those with ClearAll["Global` *"]), rerun this entire notebook. If you do not do this, intra-document links will not work.**

Table of Contents

Click on any of the links below to jump to that section of the document.

The color scheme to this notebook is as follows: all text cells are a light brown, all input cells are light yellow, and all output cells are green.

Some output cells are interactive.

If you see thin yellow horizontal bars going across the notebook, those are hidden (as best as possible) cells that contain formatting code. To reveal these cells, select the cells and go to: Cell→Cell Properties→Open

Out[37]=

Introduction

This project introduces how to refocus an image after it has been taken. If you want a sneak peak, skip to the Animations section. There are some useful papers listed at the end of this notebook which cover certain topics in much greater detail than this report. In addition, the work presented here would not have been possible without the Stanford Light Field Archive.

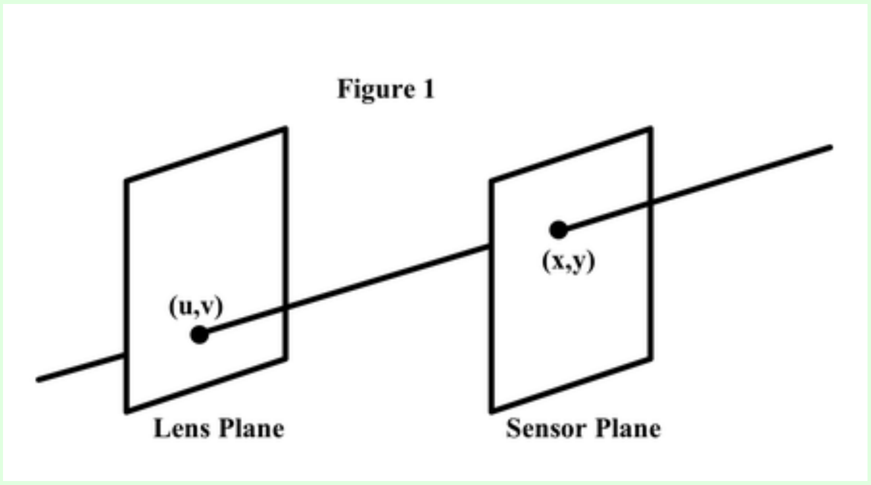
If for some reason the files referenced in this notebook do not load within Mathematica, they are located in the folder that this notebook came in.

Light Field

Light rays can be described by the plenoptic function created by Adelson and Bergen[1]. It is a 7 dimensional function given by $P(x, y, z, \theta, \phi, t, \lambda)$, where x, y, and z represent the spatial location of a point on the ray, θ and ϕ represent angular directions of the ray at the point (x, y, z), t is time, and λ is wavelength. The function is often reduced by assuming it is not changing with time (no more t), light is constant along its path (lose one of the spatial coordinates and in this case z), and the light is monochromatic (no more λ). This leaves the function as $P(\theta, \phi, x, y)$. Further, it is then useful to parametrize the plenoptic function using two planes[2]. Referring to figure 1 (interactive button), by considering the ray's intersection point on each of the two planes, the angular and spatial information of the ray is captured. Light propagates left to right in figure 1. In a light field camera setup, the first plane is thought of as the lens plane, and the second plane is thought of as the sensor plane.

Out[40]= View Figure 1 Close Figure 1

Out[41]=



In this two-plane parametrization seen in figure 1, the angular coordinates θ and ϕ are generally replaced with the spatial Cartesian coordinates u and v, giving a final function represented by $P(x, y, u, v)$. This simplified function is referred to as the light field and its notation is changed to: $L(x, y, u, v)$. Perhaps this is a bad name, because it is more of a ray than a field. The value of L is radiance[3], or the amount of light. In addition, a regular camera does not capture a light field because it does not capture the angular information of rays. However, as discussed later, using a lens array in front of the image sensor or even having a large number of sensors in a grid, captures angular information.

Imaging Equations

Referring to figure 2, an image captured by an image sensor is proportional to the irradiance at the sensor plane. The irradiance at a point (x,y) on the sensor plane is given by[4]:

$$I(x, y) = \frac{1}{F^2} \int\int L_f(x, y, u, v) \cos^4(\theta)\, du\, dv$$

where F is the distance between the exit pupil of the lens and the film, L_f is the light field parameterized by the lens and sensor plane at a separation distance of F, and θ is the angle of incidence to the planes' normals (which are assumed to have the same direction). The $\cos^4(\theta)$ represents the fact that light entering at a large θ contributes less to the irradiance measured. Notice how the irradiance at a point (x,y) on the sensor plane is found by integrating all the light fields coming to the point (x,y) from the lens plane. To simplify the irradiance equation it is assumed that the planes are infinite and that the value of L is zero beyond the bounds of the physical system being modeled. Also, the $\cos^4(\theta)$ is absorbed into the light field definition[5].

For simplicity in illustrating the following concept, in the following figures and accompanying discussion we restrict ourselves to one of the dimensions of each of the parametric planes (in this case we restrict ourselves to u and x). As a side note, located in the referenced documents are nice discussions on ray-space diagrams (not shown in this notebook), which help visualize light fields.

Focusing at different depths in a real camera consists of altering the distance between the lens and the sensor plane. This is demonstrated in figure 2 with the two-plane parametrization.

Out[43]= View Figure 2 Close Figure 2

Out[44]=

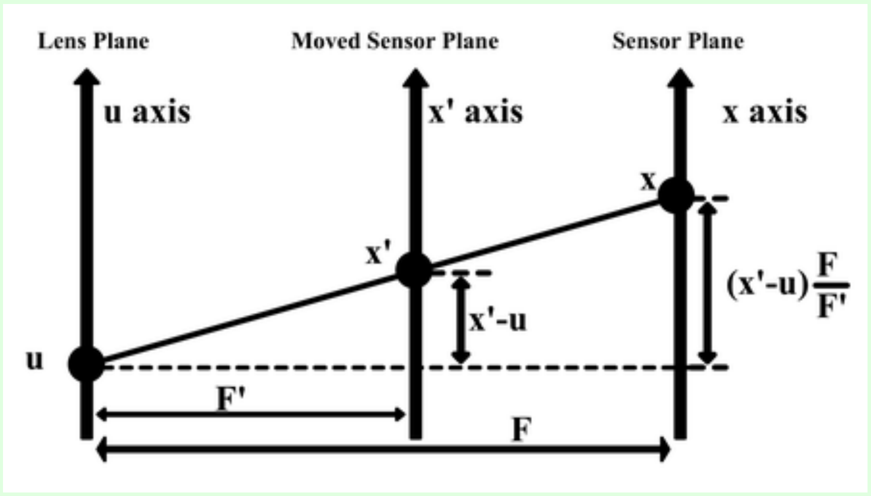


Figure 2 only depicts one dimension in each of the planes for simplicity in illustrating the concept (so the *u* dimension for the lens plane and the *x* dimension for the sensor plane). Both these dimensions operate along the vertical axis of the picture, as shown by how the axes' values increase upwards in the picture. In figure 2, the sensor plane has been moved to the left. The dots indicate points on an axis and the text next to them indicate the coordinate value. The bidirectional arrows indicate distances. Originally, the sensor plane was a distance *F* away from the lens plane. It then moved to a distance *F'* away from the lens plane. Prime (or an apostrophe) does not indicate a derivative anywhere in this notebook. It is purely to distinguish values.

In this scenario, the picture was taken at a distance of *F,* and it is desired that the image be refocused at a distance of *F'*, without having to actually retake the picture at a distance of

F'. Thus, the light field at a distance F' needs to be expressed in terms of the light field at F. Ultimately, in order to do this, the values of x' and y' need to be known, which can be found by ray tracing the ray that intersects the point (u,v) and (x,y). However, as is shown later, ray tracing is not necessary.

The light field at F' is given by: $L'(x', y', u, v)$. This needs to be expressed in terms of $L(x, y, u, v)$. Looking at figure 2, the ray intersects the sensor plane at a value of $x = u + \frac{(x'-u)}{\alpha}$ where $\alpha = \frac{F'}{F}$. Similarly for the y and v' dimension, the ray intersects the sensor at a value of $y = v + \frac{y'-u}{\alpha}$. Thus the light field at F' is given by:

$$L'(x', y', u, v) = L\left(u + \frac{(x'-u)}{\alpha},\ v + \frac{y'-u}{\alpha},\ u,\ v\right)$$
$$= L\left(u\left(1 - \frac{1}{\alpha}\right) + \frac{x'}{\alpha},\ v\left(1 - \frac{1}{\alpha}\right) + \frac{y'}{\alpha},\ u, v\right)$$

where x and y have been replaced by the values found earlier.[6] (Although I must admit I don't completely understand that equality. The two ($L' = L$) might be equal because in the beginning it was assumed that light is constant along its path). Now this L' is used to calculate the irradiance $I(x', y')$ of pixel value (x', y') when the sensor is at a depth of F' and this is given by:

$$I(x', y') = \frac{1}{\alpha^2 F^2} \iint L\left(u\left(1 - \frac{1}{\alpha}\right) + \frac{x'}{\alpha}, v\left(1 - \frac{1}{\alpha}\right) + \frac{y'}{\alpha}, u, v\right) du\, dv$$

In order to proceed with this equation, a knowledge of what a light field camera is and what it captures is necessary.

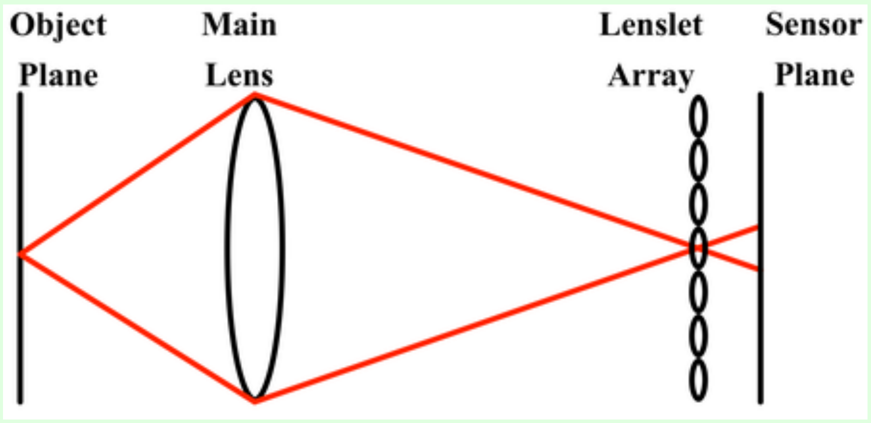
Light Field Camera

An question still unanswered in this report is, what is a light field camera? A light field camera system is depicted in figure 3. Recall that angular information of the light rays is necessary in order to capture a light field. The red lights in the figure show how the lenslet (mircolens) array separates rays from different directions, preserving angular information. The Lenslet array in the figure is placed a certain distance (calculated using the lens equation) away from the main lens, so that the light converges on it, causing the light to diverge to the right of it.

Out[46]=

View Figure 3 Close Figure 3

Out[47]=



Of interest are the sub-aperture images. These images are the pictures of the scene from the different points (u,v) on the main lens (the aperture), which means they each have a slightly different view of the scene[7].

An alternative method for capturing light field information is to have a large array of cameras (like that which took the pictures for the Stanford Light Field Archive Database). Each image captured by each camera can be thought of as a sub-aperture image.

Digitally Refocusing an Image

Recall that the irradiance $I(x', y')$ of pixel value (x', y') when the sensor is at a depth of F' is given by:

$$I(x', y') = \frac{1}{\alpha^2 F^2} \iint L\left(u\left(1 - \frac{1}{\alpha}\right) + \frac{x'}{\alpha}, v\left(1 - \frac{1}{\alpha}\right) + \frac{y'}{\alpha}, u, v\right) du\, dv$$

This equation tells us that “is conceptually a summation of dilated and shifted versions of the sub-aperture images over the entire uv aperture” [8]. The images are scaled by a factor of α and translated by a factor of $u(1 - 1/\alpha) + x'/\alpha,\ v(1 - 1/\alpha)$[9].

Refocusing Animations

In order to implement digital refocusing, I used Mathematica to translate the sub-aperture images that I got from the Stanford database (I did not explore scaling the images). Images were translated in both the x and y directions by an arbitrary number multiplied by there distance away from the center image in both the x and y directions. The arbitrary number specified the refocus depth. The code ensures that objects are always brought translated the center. The code is in the next section.

Below I have created animations showing a changing focus for four different scenes. Most of the color blurriness, and some of the pixelated-ness, is due to the conversion to a small enough gif animation (they are still around 10 MB though). If you wish to control the animation, click pause, and then move the slider circle. Click close animation when you are done with it as that will keep the memory in use down.

Below each animation is an image taken by one of the cameras in the array.

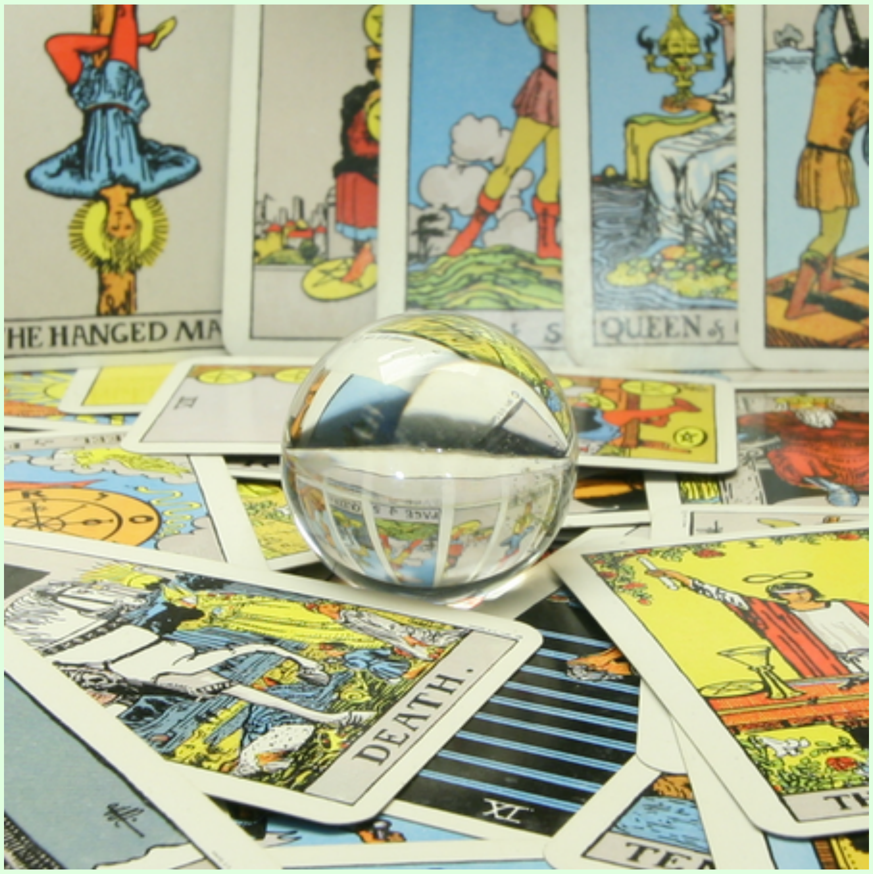
If for some reason these animations do not load within Mathematica, they are located in the folder that this notebook came in.

Out[50]= View Card Animation | Close Card Animation

Out[51]= doc1

Out[52]= View Cards Sub Aperture Image | Close Image

Out[53]=



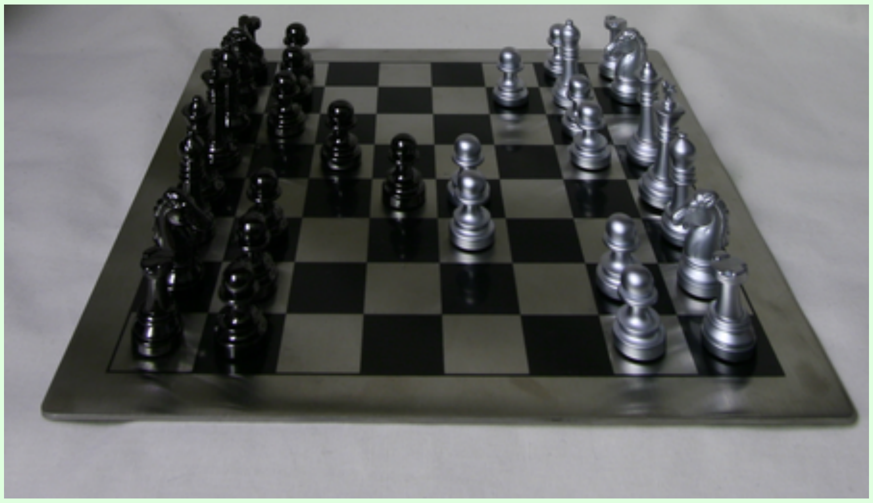
Out[54]= View Chess Animation | Close Chess Animation

Out[55]= doc2

Out[56]=

```
View Chess Sub Aperture Image  Close Image
```

Out[57]=



Out[58]=

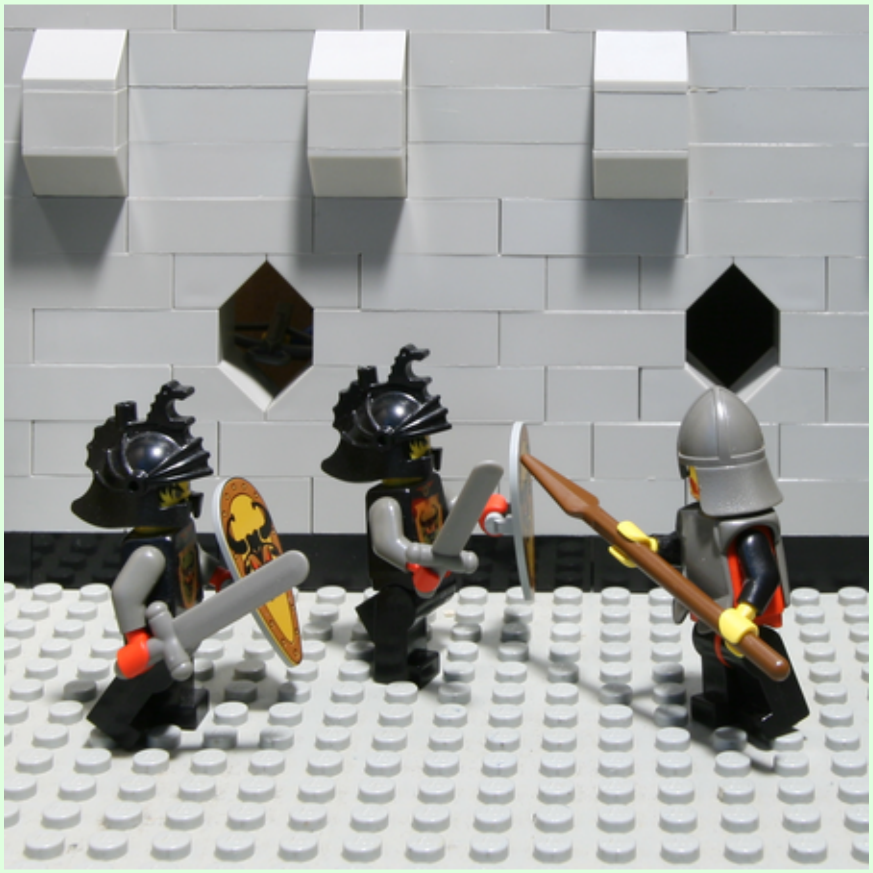
```
View Lego Animation  Close Lego Animation
```

Out[59]=

```
doc3
```

Out[60]= View Lego Sub Aperture Image Close Image

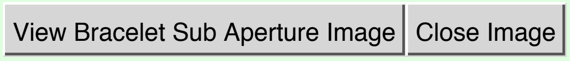
Out[61]=



Out[62]= View Bracelet Animation Close Braclet Animation

Out[63]=
```
doc4
```

Out[64]=



Out[65]=

Refocusing Code

The following is code set to not evaluate. This is the code I used to create the images that were then used to create the animation above. It is coded for Linux and macOS although only a single line needs to be changed (as is noted in a comment on that line) to work with windows.

This code assumes the notebook that it operates in is located with the images you wish to use to digitally refocus the scene. The code generally works by mapping a function over a list of images.

I need to read in the paths of all the images to be used:

In[•]:=
```
set = FileNames["*png", NotebookDirectory[]][[ ;; ]];
```

Assuming 289 images (from Stanford Light Field Archive), I am finding the coordinates of the center image:

```
In[•]:= center = set[[145]];
```

```
In[•]:= lencenter = Length[FileNameSplit[center]];
```

```
In[•]:= centername = FileNameSplit[center][[lencenter]]
```

```
In[•]:= lencentername = StringLength@centername
```

```
In[•]:= centername = StringTake[centername, lencentername - 4]
```

```
In[•]:= findcenter[s1_] :=
         Module[{newstring = s1, splitnewstring, wcurrent, hcurrent, centerwidth,
            centerheight}, splitnewstring = StringSplit[newstring, "_"];
          wcurrent = ToExpression@splitnewstring[[5]];
          hcurrent = ToExpression@splitnewstring[[4]];
          centerwidth = wcurrent;
          centerheight = hcurrent;
          {centerwidth, centerheight}]
```

```
In[•]:= wcenter = findcenter[centername][[1]]
```

```
In[•]:= hcenter = findcenter[centername][[2]]
```

Here I get the dimensions of images (assuming they are the constant) from one of the images:

```
In[•]:= centerim = Import@center;
```

```
In[•]:= ratio = ImageMeasurements[centerim, "AspectRatio"];
```

```
In[•]:= height = ImageMeasurements[centerim, "Dimensions"][[1]];
```

```
In[•]:= width = ImageMeasurements[centerim, "Dimensions"][[2]];
```

```
In[•]:= Clear[centerim]
```

Here is the function used to translate the images by a certain amount kw and kh.

In[•]:=
```
fun[image_, kw_, kh_] :=
 ImageTransformation[image, {#[[1]] + kw , #[[2]] - kh ratio} &,
  Resampling → "Bilinear", Padding → 0]
```

This function calculates the distance (in both the x and y directions) the current image is from the center image.

```
fun2[s1_] :=
 Module[{set1 = s1, list, len, newstring, splitnewstring, newstringnew,
    wcurrent, hcurrent, diffw, diffh}, list = StringSplit[set1, "/"];
   (*this must be changed to forward slash for Windows*)len = Length[list];
  newstring = ToString[list[[len ;;]][[1]]];
  newstringnew = StringTake[newstring, StringLength[newstring] - 4];
  splitnewstring = StringSplit[newstringnew, "_"];
  wcurrent = ToExpression@splitnewstring[[5]];
  hcurrent = ToExpression@splitnewstring[[4]];
  diffw = wcurrent - wcenter;
  diffh = hcurrent - hcenter;
  {diffw, diffh}]
```

This function gives the output of the second function to the first function to translate the current image by the desired amount.

In[•]:=
```
fun3[string_, maxdiffw_, maxdiffh_, resfac_, c_] :=
 Module[{st = string, maxdw = maxdiffw, maxdh = maxdiffh,
    curdiffw, curdiffh, factor = resfac, curr}, curr = fun2[string];
  curdiffw = curr[[1]];
  curdiffh = curr[[2]];
  fun[ImageResize[Import[string], {width / factor, height / factor}],
   c curdiffw / (1,000 maxdw), c curdiffh / (1,000 maxdh)]]
```

Below I calculate the biggest distances from the center. It is used to normalize the translate amounts. The function Image Transformation (found in function 1) is very picky and requires values between 0 and 1. Notice the divide by 1000 on the last line in function 3. This was needed to keep the function Image Transformation from translating the image too much. The value of 1000 was found experimentally.

```
finlist = fun2[#] & /@ set;
```

```
In[•]:= diffws = Abs[finlist[[All, 1]]];
```

```
In[•]:= diffhs = Abs[finlist[[All, 2]]];
```

```
In[•]:= maxdiffw = Max[diffws];
```

```
In[•]:= maxdiffh = Max[diffhs];
```

Below I generate a list of refocusing depths I want to try. The correct range of these is found experimentally (and also are related to the value of 1000 I found earlier) and the numbers shown here (for some particular scene) are not the same for a different scene.

```
In[•]:= clist = Array[# &, 131, {-25, 40}] // N
```

Here is the final function that takes in a desired focus depth. It also allows the images to be downsampled to speed up the computation.

```
In[•]:= ex[cva_, resam_] :=
         Export[NotebookDirectory[] <> "focused/c=" <> ToString[cva] <> ".png",
          ImageResize[Image[Fold[#1 + ImageData@fun3[#2, maxdiffw, maxdiffh, resam, cva] &,
              0 set[[1]], set] / Length@set], {width, height}, Resampling → "Bilinear"]]
```

This maps the previous function of the list of focus depths, generating a number of images focused at different depths.

```
In[•]:= ex[#, 4] & /@ clist
```

The following codes reads in the list of refocused images and animates them. It creates two animations, one at a higher resolution than the other.

```
In[]:= names = FileNames["*.png", NotebookDirectory[]];
```

```
In[]:= names2 = names[[
          Ordering@PadRight@StringSplit[names, x : NumberString :→ ToExpression@x]]];
```

```
In[]:= imagelist = Import@# & /@ names2;
       Export[NotebookDirectory[] <> "/try1.gif", imagelist,
        "AnimationRepetitions" → ∞, "DisplayDurations" → .2]
```

```
In[]:= testim = Import[names2[[1]]]
```

```
In[]:= height = ImageMeasurements[testim, "Dimensions"][[1]]
```

```
In[]:= width = ImageMeasurements[testim, "Dimensions"][[2]]
```

```
In[]:= Clear[testim]
```

```
In[]:= imagelist =
         ImageResize[Import@#, {width / 2, height / 2}, Resampling → "Bilinear"] & /@
          names2;
       Export[NotebookDirectory[] <> "/try2.gif", imagelist, "AnimationRepetitions" → ∞,
        "DisplayDurations" → .2, ImageResolution → 200]
```

```
Clear[imagelist]
```

```
CellPrint@
 Cell["References", "Chapter"(*format style option*), CellTags → {"ref"}]
```

That’s it!

References

Below are the numbers corresponding to the superscript references.

For each number there is a page number and an abbreviation to the document the reference is from.

ng refers to:

R. Ng, “Digital light field photography,” Ph.D. thesis (Stanford University), 2006.

josaa refers to:

Edmund Y. Lam, “Computational photography with plenoptic camera and light field capture: tutorial,” Journal of the Optical Society of America, Vol. 31, No.11, November 2015.

1. 2022 josaa

2.13 ng

3. 2023 josaa

4. 19 ng

5. 20 ng

6. 20 ng

7. 31 ng

8. 54 ng

9. 55 ng