NUCS 331 - Introduction to Computational Photography
Northwestern University
Summer 2021
Name: Frank Yang, Priyanka Amin
NetID: fyx5811 psa5465
E-Mail: frankyang2024@u.northwestern.edu, priyankaamin2024@u.northwestern.edu
GitHub-Repo: https://github.com/NUCS331/summer2021-hw6-lightfields-yyf20001230
GitHub-Name: yyf20001230, priyanka-amin

Homework 6 - Lightfields

Abstract

Light field imaging is great for capturing images with fast moving objects in them. It helps to accurately portray these objects and limits defocusing effects on the outcome image. In addition, it can assist with digital refocusing. Some instances where light field imaging can be used is to focus through a splash of water or a dramatic gun fight with many fast-moving bullets.

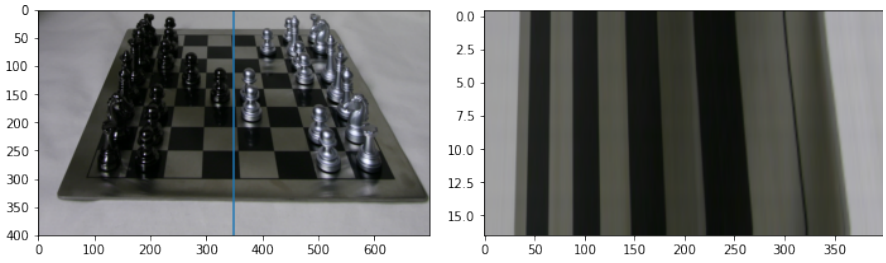
1 Introduction

Light fields can be seen as a virtual camera that is defined as "radiance as a function of position and direction in regions of space free of occluders" [1]. We can use light fields to help collect any 2D subset of light rays and create a new image from it. The light field is 5D (viewpoint is 3D and ray direction is 2D) but in free space, it is 4D. There are two methods to acquire light fields. The first is through camera arrays which are composed of many small cameras arranged in a grid where each position is giving one image from a uv coordinate. The second is a micro-lens array which is where you place a micro-lens array that decomposes the rays from the camera to a new photosensor plane where intensity is captured from different ray directions. There is a tradeoff between spatial and angular resolution as each micro-lens array will only correspond to a single pixel. However, it can also tell the distribution of all incoming rays. There are many different applications with light fields. These include light field rendering, creating novel projections, digital refocusing, digitally stopping-down, creating synthetically large apertures, light field displays, and video stabilization.

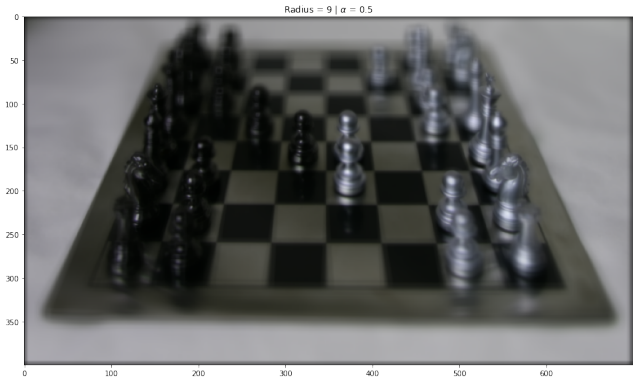
2 Results

3 Conclusion

In this assignment, we implemented various functions to help us work with light fields and synthetic apertures. We were able to extensively test a very important application of light fields: refocusing. We feel as though this assignment, especially with its vast coding section, really helped us understand the concepts and uses of light fields much more.

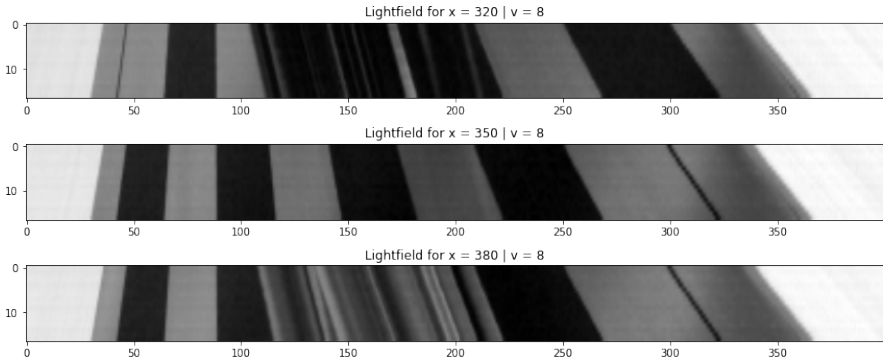


(a) Bilinear 1D refocusing example



(b) 2D refocused lightfield example

Figure 1: These images depict our results from the digital refocusing problem. This is one of the major applications of light fields. Refocusing is the summing of windows extracted from several micro-lenses. We were interested in this section, especially when we messed around with the sliders.

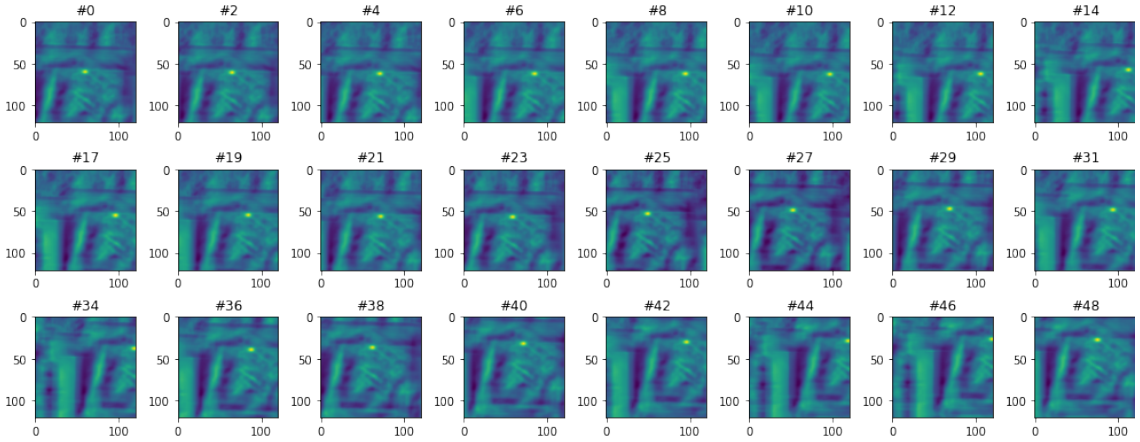


(a) Bilinear lightfield for x = 320, x = 350, x = 380 | v = 8

Figure 2: These images display our 1D-lightfield. The values are changing between black and gray based on the colors of the chessboard image. There are significant differences between the 3 images at around 130 to around 200. You see mostly black at x = 320 because there is a black chess piece there on the original image. You do not see anything at x = 350 because there is no chess piece there on the image. And you see silver at x = 380 because there is a silver chess piece there on the original image.

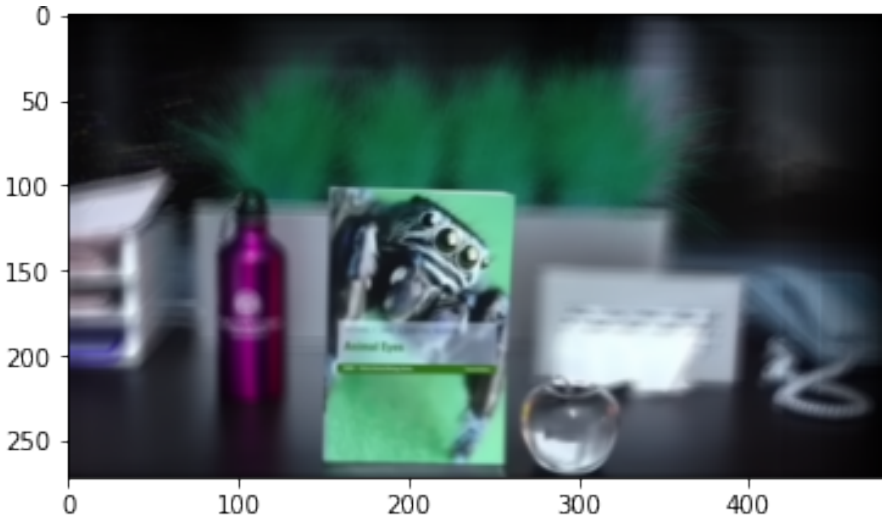
References

[1] Marc Levoy. Light fields and computational photography.



(a) Visualizing the correlation in all frames

Figure 3: Our results from our correlation of all frames. The lower images are the originals while the top ones (with the red points) signify the correlation and maps to the yellow points on the original images. We thought these results were satisfying when our red points exactly matched up with the yellow spot on each image, especially when all of them lined up (without any points being messed up).



(a) Synthetic aperture image

Figure 4: This is our synthetic image. We think the way this image has been refocused has resulting in a strange, yet nonetheless cool image. It looks like there are some odd artifacts near the top and right side of the image but it its not that noticeable.

Coding Section

External listing highlighting - lightfield.py

```
import glob
import numpy as np
import matplotlib.pyplot as plt
import cv2
import scipy
import scipy.misc
from matplotlib import gridspec
import matplotlib.pyplot as plt
import matplotlib.patches as patches
from PIL import Image
import numpy as np
import skimage
import skimage.transform as skimage_transform

import os


def bilinear_interpolate_numpy_matrix(data, x, y):
    x0 = np.floor(x).astype(int)
    x1 = x0 + 1
    y0 = np.floor(y).astype(int)
    y1 = y0 + 1

    wa = (x1-x) * (y1-y)
    wb = (x1-x) * (y-y0)
    wc = (x-x0) * (y1-y)
    wd = (x-x0) * (y-y0)

    x0 = np.clip(x0, 0, data.shape[1]-1)
    x1 = np.clip(x1, 0, data.shape[1]-1)
    y0 = np.clip(y0, 0, data.shape[0]-1)
    y1 = np.clip(y1, 0, data.shape[0]-1)

    Ia = data[ y0, x0 ]
    Ib = data[ y1, x0 ]
    Ic = data[ y0, x1 ]
    Id = data[ y1, x1 ]

    return (Ia*wa) + (Ib*wb) + (Ic*wc) + (Id*wd)

def bilinear_interpolate_numpy_2D_image(data, x, y):
    x0 = np.floor(x).astype(int)
    x1 = x0 + 1
    y0 = np.floor(y).astype(int)
```

```
    y1 = y0 + 1

    wa = (x1-x) * (y1-y)
    wb = (x1-x) * (y-y0)
    wc = (x-x0) * (y1-y)
    wd = (x-x0) * (y-y0)

    x0 = np.clip(x0, 0, data.shape[1]-1)
    x1 = np.clip(x1, 0, data.shape[1]-1)
    y0 = np.clip(y0, 0, data.shape[0]-1)
    y1 = np.clip(y1, 0, data.shape[0]-1)

    Ia = data[ y0, x0 ]
    Ib = data[ y1, x0 ]
    Ic = data[ y0, x1 ]
    Id = data[ y1, x1 ]

    return (Ia*wa) + (Ib*wb) + (Ic*wc) + (Id*wd)


def bilinear_interpolate_numpy_RGB_image(data, x, y):
    x0 = np.floor(x).astype(int)
    x1 = x0 + 1
    y0 = np.floor(y).astype(int)
    y1 = y0 + 1


    wa = (x1-x) * (y1-y)
    wb = (x1-x) * (y-y0)
    wc = (x-x0) * (y1-y)
    wd = (x-x0) * (y-y0)

    wa = np.dstack([wa] * 3)
    wb = np.dstack([wb] * 3)
    wc = np.dstack([wc] * 3)
    wd = np.dstack([wd] * 3)

    x0 = np.clip(x0, 0, data.shape[1]-1)
    x1 = np.clip(x1, 0, data.shape[1]-1)
    y0 = np.clip(y0, 0, data.shape[0]-1)
    y1 = np.clip(y1, 0, data.shape[0]-1)

    Ia = data[y0,x0,:]
    Ib = data[y1,x0,:]
    Ic = data[y0,x1,:]
    Id = data[y1,x1,:]

    return ((Ia * wa) + (Ib * wb) + (Ic * wc) + (Id * wd)).astype(np.ui
```

```
def bilinear_interpolate_numpy(data, x, y):
    x0 = np.floor(x).astype(int)
    x1 = x0 + 1
    y0 = np.floor(y).astype(int)
    y1 = y0 + 1


    wa = (x1-x) * (y1-y)
    wb = (x1-x) * (y-y0)
    wc = (x-x0) * (y1-y)
    wd = (x-x0) * (y-y0)

    wa = np.stack([wa] * data.shape[2])
    wa = np.stack([wa] * data.shape[3])
    wa = np.stack([wa] * data.shape[4])
    wa = wa.T
    if len(wa.shape) == 5:
        wa = np.swapaxes(wa,0,1)


    wb = np.stack([wb] * data.shape[2])
    wb = np.stack([wb] * data.shape[3])
    wb = np.stack([wb] * data.shape[4])
    wb = wb.T
    if len(wb.shape) == 5:
        wb = np.swapaxes(wb,0,1)

    wc = np.stack([wc] * data.shape[2])
    wc = np.stack([wc] * data.shape[3])
    wc = np.stack([wc] * data.shape[4])
    wc = wc.T
    if len(wc.shape) == 5:
        wc = np.swapaxes(wc,0,1)

    wd = np.stack([wd] * data.shape[2])
    wd = np.stack([wd] * data.shape[3])
    wd = np.stack([wd] * data.shape[4])
    wd = wd.T
    if len(wd.shape) == 5:
        wd = np.swapaxes(wd,0,1)

    x0 = np.clip(x0, 0, data.shape[1]-1)
    x1 = np.clip(x1, 0, data.shape[1]-1)
    y0 = np.clip(y0, 0, data.shape[0]-1)
```

```
    y1 = np.clip(y1, 0, data.shape[0]-1)

    Ia = data[y0,x0,:]
    Ib = data[y1,x0,:]
    Ic = data[y0,x1,:]
    Id = data[y1,x1,:]
    print(Ia.shape)

    return ((Ia * wa) + (Ib * wb) + (Ic * wc) + (Id * wd)).astype(np.uin


def get_shift_1D(img,alpha):
    u_k = list(range(-8,9))

    return np.dot((1-alpha),u_k)


def shift_images_1d(img,shifts):
    out = np.empty((img.shape))
    for i in range(len(shifts)):
        out[i,:] = scipy.ndimage.shift(img[i,:],shifts[i],mode='mirror')

    return out




def average_1d_signal(img):
    refocused = np.mean(img,axis = 0)

    return refocused

def get_alpha_values():
    return np.array([0.5,0.75,1])


def get_shifts(lf_shape,alpha):
    u_k = [i for i in range(-8,9)]
    line =  np.dot((1-alpha),u_k)
    DY,DX = np.meshgrid(line,line[::-1])

    return DX,DY


def translate_image(img,dx,dy):
    M = np.float64([[1, 0, dy],[0, 1, dx]])
```

```
    result = cv2.warpAffine(img,M=M,dsize=(img.shape[1], img.shape[0]))

    return result


def shift_lightfield(data,DX,DY):
    for i in range(data.shape[0]):
        for j in range(data.shape[1]):
            data[i,j,:,:,:] = translate_image(data[i,j,:,:,:],DX[i,j],DY

    return data




def weight_shifted_ligthfield(data,mask = None):
    if not np.all(mask):
        return data

    mask = np.stack([mask] * data.shape[2])
    mask = np.stack([mask] * data.shape[3])
    mask = np.stack([mask] * data.shape[4])
    mask = mask.T

    return (data * mask)




def crop_part(data,A_mask = None):
    if np.all(A_mask):
        ymin = int((data.shape[0] - A_mask.shape[0])/2)
        ymax = int((data.shape[0] + A_mask.shape[0])/2)
        xmin = int((data.shape[1] - A_mask.shape[1])/2)
        xmax = int((data.shape[1] + A_mask.shape[1])/2)
        return data[ymin:ymax,xmin:xmax,:,:,:]

    else:
        return data




def average_shifted_lightfield(tmp_lf,A_mask=None):
    if np.all(A_mask):
        img = np.mean(tmp_lf,axis = (0,1))
        return img.astype(np.uint8)
```

```
    else:
        img = np.mean(tmp_lf,axis = (0,1))
        return img.astype(np.uint8)
```

External listing highlighting - synthetic-aperture.py

```
import glob
import numpy as np
import matplotlib.pyplot as plt
import cv2
import scipy
import scipy.misc
from matplotlib import gridspec
import matplotlib.pyplot as plt
import matplotlib.patches as patches
from PIL import Image
import numpy as np
import skimage

import code as code

import os


# The methods that openCV can use for correlation
methods = ['cv2.TM_CCOEFF', 'cv2.TM_CCOEFF_NORMED', 'cv2.TM_CCORR',
            'cv2.TM_CCORR_NORMED', 'cv2.TM_SQDIFF', 'cv2.TM_SQDIFF_NORM

def crop_search_window(img, window_size, template_size, focus_center = 
    x_center = img.shape[1]
    y_center = img.shape[0]

    if np.all(focus_center):
        x_center = focus_center[1] * 2
        y_center = focus_center[0] * 2

    xmin = int((x_center - window_size) / 2)
    xmax = int((x_center + window_size) / 2)
    ymin = int((y_center - window_size) / 2)
    ymax = int((y_center + window_size) / 2)
    xmin_template = int((x_center - template_size) / 2)
    xmax_template = int((x_center + template_size) / 2)
    ymin_template = int((y_center - template_size) / 2)
    ymax_template = int((y_center + template_size) / 2)
    windows = img[ymin:ymax,xmin:xmax,:,:]
    template = img[ymin_template:ymax_template,xmin_template:xmax_templa
```

```
    return windows, template, [ymin,xmin], [ymin_template,xmin_template]


def findCorrelation(img,template,method='cv2.TM_CCOEFF_NORMED'):
    method_cv = eval(method)
    res = cv2.matchTemplate(img,template,method_cv)
    _, maxVal, _, maxLoc = cv2.minMaxLoc(res)

    return res,maxLoc[::-1]


def findCorrelationAll(imgs,template,method='cv2.TM_CCOEFF_NORMED'):
    temp,_ = findCorrelation(imgs[:,:,:,0],template)
    res = np.empty((imgs.shape[3],temp.shape[0],temp.shape[1]))
    maxLoc = np.empty((2,imgs.shape[3]))
    for i in range(imgs.shape[3]):
        res[i,:,:],maxLoc[:,i] = findCorrelation(imgs[:,:,:,i],template)

    return res,maxLoc.astype(int)



def correct_correlation_result_to_image_coordinates(top_left_correlated_
    result = np.empty((top_left_correlated_list.shape))
    for j in range(top_left_correlated_list.shape[1]):
        result[0,j] = top_left_correlated_list[0,j] + template.shape[0]
        result[1,j] = top_left_correlated_list[1,j] + template.shape[1]

    return result.astype(int)


def calculate_pixel_shifts(top_left_correlated):
    pixel_shifts = np.empty((top_left_correlated.shape))
    pixel_shifts[:,0] = [0,0]
    for i in range(1,pixel_shifts.shape[1]):
        pixel_shifts[:,i] = top_left_correlated[:,i] - top_left_correlat

    return pixel_shifts


def translate_image(img,dx,dy):
    M = np.float64([[1, 0, dx],[0, 1, dy]])
    result = cv2.warpAffine(img,M=M,dsize=(img.shape[1], img.shape[0]))
```

```
    return result


def translate_all_images(imgs,pixel_shifts):
    shifted_imgs = np.empty((imgs.shape))
    for i in range(imgs.shape[3]):
        shifted_imgs[:,:,:,i] = translate_image(imgs[:,:,:,i],-pixel_shi

    return shifted_imgs.astype(np.uint8)



def average_images(imgs):
    return np.mean(imgs,axis = 3).astype(np.uint8)
```