DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING, UCLA
ECE 239AS: COMPUTATIONAL IMAGING

INSTRUCTOR: Prof. Achuta Kadambi
TA: Pradyumna Chari

NAME: Swapnil Sayan Saha
UID: 605353215

PSET 3 : LIGHT FIELDS

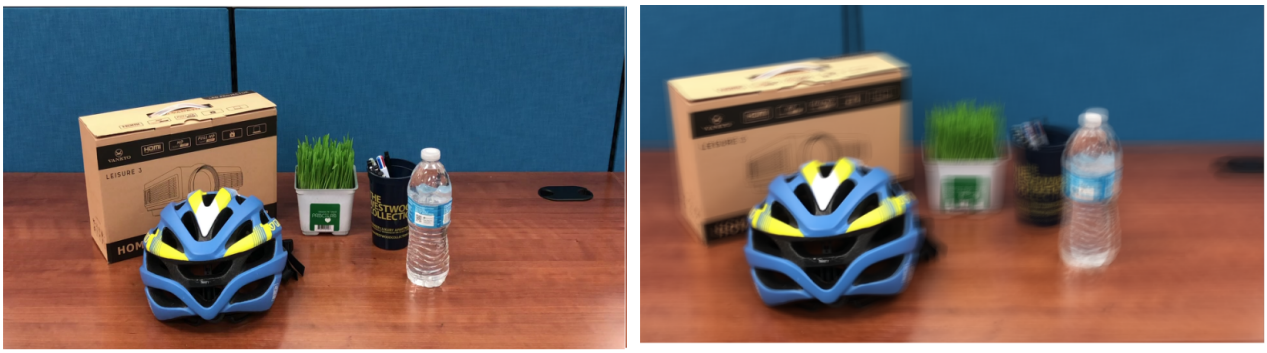
PROBLEM	TOPIC	MAX. POINTS	GRADED POINTS	REMARKS
2.1	Set up a Static Scene	0.5		
2.2	Capture a 4D Light Field	0.5		
2.3	Acquiring the Data	0.5		
2.4.1	Template and Window	0.5		
2.4.2	Normalized Cross Correlation	0.5		
2.4.3	Retrieving the Pixel Shifts	0.5		
2.5	Synthesizing an Image with Synthetic Aperture	1.0		
2.6	Repeating the Experiment for Different Templates	1.0		
3.1	Deriving the Blur Kernel Width	3.0		
3.2	Blur Kernel Shape	1.0		
3.3	Blur and Scene Depth	0.5		
3.4	Blur and Focal Length	0.5		
	Total	10		

1 Motivation

At the top-level, this problem set enables you to turn your cell phone into a 4D light field camera.

In class we learned how ”Bokeh” and shallow depth of field is a desirable aesthetic quality in a photograph. Unfortunately, this effect requires a large aperture, i.e., the lens is going to be big and bulky! But what if it was possible to turn your cell phone into a camera with a large aperture? What if we could selectively focus on objects in post-processing?

The goal of this homework is to synthesize images with smaller depths of field thus making it appear to have been taken from an expensive camera with a larger aperture[4] [2]. Figure 1a and b show a scene image with the corresponding synthetic aperture image with lower depth of field.



(a) All-in focus image.

(b) Post-processed to blur the background

Figure 1: **Turning a cell phone into a light field camera.** (a) An all-in focus image taken with a cell phone camera. (b) A light field stack is post-processed to blur out the background. Notice how the helmet stands out from the background.

2 Experimental Component

We will capture a video by moving the camera in a zig-zag path as shown in Figure 2 in front of the static scene. Unless otherwise discussed with the instruction staff, please use MATLAB for all codes. Fill in each box below for credit.

Please note:

1. The algorithm being implemented does not take camera tilt into account. Avoid tilting and rotating the camera as much as possible.
2. The instruction set use a planar zig-zag path for camera motion as in 2. However, you are allowed to try different paths like circular or polyline.
3. The number of frames in the video captured will determine the time required to compute the output. Make sure the video is not too long.

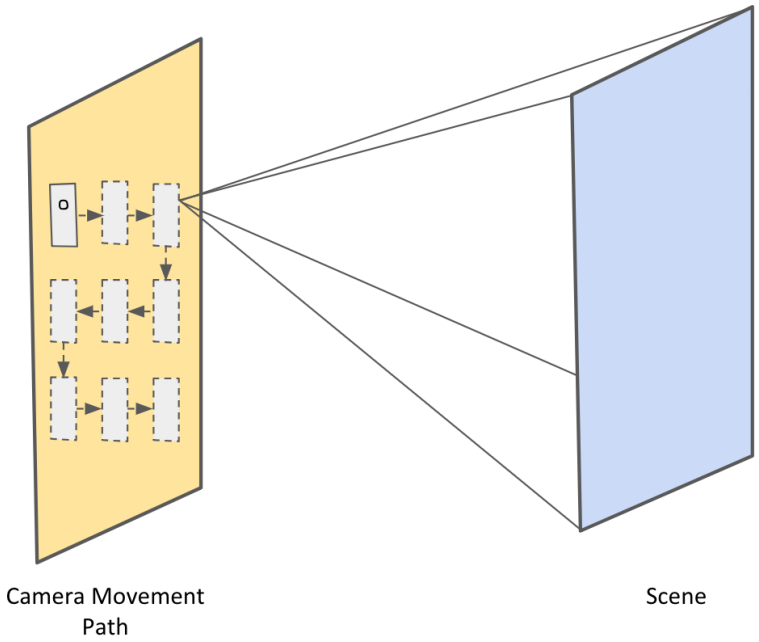


Figure 2: A zig-zag planar motion of the camera in front of the static scene to capture a video.

2.1 Set up a Static Scene (0.5 points)

Set up a static scene similar to the one shown in Figure 1a. Try to have objects at different depths. For credit, place your image in the box below (replace our helmet scene with your own scene).

Figure 3: Static scene with objects at 5 different depths.

2.2 Capture a 4D Light Field (0.5 points)

Take a video by waving your camera in front of the scene by following a specific planar motion. The more you cover the plane, the better will be your results. Ensure that all objects are in focus in your video. For credit, place three frames of the video in the box below (replace our example). These frames differ in their *parallax*, i.e., an effect where object positions change in response to view.

Figure 4: Frames 1, 102 and 224 out of 248 frames (@ 30 fps) for static scene following zig-zag planar motion (bottom to top).

2.3 Acquiring the Data (0.5 points)

Write a function to read your video file and convert the video into a sequence of frames. Since this was captured from a cell phone, each frame image is in RGB color. Write a script to convert each frame to gray-scale. For credit, place the gray scale image of the first frame of your video in the box below (replace our example).

```
%function to convert video to frames
function [frames] = vidToFrames(video_file)
    frames = read(VideoReader(video_file));
end

%script to convert video frames to grayscale frames
clear; clc;
RGBframes = vidToFrames('VID_20201108_225443.mp4');
grayframes = zeros(size(RGBframes,1),size(RGBframes,2),...
size(RGBframes,4),'uint8');
for i = (1:size(RGBframes,4))
    grayframes(:,:,i)= rgb2gray(RGBframes(:,:,:,i));
end
imshow(grayframes(:,:,1)); %show first grayscale frame
```

Figure 5: Gray-scale image of first frame.

2.4 Registering the Frames (1.5 points)

2.4.1 Template and Window (0.5 points)

From the first frame of your video, select an object as a template. We will be registering all other frames of the video with respect to this template. Once a template has been selected in the first frame, we search for it in the subsequent frames. The location of the template in a target frame image will give us the shift(in pixels) of the camera. Since we don't have to search for the template in the entire target frame image, we select a window to perform this operation. Note, however,

that selecting a window is optional. This is done just to reduce the computation time. For credit, place the image of the first frame of your video in the box below with the template and the window markings (replace our example).

Figure 6: Search space (blue) and object template (red) for frame 1.

2.4.2 Normalized Cross Correlation (0.5 points)

Perform a normalized cross correlation of the template with the extracted search window.

Let $A[i,j]$ be the normalized cross-correlation coefficient. If $t[n,m]$ is our template image and $w[n,m]$ is our window, then from [3] we have:

$$A[i,j] = \frac{\sum_{n,m=1}^{T}[w(n,m)-\overline{w}_{i,j}][t(n-i,m-j)-\overline{t}]}{\{\sum_{n,m=1}^{T}[w(n,m)-\overline{w}_{i,j}]^2[t(n-i,m-j)-\overline{t}]^2\}^{0.5}}, \tag{1}$$

where, $\overline{t}$ is the mean of the template and $\overline{w}_{i,j}$ is the mean of the window $w[n,m]$ in the region under the template. Plot the cross correlation coefficient matrix $A[i,j]$ for one of the frames. For credit, place the plot in the box below. (replace our example)

```
template = grayframes(1111:1525,1581:1864,1); %template
crr_list = zeros(1615, 1284,size(grayframes,3));
crr_max_list = zeros(2, size(grayframes,3));
for i = 1:size(grayframes,3)
    ROI = grayframes(900:2100,1200:2200,i); %window
    crr =  normxcorr2(template,ROI); %cross correlation matrix
    crr_list(:,:,i) = crr;
    %find maximum of cross correlation
    [ymax,xmax] = find(crr==max(crr(:)));
    %account for padding (black border)
```

```
        xTL = xmax-size(template,1);
        yTL = ymax-size(template,2);
        %corresponds to top left corner of the template
        crr_max_list(1:2,i) = [xTL, yTL];
    end
    imagesc(crr_list(:,:,1));
    set(gca,'FontSize',14);
    colormap gray;
    colorbar;
    xlabel('Pixel Location in X direction','FontSize',14);
    ylabel('Pixel Location in Y direction','FontSize',14);
```

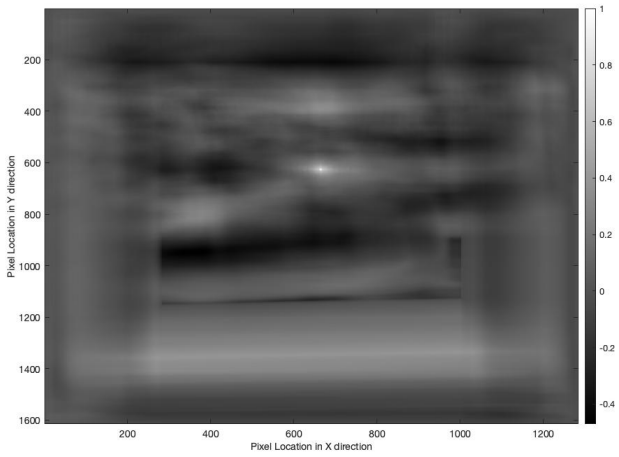


Figure 7: Plot for cross correlation coefficient matrix for first frame

[Hint: Use the MATLAB function *normx2corr* or *xcorr2* to perform the 2D cross correlation function.]

2.4.3 Retrieving the Pixel Shifts (0.5 points)

The location that yields the maximum value of the coefficient$A[i, j]$ is used to compute the shift [1]. The shift in pixels for each frame can be found by:

$$[s_x, s_y] = max_{i,j}\{A[i, j]\}. \tag{2}$$

For credit, please place the plot of s_x v/s s_y in the box below (replace our example).

```
    %shifts in pixel already found in previous section's code

    plot(crr_max_list(1,:),crr_max_list(2,:),'-x','LineWidth',2);
```

```
set(gca,'FontSize',14);
xlabel('X-Pixel Shift','FontSize',14);
ylabel('Y-Pixel Shift','FontSize',14);
grid minor;
```

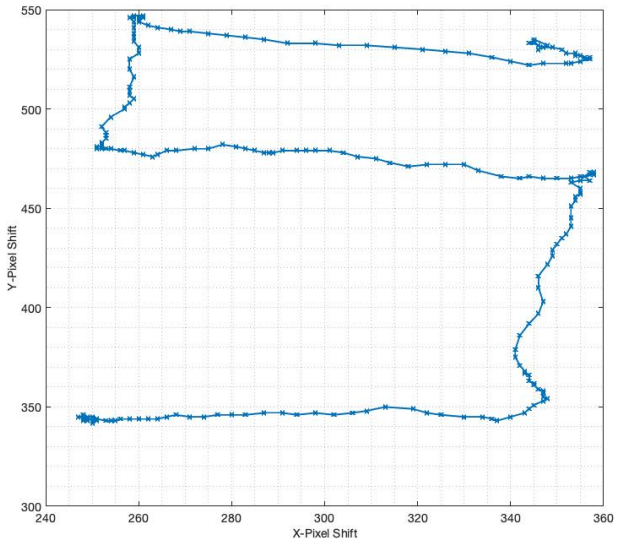


Figure 8: X pixel shift v/s Y pixel shift.

2.5 Synthesizing an Image with Synthetic Aperture (1.0 points)

Once you have the pixel shifts for each frame, you can synthesize refocused image by shifting each frame in the opposite direction and then summing up all the frames. (Note: in Section 3, you will need to explain why this operation works. Start thinking about this now!)

Suppose the pixel shift vector for Frame Image $I_i[n,m]$ is $[s_{x_i}, s_{y_i}]$. Then, the image output, $P[n,m]$ with synthetic aperture is obtained as:

$$P[n,m] = \sum_i I_i[n - s_{x_i}, m - s_{y_i}]. \tag{3}$$

For credit, place your synthetically "defocused" image in the box below (replace our example).

```
%%function to perform shifting and appending with zeros
function B=shiftMat(A,offsets)
dimA=size(A);
N=length(dimA);
if length(offsets)<N
```

```
        offsets(N)=0;
    end
    B=zeros(dimA);
    indices=cell(3,N);
    for i=1:N
        for s=[1,3]
            idx=(1:dimA(i))+(s-2)*offsets(i);
            idx(idx<1)=[];
            idx(idx>dimA(i))=[];
            indices{s,i}=idx;
        end
    end
    src=indices(1,:);
    dest=indices(3,:);
    B(dest{:})=A(src{:});
    end

    %shift and add
    x = crr_max_list(1,:);
    y = crr_max_list(2,:);
    dfocrgbframes = zeros(size(RGBframes),'uint8');
    for i = 1:size(grayframes,3)
        dfocrgbframes(:,:,1,i) = uint8(shiftMat(RGBframes(:,:,1,i),...
        [-y(:,i),-x(:,i)]));
        dfocrgbframes(:,:,2,i) = uint8(shiftMat(RGBframes(:,:,2,i),...
        [-y(:,i),-x(:,i)]));
        dfocrgbframes(:,:,3,i) = uint8(shiftMat(RGBframes(:,:,3,i),...
        [-y(:,i),-x(:,i)]));

    end
    myIm = im2double(dfocrgbframes(:,:,:,1));
    for i = 2:size(grayframes,3)
        myIm = myIm + im2double(dfocrgbframes(:,:,:,i));
    end
    o = uint8(255 * mat2gray(myIm));
    figure
    imshow(o);
```

Figure 9: Synthetic focus on the template (tomato paste), while blurring everything else.

[Hint: Use the MATLAB function *imtranslate* to perform the shifting operation.]

2.6 Repeating the Experiment for Different Templates (1.0 points)

Now, we will exploit the fact that we can synthetically focus on different depths. To do this, select a new object as your template and repeat all the steps to generate an image that is focused on this new object. Here, we have selected the cup as our new object. For credit, place a de-focused image with a different template object in focus in the box below (replace our example).

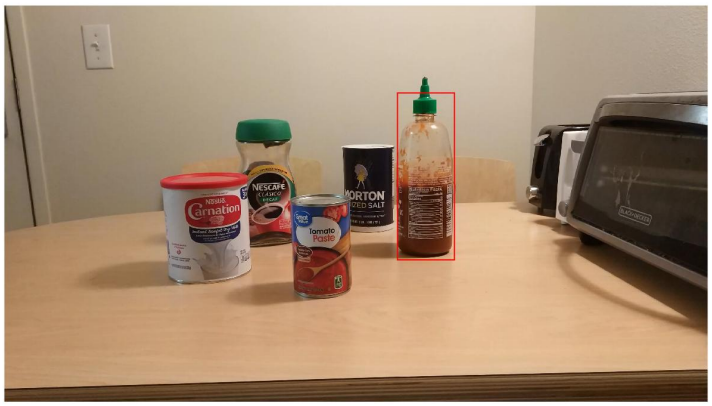


Figure 10: Synthetic focus on the template (Sriracha sauce), while blurring everything else.

3 Assessment

3.1 Deriving the Blur Kernel Width (3.0 points)

The goal is to understand how much blur is synthetically added by using a model of pinhole cameras. Consider the coordinate diagram shown in Figure 11. Here, $[X_1,Z_1]$ is a scene point of an object in the template, $[X_2,Z_2]$ is a scene point of an object in the background and $C^{(i)}$ for $i = 1,\ldots,k$ are positions of the apertures of cameras at which the scene is captured. The maximum camera translation is Δ and f is the focal length of the cameras (all are assumed to be the same).

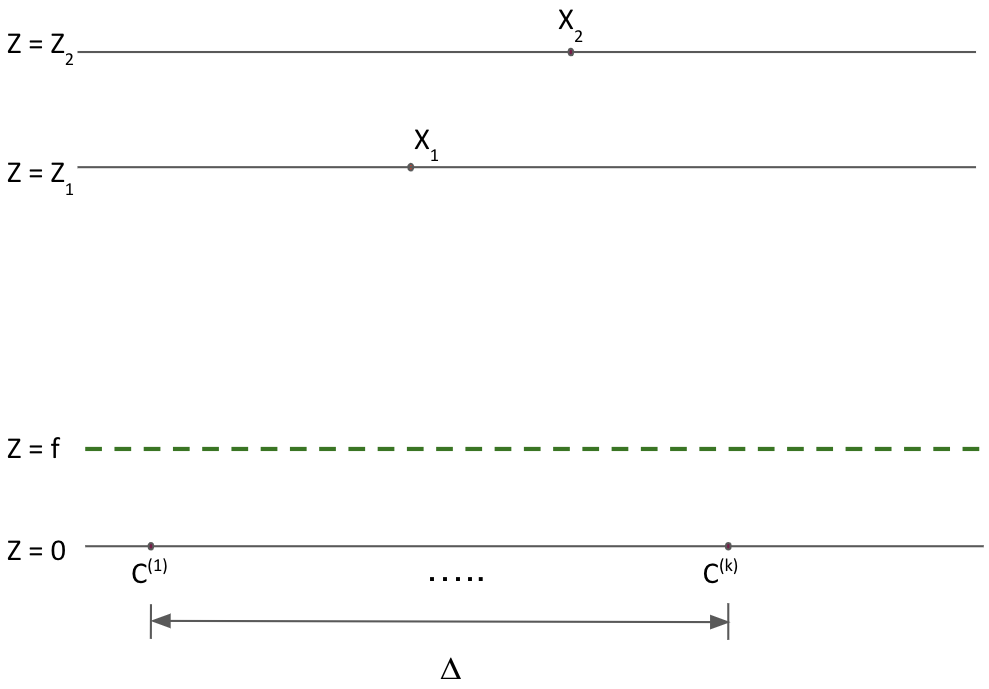


Figure 11: Example coordinate system and notation. In this figure, the dashed plane is the virtual film plane, placed one focal length *above* the apertures located at $C^{(1)},\ldots,C^{(k)}$. This is a common shorthand convention so we do not have to flip the camera images. In reality, the actual film plane would be one focal length below the aperture location. This coordinate system is used as a guide - you are welcome to modify as needed.

We will use the shift-and-add method for light field imaging such that X_1 is the point in focus (i.e. as the "template" that we shift and add"). Derive a mathematical expression for the full-width half maximum (FWHM) of the blur kernel (W) applied to X_2. Credit will be assessed both for technical correctness and the presentation of the derivation. You should not need figures, but are welcome to include them. Insert your derivation in the box below.

[Hint: Our solution to derive W was about a half page.]

[Hint: To check your solution, if $Z_1 = Z_2$ the width of the blur kernel should be zero.]

Suppose the camera is at $C^{(1)}$. By law of similar triangles, the distance $t^{(1)}$ between the projection of X_1 on virtual plane and projection of $C^{(1)}$ on virtual plane is given by:

$$t^{(1)} = \left| \frac{f(X_1 - C^{(1)})}{Z_1} \right| \tag{4}$$

Similarly, the distance $u^{(1)}$ between the projection of X_2 on virtual plane and projection of $C^{(1)}$ on virtual plane is given by:

$$u^{(1)} = \left| \frac{f(X_2 - C^{(1)})}{Z_2} \right| \tag{5}$$

The distance between the two projections $x^{(1)}$ at camera position $C^{(1)}$ is given by:

$$x^{(1)} = |t^{(1)} - u^{(1)}| = \left| \frac{f(X_1 - C^{(1)})}{Z_1} - \frac{f(X_2 - C^{(1)})}{Z_2} \right| \tag{6}$$

When the camera is at $C^{(k)}$, value of x is maximum and is given by:

$$x^{(k)} = \left| \frac{f(X_1 - C^{(k)})}{Z_1} - \frac{f(X_2 - C^{(k)})}{Z_2} \right| \tag{7}$$

Since the projections are being shifted and added, the width of the blur kernel is given by the difference between the maximum value of x and minimum value of x (at $C^{(1)}$):

$$W = x^{(k)} - x^{(1)} = \left| \left(\frac{f(X_1 - C^{(k)})}{Z_1} - \frac{f(X_2 - C^{(k)})}{Z_2} \right) - \left(\frac{f(X_1 - C^{(1)})}{Z_1} - \frac{f(X_2 - C^{(1)})}{Z_2} \right) \right| \tag{8}$$

$$= \left| f\Delta \left(\frac{1}{Z_2} - \frac{1}{Z_1} \right) \right| \tag{9}$$

3.2 Blur Kernel Shape (1.0 points)

Now that you have derived the FWHM of the blur kernel, please write the functional expression for the blur kernel. For example, is it a Gaussian blur?

$$G(x,y) = \frac{1}{k} \left| f\Delta \left(\frac{1}{Z_2} - \frac{1}{Z_1} \right) \right| * I(x,y) \tag{10}$$

3.3 Blur and Scene Depth (0.5 points)

Plot the width of the blur kernel, W, as a function of the difference in depth planes, $|Z_2 - Z_1|$. Insert your plot in the box below. Comment on the relationship between these variables.

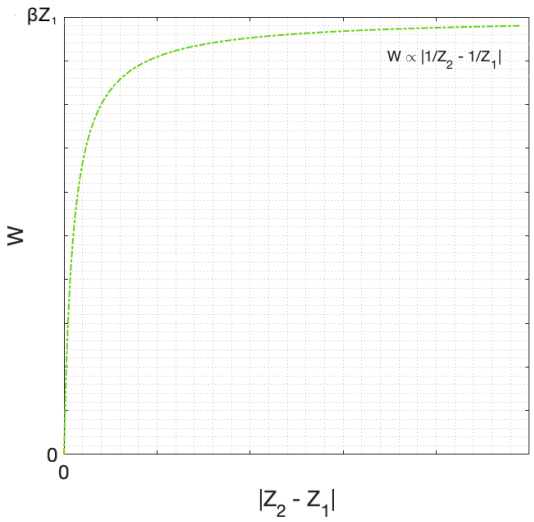


Figure 12: Plot of W vs $|Z_2 - Z_1|$. As $|Z_2 - Z_1| \to \infty$, the blur kernel width converges to βZ_1, where $\beta = f\Delta$.

Assuming f and Δ to be constant, as $|Z_2 - Z_1|$ increases, the blur kernel width W also increases, converging to $f\Delta Z_1$.

3.4 Blur and Focal Length (0.5 points)

Plot the width of the blur kernel, W, as a function of the focal length of the camera, f. Insert your plot in the box below. Comment on the relationship between these variables.

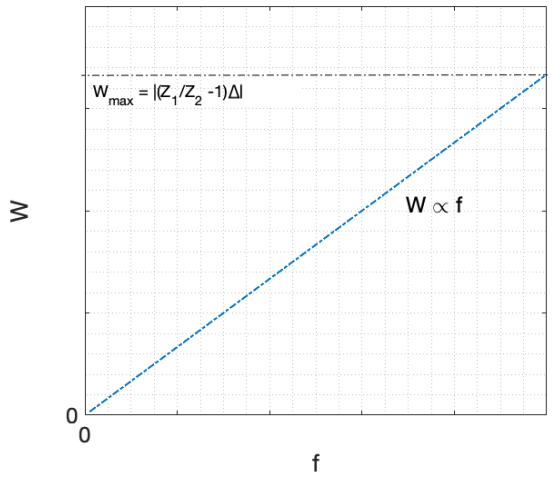


Figure 13: Plot of W vs f.

Assuming $|Z_2 - Z_1|$ and Δ and to be constant, as f increases, the blur kernel width W increases linearly. However, the maximum value of f is Z_1. Hence, as $f \to Z_1$, $W = |(\frac{Z1}{Z2} - 1)\Delta|$.

References

[1] Todor Georgeiv and Chintan Intwala. Light field camera design for integral view photography.

[2] Marc Levoy, Billy Chen, Vaibhav Vaish, Mark Horowitz, Ian McDowall, and Mark Bolas. Synthetic aperture confocal imaging. *ACM Trans. Graph.*, 23(3), August 2004.

[3] J. P. Lewis. Fast normalized cross-correlation, 1995.

[4] Andrew Lumsdaine and Todor Georgiev. The focused plenoptic camera. In *In Proc. IEEE ICCP*, pages 1–8, 2009.