Image Processing Algorithms for a Tiled Multi-projection Screen

Image Processing Algorithms for a Tiled Multi-projection Screen

By

DAWEI GUO, B.Eng.

A THESIS

SUBMITTED TO THE DEPARTMENT OF ELECTRICAL & COMPUTER ENGINEERING

AND THE SCHOOL OF GRADUATE STUDIES

OF MCMASTER UNIVERSITY

IN PARTIAL FULFILMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

MASTER OF APPLIED SCIENCE

Master of Applied Science (2020)

McMaster University (Electrical & Computer Engineering)

Hamilton, Ontario, Canada

TITLE:Image Processing Algorithms for a Tiled Multi-projection Screen

AUTHOR:Dawei Guo

B.Eng.(Measuring and Control Technology with High Precision Instrument)

Tianjin University, TianJin, China

SUPERVISOR:Dr. Jun Chen

NUMBER OF PAGES: viii, 56

Abstract

Nowadays the development of the screen technology is really fast, there are lcd, led, oled screen and many kinds of screens. They all have their own advantages and disadvantages, LCD screen is usually constrained by the size, and the LED screen is usually constrained by the resolution. In this thesis I will introduce a tiled projection screen which combined LED and LCD together. My major work is to develop algorithms which are used to solve three major problems. The first problem is the radial distortion caused by the lens. This problem is different from the usual distortion correction problem, the method used in this thesis is a reverse process of the camera calibration and the key is to simulate the distortion formula. The second problem is the complex brightness condition of the tiled projected image. In the thesis a non-linear edge blending method is applied so the projected images could merge together seamlessly. The third problem is combination of the LED and the LCD without harming the resolution of the whole picture. The result shows that the size and the resolution have been improved greatly.

Acknowledgements

First of all, I would like to express my gratitude to my supervisor Dr. Jun Chen. I am really grateful for his kind advise and help during these two years. With his help, I broadened my horizons and increased my insight. He is more than a teacher or supervisor to me, His kindness and erudition influenced me deeply and I will always respect him and be greatly honored that I'm his student.

Secondly, I want to thank Dr. Adrain Kitai and Chris. During the coop term, they helped me to get participated in a fresh new project and taught me patiently, I learned a lot from them and this experience is valuable to me.

Thirdly, I want to thank my mom Huahua Zhang heartily. She always has a crystal clear mind and she always give me the best advise when I feel depressed or get lost about the future. She is and always will be the most important person in my life, love her forever.

Contents

List of Figures

Chapter 1

Introduction and problem statement

1.1 Introduction

In our daily life we can see all kinds of screens everywhere. The electronic menu boards in Tim Hortons, the laptop screen, the TV screen, the billboards on the street, etc. The screens play a very important role in our life, without them our life could be very inconvenient. When we talk about the quality of the image shown on the screen, it's usually about resolution, brightness, color and size. These screens could basically be divided into two categories, the LED screen or the LCD screen. LED is usually constrained by the resolution, LCD screen is usually constrained by the size. Because of these two problems, I applied a new screen technology which combined LCD screen and LED screen together [5]. The schematic diagram of the screen is shown in Figure 1.1. We could use as many LCD panels as we want, so the size of this screen theoretically could be infinitely large.

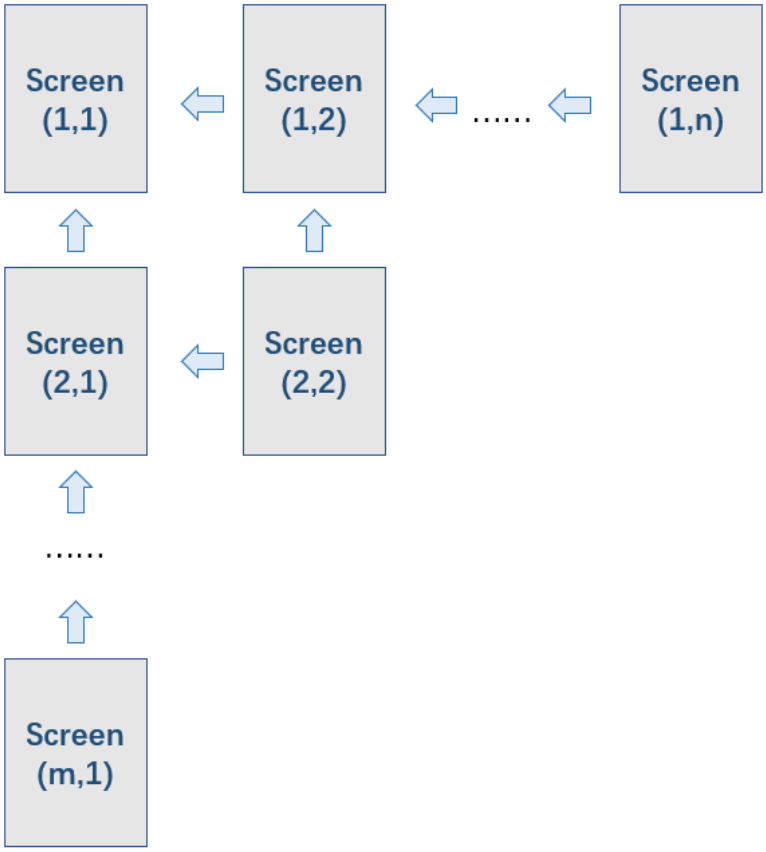


Figure 1.1: Schematic Diagram Of The Screen

There have been a lot of articles about the tilled projection technology [13] [16] [20].

1.1.1 First Problem

To adapt to this technology, there are several obstacles need to overcome. The first problem is the radial distortion of the projected image [9] [11] [21]. The flow chart of the processing procedures is shown in Figure 1.2.

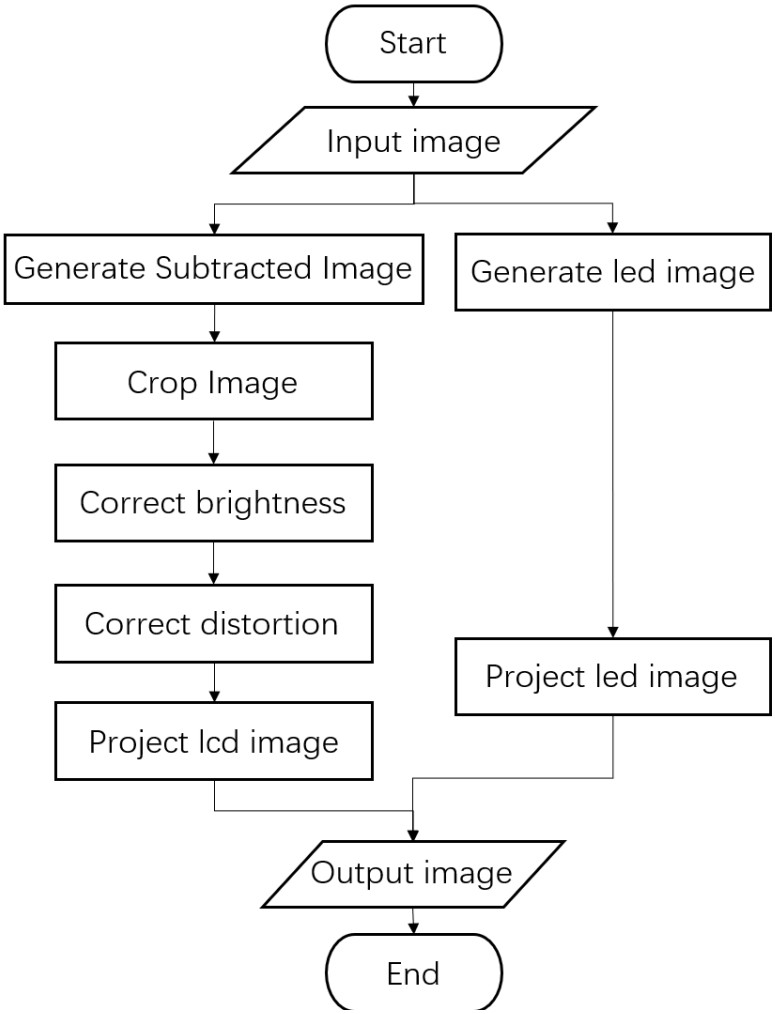


Figure 1.2: Flow Chart

The image on the LCD screen will go through the crystal lens then the image is projected on the screen. The light go through the lens will be refracted. In the center of the lens, the light will keep straight, but on the edges of the lens, the light will be refracted greatly; therefore, the lens will cause distortion to the images, mostly radial distortion and slight tangential distortion. There are two kinds of radial distortion, the first one is pincusion distorton, the second one is barrel distortion. For tangential distortion usually happens when the image plane

and the lens are not parallel. Tangential distortion is very common for camera. In this case I would only consider the radial distortion because the influence of the tangential distortion is negligible. For this lens it cause pincusion distortion to the images.

For pincushion distortion, image magnification increases with the distance from the optical axis. The result is that lines that do not go through the centre of the image are bowed inwards, towards the centre of the image just like a pincushion, and the image points will move outwards the center of the image. In Figure 1.3 we could see how the lens cause the distortion. The radial distortion usually happened when the light is blocked before or after it go through the lens. We could see from Figure 1.3 that when the aperture is placed ahead of the lens, the image is barrel distorted. The image is pincusion distorted when the aperture is placed behind the lens.

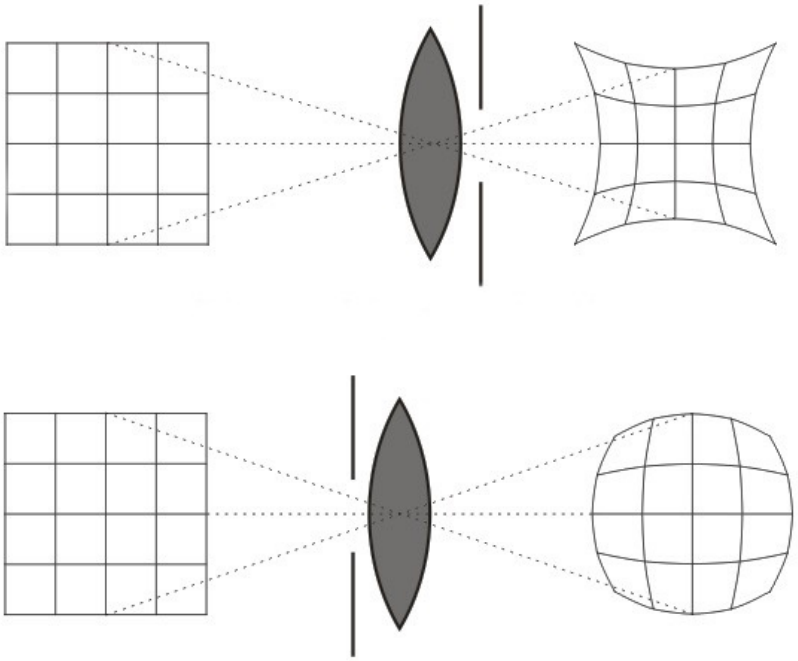


Figure 1.3: Origin Of Pincushion And Barrel Distortion

The radial distortion correction is very common and usually the most difficult part is to depict the distortion precisely [1] [3]. We need to use a mathematical way to depict the distortion. This is the simplified pincusion distortion correction model:

$$r_u = r_d * (1 + k_1 * r_d^2 + k_2 * r_d^4 + k_3 * r_d^6)$$

Where r_u and r_d are the distance from the center in the undistorted and distorted images respectively. k_1, k_2, k_3 are distortion parameters which vary from lens to lens. Usually for barrel distortion the value of k_1 would be positive, for pincusion distortion, the value of k_1 would be negative. Because the effect of k_3 to the final result is too small, I will only take k_1 and k_2 into account in the real calculation.

Figure 1.4 is the projected image without correction, I will use the checkerboards as the test image so that the level of the distortion at different points could be observed more clearly and intuitively.

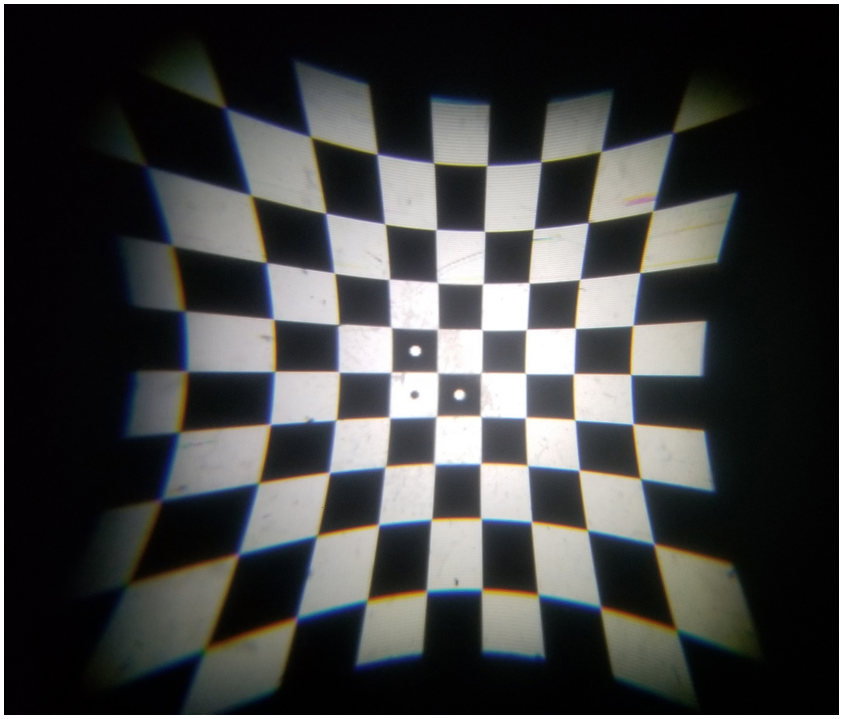


Figure 1.4: Single Projected Image Without Correction

1.1.2 Second Problem

The second problem is the complex brightness situation [6]. After the first step, the images are projected on the screen. The good side of this method is that it makes it possible to make an extendable and theoretically infinitely large screen. The bad side is that after the projection, the original image would be amplified, the edges of the projected images would overlap with each other. In Figure 1.5 you could see the overlapped region before correction. It it very obvious that the boundary of the projected image intrude into the other projected image, the whole

image on the screen is completely disordered. Because the overlap region, the brightness will change according to the number of layers which means the color of the whole image has been changed [12].

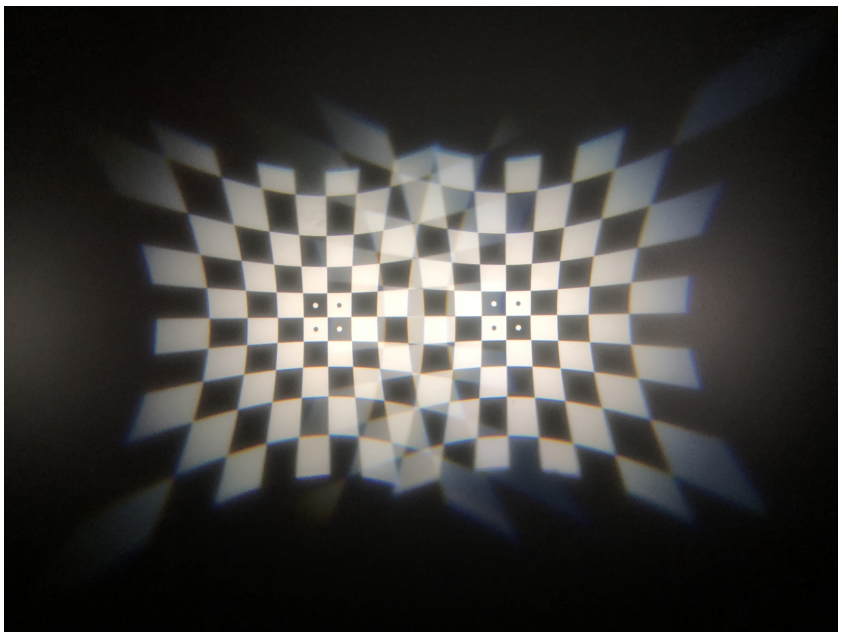


Figure 1.5: Projected Image Without Correction

1.1.3 Third Problem

The third problem is that we want to use the LCD screen and the LED screen at the same time. The LED will give out light by themselves, then the light get projected on the screen directly. The LED screen has many advantages, for example, the LED screen provides brighter display with less heat and higher contrast; the color of the LED screen is more accurate compared with LCD screen. The problem is that for a single LCD screen, the resolution of the LED screen is much lower than the LCD screen. Because of that the projected image by LED lights on the screen is pretty blurry, so the problem is how to combine the LCD and the LED together

properly to make a normal and detailed image without harming the resolution of the whole picture.

Chapter 2

Correction Of The Radial Distortion

2.1 Crop Image

For the projected image, because the image will be amplified and there would be overlapped region, so we need to make sure the overlapped region has the same content, otherwise the image would be disordered, in this way we could enhance the brightness and resolution of the picture. For adjacent two LCD screens, the images shown on them have to have repetitive content, otherwise the projected images would be two unique images without anything in common, the overlapped region would be a mess. So how should we crop the images for each LCD screen is the key point. To solve this problem I divide the image shown on the LCD screen into two parts and we could see the schematic diagram in Figure 2.1, the unique part and the repetitive part.

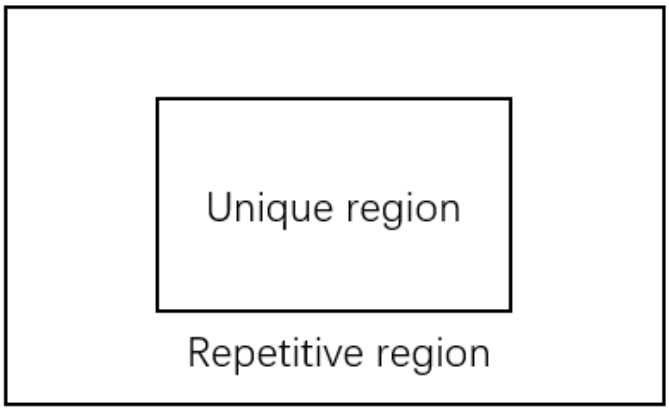


Figure 2.1: A Single Aperture

As for the size of the unique part and the repetitive part, after the measurement and tests, the amplification value is 2.2 and the aspect ration of the image is 1920:1200, all we need to know is the distance between LCD so that we could start to crop the image. the aspect ration of the LCD image has to be identical to the aspect ration of the adjacent modules, otherwise the projected images from different modules could not combine with each other seamlessly. If we want the images combine with each other with no overlapping, the length and width of the projected image on the screen should be identical to the distance between two LCD for different directions. Then we divide the two distance values with amplification factor, so we can get the actual size of the little image for a single module. In this process, all the distance and the image size are all measured with pixels.

Figure 2.2 and Figure 2.3 are the projected images before and after the crop respectively. We could see the image looks a little bit better and more like a normal image, but the overlapping is still pretty obvious. In this two images the

distortion correction has already been applied but the brightness correction has not been applied.

Figure 2.2: Projected Image One

Figure 2.3: Projected Image Two

In this way, I make full use of the available region in the LCD screen. The resolution for the LCD screen is 1920×1200 pixels, and the total resolution of the unique parts for all the apertures is 1068×739 pixels. The whole screen is made up with 160 modules, 10 rows and 16 columns; therefore, the maximum resolution is 9483×9319 pixels which is almost 10k.

2.2 Correction Algorithm

For most cases, when we take a picture, the image is distorted because of the lens, then we want to correct the distortion based on the distorted image. This is called

camera calibration. This kind of method is widely used and it has been talked a lot [8] [19] [4] [14]. But for this project we do not have a digital version of the distorted image. All we can see is the distorted image shown on the screen. Our purpose is not to get a digital corrected image, our purpose is to get the projected image looks good and normal. So the key point is to eliminate the pincusion distortion caused by the lens, the method I use is firstly generate a barrel distorted image. Secondly we project the barrel distorted image onto the screen and the pincusion distortion caused by the lens will be counteracted. Consequently distortion of the projected images would be corrected. It is an inverse process compared with the typical camera calibration. We could depict the distortion of the image in a mathematical way

$$r_d = r_u * (1 + k_1 * r_d^2 + k_2 * r_d^4) \tag{2.1.1}$$

as we can see in the formula 2.2 the amount of distortion caused by the lens is controlled by three parameters, k_1, k_2 and r_u, k_1 and k_2 are parameters determined by the lens itself, usually when k_1 is positive it will cause pincusion distortion, when k_1 is negative it will cause barrel distortion. r_u and r_d are the distance from the center in the undistorted and distorted images respectively. For example, when k_1 and k_2 are positive, for a random point in the undistorted image, $1+k_1*r_d^2+k_2*r_d^4$ would be larger than 1; therefore, r_d would be larger than r_u. That means all the points whose distance to the center is r_u in the undistorted image, adter the distortion these points would be moved outward the center and the new distance to the center would be r_d. In the function above the variable r_d and r_u are matrix, whose size is same with the size of the image, the value of each point is the distance of that point to the center point of the matrix.

The method I use is to pre-process the image before the projection, so I impose barrel distortion on the image ahead of time to counteract the pincusion distortion [17] [2]. But there will be a new problem which is how to choose the proper barrel and pincusion distortion function so that the distortion could be offset. In Figure 2.4 there are three images, the normal one, pincusion distorted one and the barrel distorted one. From this figure we could understand the relationship between these three kinds of images.

Figure 2.4: Different types of Radial Distortion

First of all, according to the parameters and the test data of the lens I got we could get the estimation of the pincusion distortion function, k_1 is set to 1.1 and k_2 is set to 0.7, the values of k_1 and k_2 will vary slightly from lens to lens. In the calibration process, parameters k_1 and k_2 for each lens will be stored and tested.

$$r_d = r_u * (1 + 1.1 * r^2 + 0.7 * r^4) \tag{2.1.2}$$

Secondly, for the barrel distortion, to find the correct distortion function more precisely and to process the data more efficiently, I use the existing Image Processing Toolbox to perform the 2-D geometric transformation.

```
[xi,yi] = meshgrid(1:w,1:l);
xt = (xi - w/2)/w;
yt = (yi - l/2)/l;
[theta,r] = cart2pol(xt,yt);
rd = ru.*(1 + 1.1*(ru).^2 + 0.7*(ru).^4);
[ut,vt] = pol2cart(theta,rd);
ui = ut*500 + 250;
vi = vt*500 + 250;
ifcn = @(c) [ui(:) vi(:)];
tform = geometricTransform2d(ifcn);
I2 = round(imwarp(I,tform,'FillValues',fill));
```

We could see from the code above. w and l are width and length of the input image. First of all we transform the pixel coordinates to the camera coordinates. Then we transform Cartesian coordinates to the polar coordinates. After this step we impose the pincusion distortion function to the matrix ru. After this step we transform polar coordinates back to Cartesian coordinates. Then we transform the camera coordinates to the pixel coordinates. For the next two steps we generate the transformation object tform which stores an inverse point-wise mapping function. ifcn contains the information of the pincusion distortion. The last step we use the transformation object to impose the inverse point-wise mapping to the normal input image A and get the output image B. Image B is barrel distorted. The reason why it is called inverse mapping is because the output image B is not the distorted image based on the distortion function $r_d = r_u * (1 + 1.1 * r^2 + 0.7 * r^4)$. If you impose the distortion function on the output image B, then the result would

be image A. Because the lens will impose the pincusion distortion to the input image; therefore, if we project image B on the screen the result will be the normal image A. In Figure 2.5 and 2.6 we can see the generated image B and the image B projected on the screen respectively.

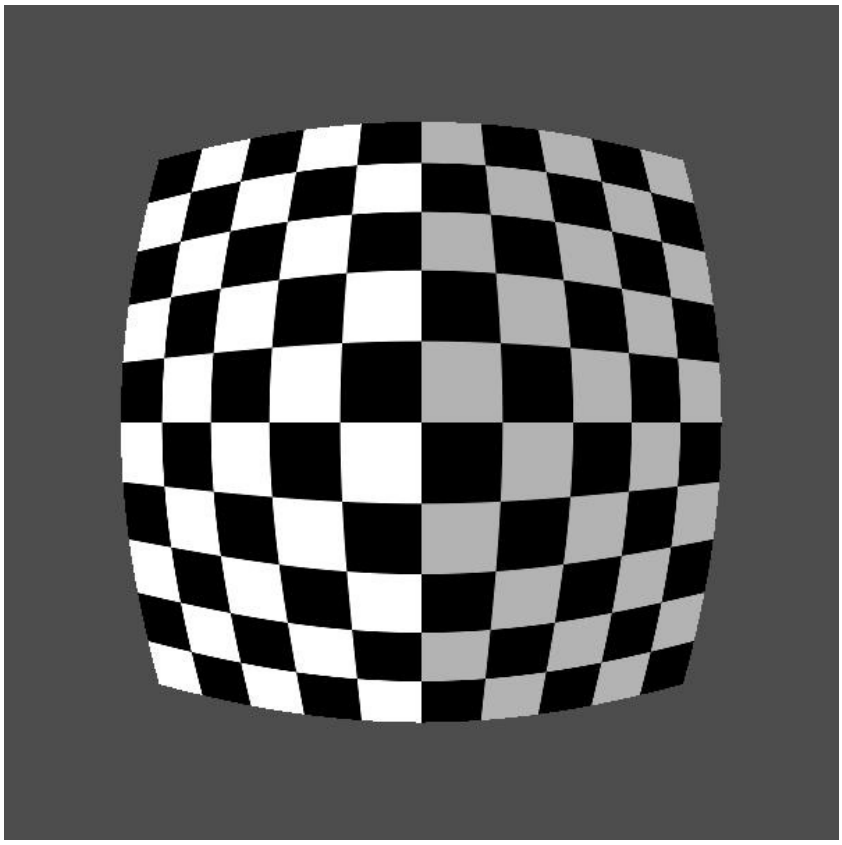


Figure 2.5: Checkerboard With Barrel Distortion

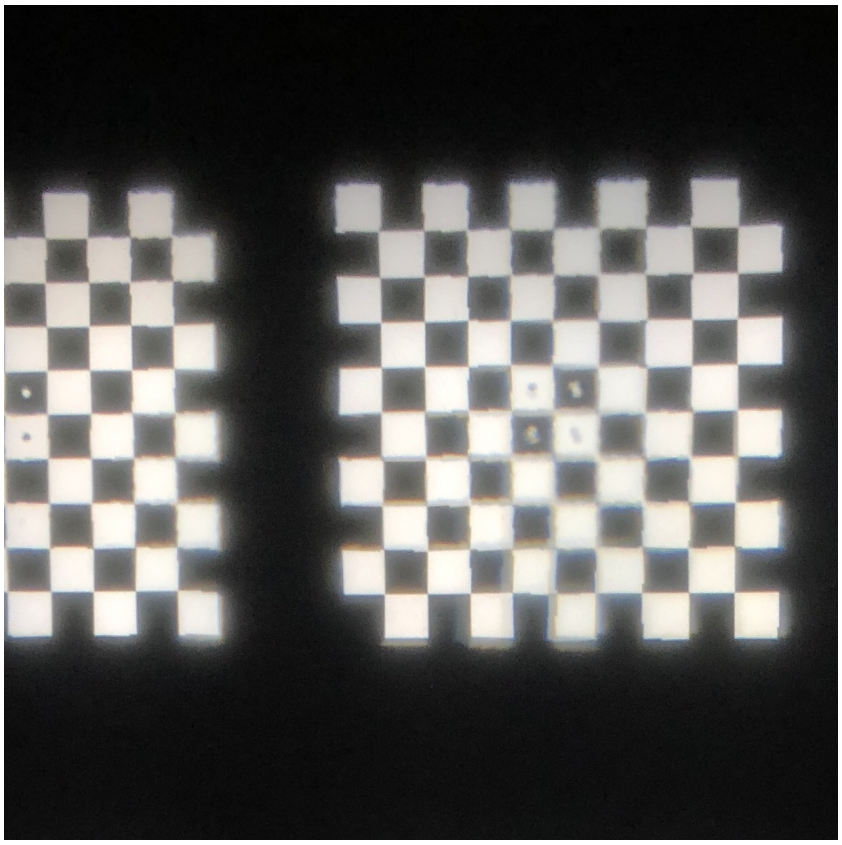


Figure 2.6: Projection Of The Distorted Image

2.3 Improve The Processing Speed

For image processing the speed is always crucial, especially when we want to process the real-time video. There have been a lot ways to solve the problem like using GPU, improving the processing algorithms, or just using a better hardware [10] [7] [18]. For this problem theoretically as long as I pre-process all the images in the apertures, the whole projected images would be integrated into a new image. The problem is that the processing speed is still too slow, it takes around 2 seconds to process 20 little images for a single module. So without considering the processing

speed of the hardware, I improved the algorithm based on the existed one.

Algorithm 1 Transofrm Image

```
1: function TRANSFORM(cinf2, Image)
2:     s ← size(Image)
3:     j ← s(1)
4:     i ← s(2)
5:     result ← Image
6:     while j > 0 do
7:         while i > 0 do
8:             j = j − 1
9:             i = i − 1
10:            if cinf2(j, i, 1) ≠ 0 and cinf2(j, i, 2) ≠ 0 then
11:                result(j, i, :) ← Image(cinf2(j, i, 1), cinf2(j, i, 2), :)
12:            end if
13:        end while
14:    end while
15:    return result
16: end function
```

After analyzing the running time of the algorithm, I find out the most time consuming part is the function *imwarp*; therefore, instead of using the function *imwarp* repeatedly, I replace the image A with a generated 3-D matrix cinf which contains the coordinate information of each pixel in the image, for example, if the size of the image A is 7*7*3, then the corresponding cinf is shown below. The size of cinf is $7 * 7 * 2$, the first layer stores the information of the x-coordinates, the

second layer stores information of the y-coordinates.

$$\begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 2 & 2 & 2 & 2 & 2 & 2 & 2 \\ 3 & 3 & 3 & 3 & 3 & 3 & 3 \\ 4 & 4 & 4 & 4 & 4 & 4 & 4 \\ 5 & 5 & 5 & 5 & 5 & 5 & 5 \\ 6 & 6 & 6 & 6 & 6 & 6 & 6 \\ 7 & 7 & 7 & 7 & 7 & 7 & 7 \end{pmatrix} \tag{2.1}$$

$$\begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 1 & 2 & 3 & 4 & 5 & 6 & 7 \end{pmatrix} \tag{2.2}$$

After the transformation the generated matrix cinf2 would be like this:

$$\begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 3 & 3 & 3 & 3 & 3 & 0 \\ 0 & 4 & 4 & 4 & 4 & 4 & 0 \\ 0 & 5 & 5 & 5 & 5 & 5 & 0 \\ 0 & 0 & 7 & 7 & 7 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \tag{2.3}$$

$$\begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 3 & 4 & 5 & 0 & 0 \\ 0 & 1 & 3 & 4 & 5 & 7 & 0 \\ 0 & 1 & 3 & 4 & 5 & 7 & 0 \\ 0 & 1 & 3 & 4 & 5 & 7 & 0 \\ 0 & 0 & 3 & 4 & 5 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \tag{2.4}$$

So instead of transforming the image itself, I transform the matrix which contains the coordinates information of each pixel. By using this way, the function imwarp would only need to be ran once instead of many times. Then the images need to be processed could be transformed according to the transformed coordinates matrix cinf2 without calling the function imwarp. After this step, the processing speed is around 20 times faster than it used to be which is around 0.2 seconds, but it is still too slow if we want to process the real time video.

After the step one we get the matrix cinf2 which stores the coordinates of the points which need to be moved inward the center of the image. So theoretically, I need to transform the image based on this matrix cinf2 and only move the pixels whose coordinates are shown in the matrix. But for a normal image there would be millions of pixels. For example, the number of pixels for a image whose resolution is 1920×1200 would be 2.3 million, if I move the pixels one by one using the for loop, it will be really inefficient.

In matrix cinf2, only the valid coordinates values are stored around the center of the matrix, the invalid coordinates values would be set to zero. Then store these valid coordinates values into two vectors rowtran and coltran, rowtran contains the information of the y-coordinates and coltran contains the information of the x-coordinates. At the mean time, store the coordinates of the points which has valid coordinates values in the matrix cinf2 into two vectors rownew and colnew, rownew contains the information of the y-coordinates and colnew contains the information of the x-coordinates. For example there is a matrix shown below which size is $3*3*2$, the corresponding rownew, colnew, rowtran and contran are:

$$\begin{pmatrix} 1 & 0 & 2 \\ 0 & 5 & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{2.5}$$

$$\begin{pmatrix} 2 & 0 & 3 \\ 0 & 1 & 0 \\ 0 & 0 & 6 \end{pmatrix} \tag{2.6}$$

$$\begin{pmatrix} 1 \\ 1 \\ 2 \\ 3 \end{pmatrix} \begin{pmatrix} 1 \\ 3 \\ 2 \\ 3 \end{pmatrix} \begin{pmatrix} 1 \\ 2 \\ 5 \\ 1 \end{pmatrix} \begin{pmatrix} 2 \\ 3 \\ 1 \\ 6 \end{pmatrix} \tag{2.7}$$

Then I use another function, ind = sub2ind(sz,row,col) which returns the linear indices ind corresponding to the row and column subscripts in row and col for a matrix of size sz. The sz for this case would be the size of the image which is a 3-D matrix. For row and col would be rownew, colnew, rowtran and coltran respectively. The result would be ind1 and ind2. ind1 stores the indices of the valid coordinates. ind2 stores the indices of the valid coordinates values. The Pseudo code for this part is shown below.

By using this method, instead moving the pixels one by one in an image, I could use indices ind1 and ind2 to transform the normal image into a distorted image directly which is much faster compared with the former method.

After these two steps the processing speed is improved greatly, the processing time for a single module is improved from 2.5 seconds to 0.06 seconds.

Algorithm 2 Transofrm Image

```
1: function TRANSFORM(cinf2, Image)
2:     s ← size(Image)
3:     j ← 1
4:     i ← 1
5:     temp ← 1
6:     rownew ← []
7:     colnew ← []
8:     rowtrans ← []
9:     rowtrans ← []
10:    result ← Image
11:    while j < s(1) do
12:        while i < s(2) do
13:            j = j + 1
14:            i = i + 1
15:            if cinf2(j, i, 1) ≠ 0 and cinf2(j, i, 2) ≠ 0 then
16:                rownew(temp) ← j
17:                colnew(temp) ← i
18:                rowtrans(temp) ← cinf2(j, i, 1)
19:                coltrans(temp) ← cinf2(j, i, 2)
20:                temp ← temp + 1
21:            end if
22:        end while
23:    end while
24:    indexa ← ones(1, length(col_tran))
25:    indexb ← 2 ∗ indexa
26:    indexc ← 3 ∗ indexa
27:    index ← [indexa, indexb, indexc]
28:    rownew ← [rownew, rownew, rownew]
29:    colnew ← [colnew, colnew, colnew]
30:    rowtrans ← [rowtrans, rowtrans, rowtrans]
31:    rownew ← [coltrans, coltrans, coltrans]
32:    ind1 ← sub2ind([s(1), s(2), 3], rownew, colnew, index)
33:    ind2 ← sub2ind([s(1), s(2), 3], rowtran, coltran, index)
34:    result(ind1) ← Image(ind2)
35:    return result
36: end function
```

Chapter 3

Brightness Correction

3.1 Origins Of The Brightness Problems

After the correction of the distortion, the overlap condition is shown in this Figure 3.1. The number in the figure represent how many layers of the images are overlapped together. As we can see, the overlapped region is symmetrical. The minimum is 2 layers and the maximum is 6 layers.

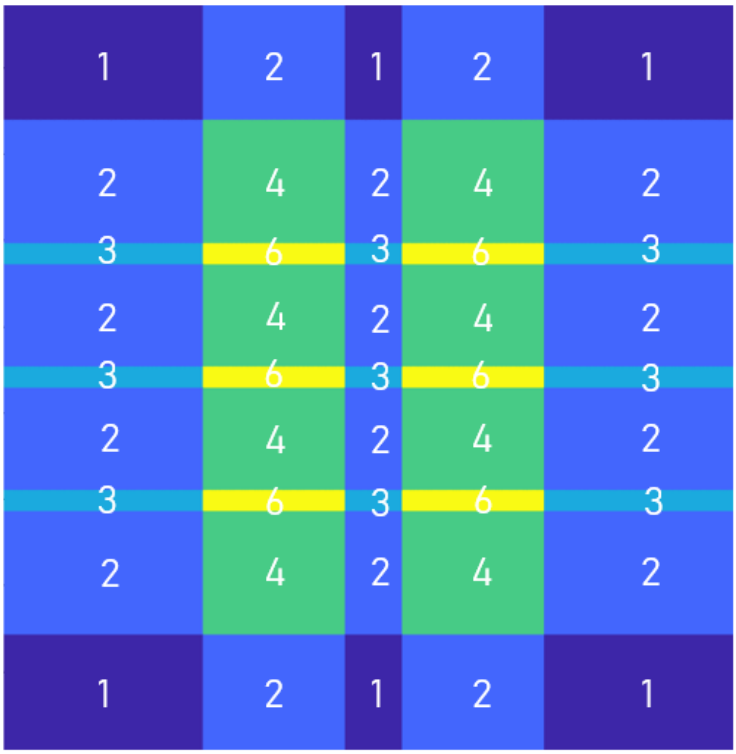


Figure 3.1: Overlap Condition

The advantages of the overlapping region are, firstly, it improves the brightness of the image; secondly it hide the edges of two adjacent images so that these images could combine with each other seamlessly. The disadvantages are, firstly it make the brightness condition for the whole image more complex; secondly the resolution might be impaired if the content in the overlapped region from different small images is not completely identical. In this section the first problem will be addressed.

For each of the lenses, because of the refraction, not only the shape of the image go through the lens is distorted, the brightness is changed as well. The formula

below is the estimation of the brightness condition of the image projected from a single lens. For this brightness decay, it has nothing to do with the pixel values, it is caused by the lens and it is only determined by the distance r and the lens itself.

$$I = a * \cos^b(\theta) \tag{2.3.1}$$

$$\theta = \arctan(\frac{r}{dis * c})$$

r is the distance between the center of the image and any point in the image. dis is the distance between the lens and the screen. a and b are coefficients control the brightness level of the lens. c is number of pixels per millimeter for the LCD screen.

We could observe the brightness distribution for a single projected image from the simulation figure.

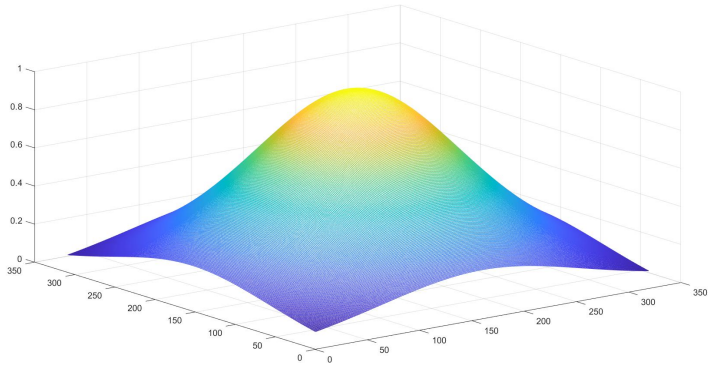


FIGURE 3.2: Simulation One Of Brightness Distribution

Figure 3.3: Simulation Two Of Brightness Distribution

So there would be two reasons which cause the brightness problem, the first one is the complex overlap condition, the second one is the decay of the brightness caused by the lens.

To present and analyze the problem more clearly and precisely, I simulate the brightness condition using matlab and the result is shown in Figure 3.4. In this figure it includes all the overlapped regions and conditions and it is the real situation. These images are not real images, for each one it is the brightness distribution map, the value of each point is a decimal and it is determined according to Function 3.1. The value is the amplification factor of the brightness. For the generated simulation figure, the value at each point is the amplification factor of the brightness for that point and it is the addition of the amplification factors of different layers. We could see from Figure 3.5 that it looks pretty similar to Figure 3.1. I would call the simulation map as BDmap which is the brightness distribution map.

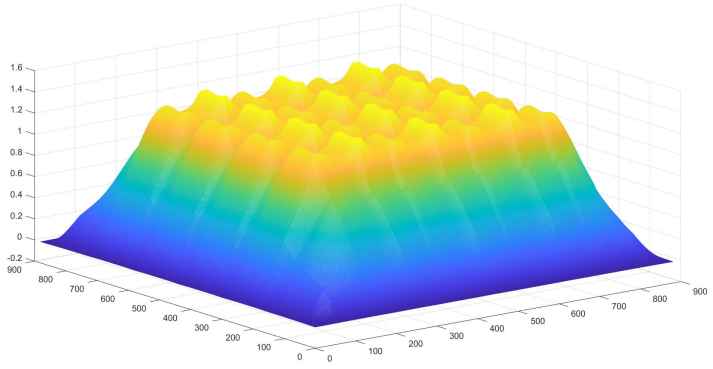


Figure 3.4: Simulation One Of Brightness Distribution For Whole Module

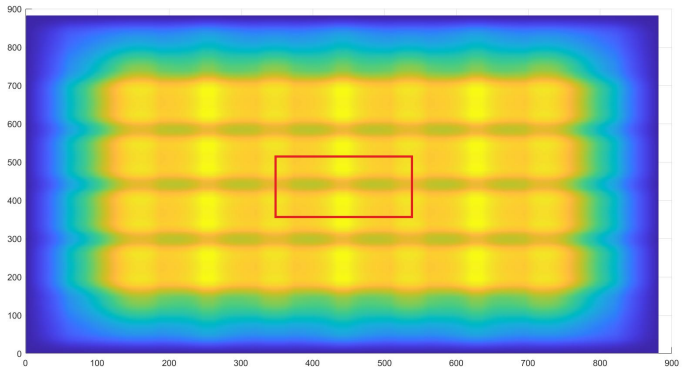


Figure 3.5: Simulation Two Of Brightness Distribution For Whole Module

3.2 Apply Normalization

The brightness of the overlapped region in an image is linear to the number of the layers to some degree if we do not consider the effect of the pixel value at that point; therefore, the edges of the projected images are very obvious. Because for different region the layers would be different. It might change from 4 to 2 directly,

that means over the edge of the region, the brightness would drop 50 percent. To solve the problem we want the projected images could merge with each other, we want the whole image is at the same brightness level and looks more smooth. To realize this goal we need to adjust the brightness of each projected image and flatten the bumppy brightness distribution shown in the Figure 3.4. The ideal situation is that for all the points in the BDmap, the amplification factors should be the same.

I crop part of the BDmap, the cropped region is in the red box which is shown in Figure 3.5. Then I choose the minimum value in this region as the standard and then using this value to normalize the cropped region. As a result I get a new map BCmap which mean brightness correction map. Therefore, in BCmap, the maximum value would be 1 so we do not need to worry that the pixel value would exceed 255 after the correction.

$$BCmap = \frac{minimumvalue}{croppedBDmap}$$

Eventually, after applying the correction map BCmap on the normal image, then simulate the brightness condition again we can see the problem is solved. In Figure 3.6 we can see the new simulation of the brightness distribution.

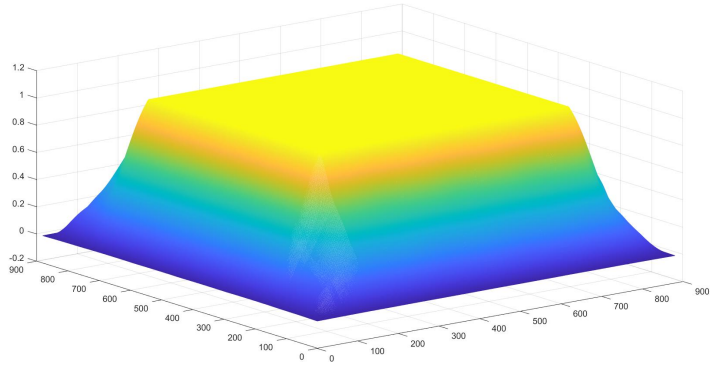


Figure 3.6: Simulation One Of Brightness Distribution After Normalization

3.3 Gamma Correction

Gamma correction is a nonlinear operation used to encode and decode luminance values in video or still image systems and Gamma correction could be defined by the following function [15].

$$V_{out} = AV_{in}^{\lambda}$$

Where V_{in} and V_{out} are the input and output matrix, usually when constant $A = 1$, the range of V_{in} and V_{out} would be zero to one, all the values in marix V_{in} would be raised to the power of γ. For real image, V_{in} would be the normalized image and $A = 255$, V_{out} would be the normal RGB image after gamma correction.

When $\gamma > 1$, γ would be called decoding gamma and the process of the gamma correction would be called gamma expansion. On the contrary when $\gamma < 1$, γ would be called encoding gamma and the process of the gamma correction would be called gamma compression.

Gamma encoding of images is very commom and widely used to optimize the usage of bits, In this way it take advantage of the non-linear manner in which humans perceive different color. The human eye could receive light, under common brightness conditions, follows an power function, human eyes have greater sensitivity to relative differences between dimmer colors than between brighter colors. If image human eyes receive is not gamma-encoded, for highlights part it allocate too many bits that humans cannot tell the difference, and too few bits to dimmer parts that humans are sensitive to.

The images shown on the LCD screen are decoded images after gamma expansion, so the brightness human eyes could receive is not linear to the pixel value. The problem is, the brightness of the overlapped region is linear to the number of the layers, if I impose the brightness correction map BCmap directly on the images, BCmap would also be decoded; Therefore, I need to achieve the gamma encoding to the brightness correction matrix BCmap ahead of time if I want to set the brightness of whole projected image at the same level. This is the function of the brightness correction. Figure 3.7 shows the relationship between the pixel value and the brightness after the images are decoded.

$$V_{out} = A * (BCmap^{1/\lambda} \odot V_{in})^{\lambda}$$

$$V_{out} = A * BCmap \odot V_{in}^{\lambda}$$

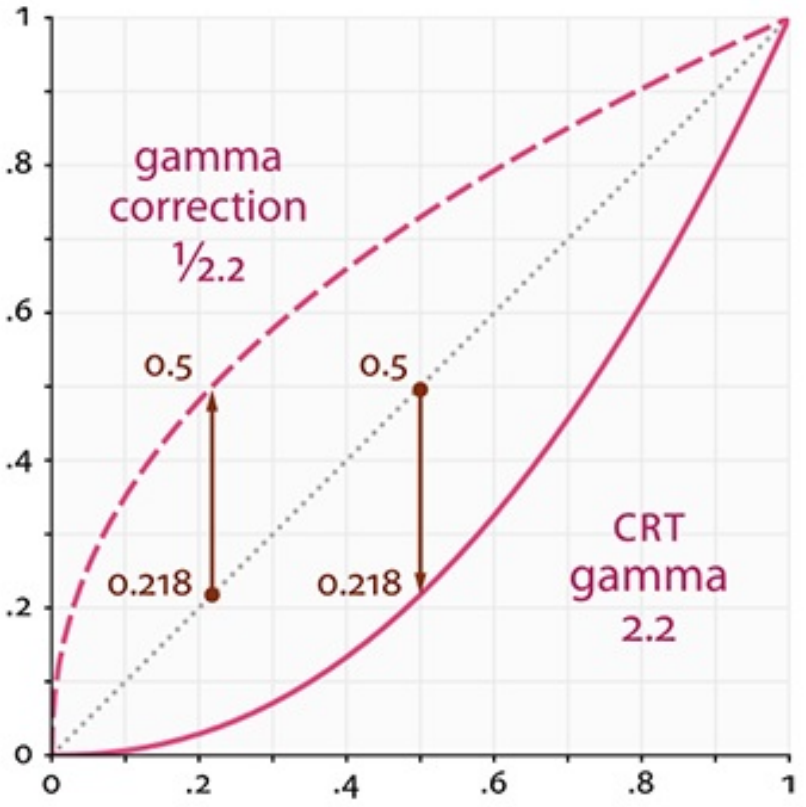


Figure 3.7: Gamma Correction Curve

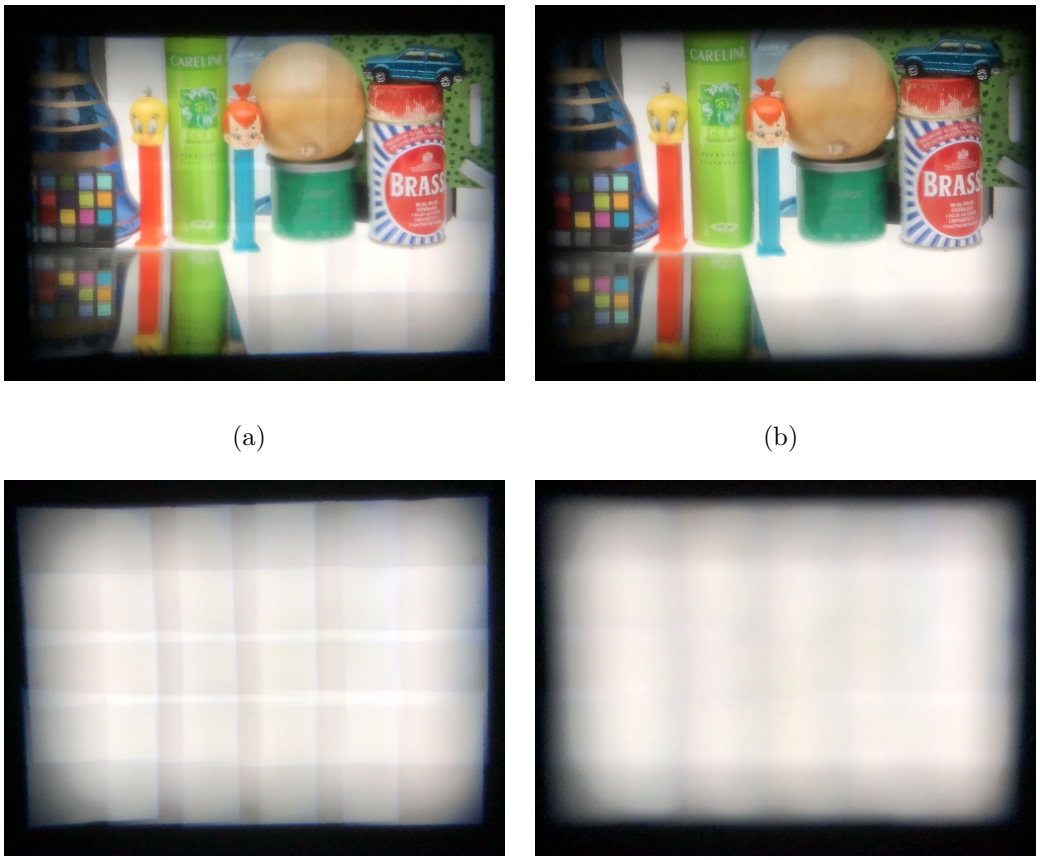
In this function we could see the result V_{out} is the real brightness that human eyes could receive range from 0 to 255. A is a constant set to 255, V_{in} is the normalized input image range from 0 to 1. V_{out} is the Hadamard product of matrix $BCmap$ and matrix V_{in}^{λ}, these three matrices have same dimension and it is an element-wise operation. For instance, if the value for a random point in BCmap is 0.5 and the BCmap is not encoded, we set $\lambda = 2.2$, when I apply the BCmap to the image, the real brightness our eyes could receive for that point would be approximately $100 * 0.5^{2.2} \approx 22$ percent of the original brightness, but we want it to be 50 percent of the original brightness, that is the reason why I need to encode

the correction map BCmap ahead of time.

For the LCD screen I use, the decoding gamma is 2.2, therefore, the corresponding encoding gamma for the brightness correction map BCmap is set to be 1/2.2.

$$V_{out} = 255 * BCmap \odot V_{in}^{2.2}$$

This is the final brightness correction function. Figure 3.8 shows the images before and after the brightness correction. We could tell in the Figure 3.8(b) and Figure 3.8(d) the projected images merge together seamlessly.



(a) (b)

(c) (d)

Figure 3.8: Projected Image Before And After Brightness Correction

Here is the Pseudo code for this part:

Algorithm 3 Generate BCmap

function GENERATE BCMAP($bright_{map}, size, s$)
  $j \leftarrow 1$
  $i \leftarrow 1$
  $LCDmap \leftarrow zeros(882, 694, 3)$
  while $i < 4$ **do**
    while $j < 6$ **do**
      $LCDmap((j-1)*size(1)+1 : (j-1)*size(1)+s(1), (i-1)*size(2)+1 : (i-1)*size(2)+s(2), :) \leftarrow LCDmap((j-1)*size(1)+1 : (j-1)*size(1)+s(1), (i-1)*size(2)+1 : (i-1)*size(2)+s(2), :) + bright_{map}$
      $i = i + 1$
      $j = j + 1$
    end while
  end while
  $BCmap \leftarrow min(LCDmap(:))./LCDmap$
  $BCmap \leftarrow BCmap^{\frac{1}{2.2}}$
  return $BCmap$
end function

Chapter 4

Combination Of LED And LCD

According to the image shown In Figure 3.8 we could tell even though the brightness of the whole projected image has been set to the same level and most part of the projected image looks smooth, even though the edges of the projected image are dimmer than the center, there are still some lines noticeable, especially when the screen is viewed from the left or right side. The reason for this problem is that the theoretical simulation and the real situation are always different. For example if the lens are not perfectly located at the correct position or the lens has flaws or it is not perfectly symmetrical, the projected image would also get influenced accordingly, then the overlap condition would be different from the simulation.

To address this problem, I introduce the LED light source. The LED lights locate between the LCD screen and the projected screen. The method is to project the LED image on the same screen, so the projected LCD image and the projected LED image will combine together and form the final image. In this way the LED image will hide those lines, so the color of the whole picture would be more smooth.

For a single LED light, the projected light spot is a circle whose size is around 20 millimeters. Therefore, the projected LED image is much more blurry compared with the projected LCD image. For most cases we want a high resolution image, but for this module the blurry image is exactly what we need.

So there would be three problems. The first one is how to generate the LED image corresponding to the LCD image. The second one is in which region the LED light should be turned on and in which region the LED light should be turned off. The third one is how to address the brightness problem when the LED image and LCD image are combined together.

4.1 Pixel Value Of LED Image

For the LED light spot projected on the screen, the color of that spot is determined by the LCD image on that spot. So I need to find out the relationship between the LCD image and the LED image. The distance between two single LED is fixed, for the LCD screen the PPI is also fixed. so the pixels between two LEDs are distance/ppl. By using this strategy I could calculate the position of the LED light spot in the corresponding LCD image.

Because the color varies from pixel to pixel. Therefore, the pixel at the point where the LED locates at might not be representative enough. For example, in Figure 4.1 below we could see there is one pink petal within the white box and the background is red, the diameter of the white circle is the diameter of the LED light spot. Obviously pink is not representative cause most part of image within

the circle is red. In this case if I set the color of the LED to pink, then the red region would be whitened which is not the result we would like to see.

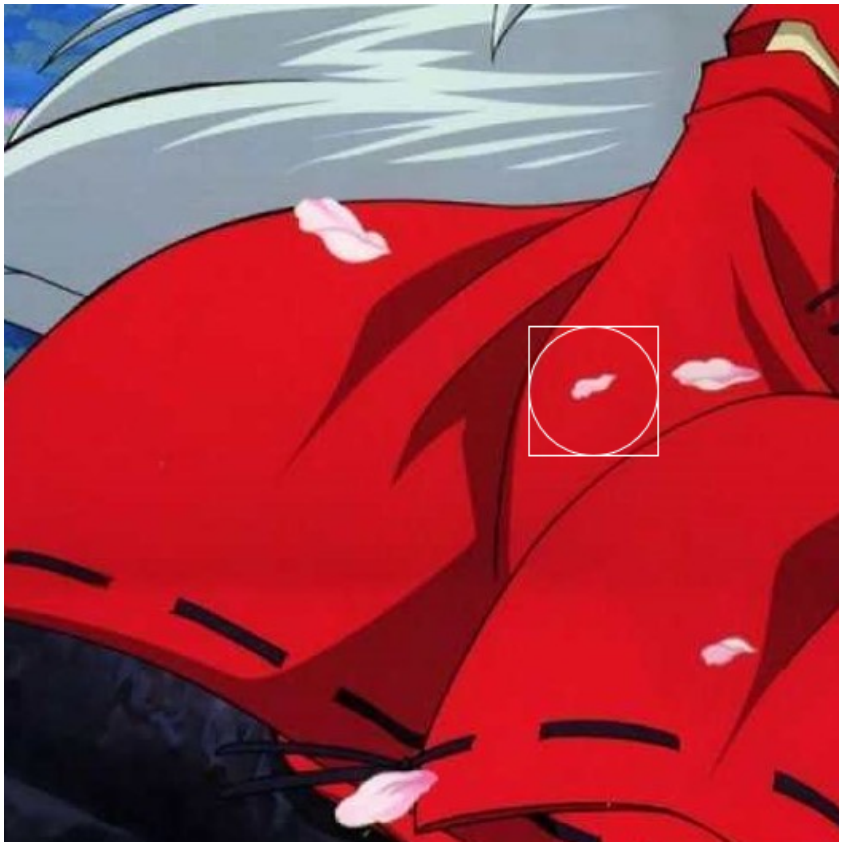


Figure 4.1: Choose The Color For LED

To solve the problem, instead of choosing one pixel, I will take all the pixels in that region into account and the size of this region is determined by the size of the LED light spot. To find out the representative color for the LED light, I choose the mean value of the color in that region. There is another problem which is that the calculation speed of the mean value for a square region is much faster than a round region. Therefore, I crop the round light spot and choose the maximum square inside it and we could see the schematic diagram in Figure **??** below. I

would call this region color box. In this way the processing speed is improved. Even through the real size of the LED light spot is larger than the color box, using the color box to determine the color for the LED is still precise, it is because the brightness of the LED light spot would decay along with the distance to the spot center. So the color box has most of the image information. This is the function to get the color of the LED.

$$R_{LED} = \frac{1}{m * n} \sum_{i=1}^{n} \sum_{j=1}^{m} r_{ij}$$

$$G_{LED} = \frac{1}{m * n} \sum_{i=1}^{n} \sum_{j=1}^{m} g_{ij}$$

$$B_{LED} = \frac{1}{m * n} \sum_{i=1}^{n} \sum_{j=1}^{m} b_{ij}$$

where m and n is the pixel size of the color box, r_{ij}, g_{ij} and b_{ij} are the RGB values for the point locates at (i, j) in the color box.

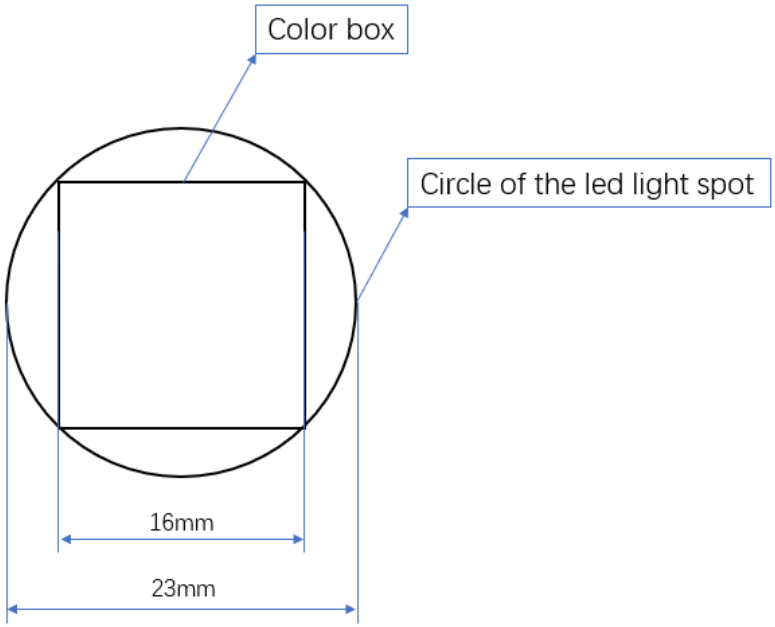


Figure 4.2: Color Box

For the LED light there is no gamma correction; therefore, the pixel value of the LED light is linear to the brightness our human eyes could receive. Because of this character, when we choose the color for the LED, we need to encode the image first, then calculate the color for the LED from the encoded color box.

For the whole module, I generate the LED image with this strategy. In Figure 4.3 and Figure 4.4 we could see the LCD and corresponding LED image.

Figure 4.3: LCD Image

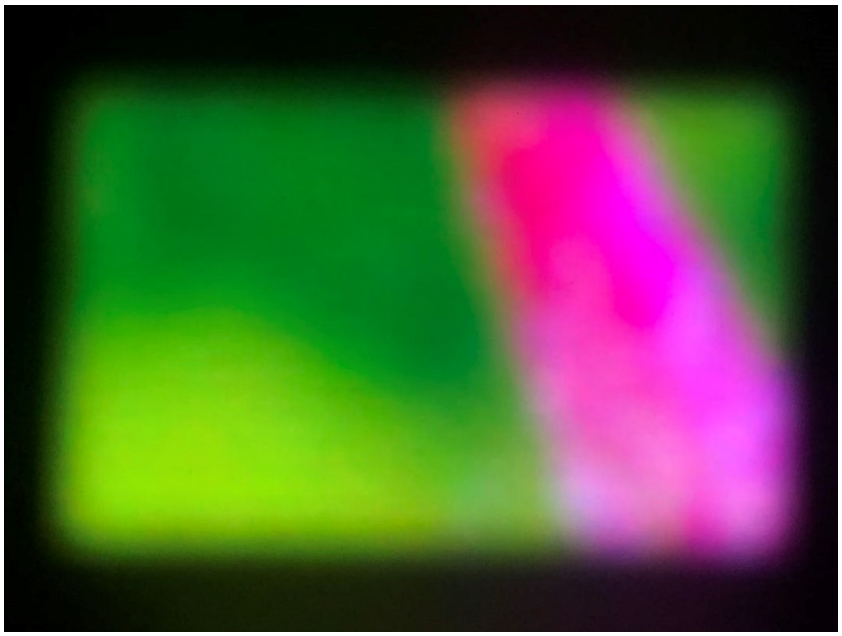
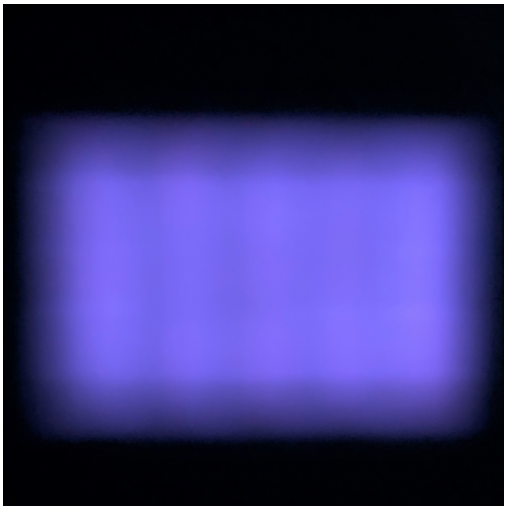


Figure 4.4: Corresponding LED Image

4.2 Determine Which LED Are Turned On

In Figure 4.5 there are three projected images without LED.

(a) (b)



(c)

Figure 4.5: Images Without LED

We could tell from these images that we barely notice the edges in the detailed image because the edges mingle with the details in the image.

But for Figure 4.5(b) it is pure white and the result is the worst. The edges are very obvious because the image has the maximum brightness; therefore, the white is more sensitive to the number of the layers. For example, In Figure 4.5(c), if the pixel values of the image are set to 1, then the maximum brightness of the projected image is just 6 even without the brightness correction and you could barely tell the edges because the whole image is too dim. But In Figure 4.5(b), the pixel values are all set to 255, once the real brightness correction is not identical to the theoretical one, the flaw of the image would be noticeable. Because of this, in the test case I usually use the white image as the test image, once the brightness of the projected white image looks even and smooth, then the brightness for other images would not be a problem. For the first case, if I apply the LED image, the whole image would be more blurry. For the second and third images, if I apply the LED image, the edges would be covered and the resolution would not change.

So from these three different situations we could tell the LED should be lightened only if the color in the color box is pure, if the image in the color box has a lot of details, the using of the LED would impair the resolution. In the mathematical way I will use the variance of the color box to measure the purity of the color. Here is the function to calculate the variance. R_{var},G_{var},B_{var} are variance for the R,G,B layer.

$$R_{var} = \frac{1}{m * n} \sum_{i=1}^{n} \sum_{j=1}^{m} (r_{ij} - R_{LED})^2$$

$$G_{var} = \frac{1}{m * n} \sum_{i=1}^{n} \sum_{j=1}^{m} (g_{ij} - G_{LED})^2$$

$$B_{var} = \frac{1}{m * n} \sum_{i=1}^{n} \sum_{j=1}^{m} (b_{ij} - B_{LED})^2$$

$$Variance = R_{var} + G_{var} + B_{var}$$

If the variance is lower than the threshold, that means the color in the color box is pure enough and the LED would be turned on, the color of the LED would be the mean value of the color box. If the variance exceed the threshold, the LED would also be turned on actually, but the color would be one third of the mean value of the color box. The advantage of turning on the LED on the detailed region is that the brightness and color of the projected image would be improved, the whole image would looks more vivid and smooth, the color would be more precise, it's more like apply the filter on the image. In Figure 4.6 we could see the generated LED image.

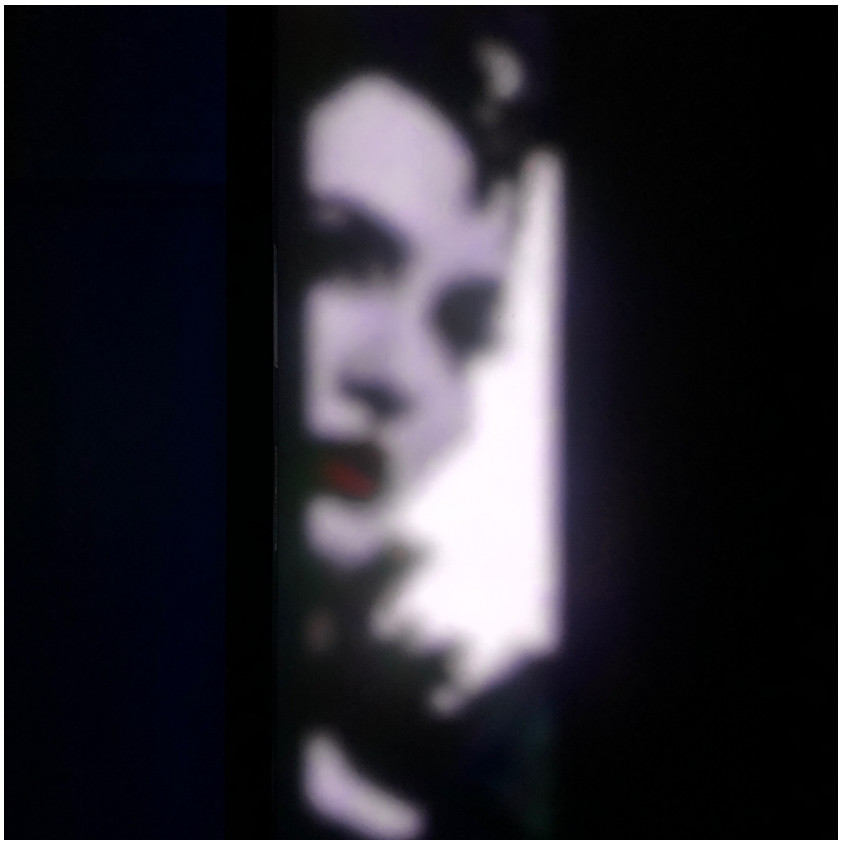


Figure 4.6: Projected LED Image

4.3 Generate Subtracted LCD Image

After the first two steps, there would be a new problem which is the brightness of the image after applying the LED light would be too high. From Figure **??**, we can see the region has a pure color is too bright compared with the rest part because there are a lot LEDs have been turned on, the brightness of that region would be too high. The brightness map of the LED would pile up. The final brightness would be the addition of the LED image and the LCD image, which is too bright compared with the detailed region.

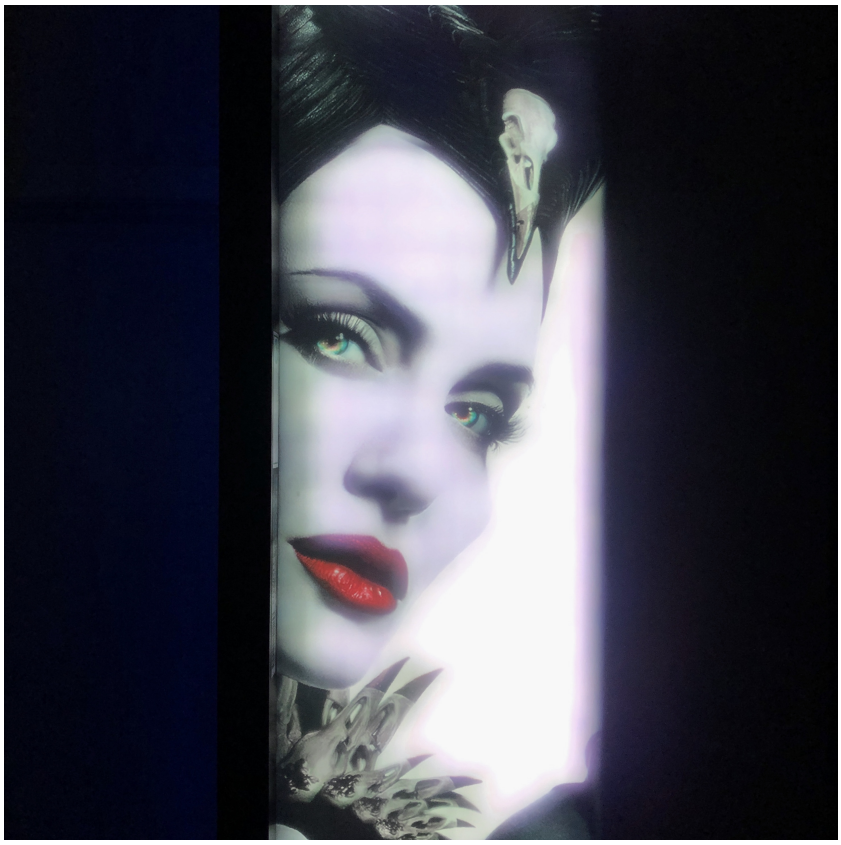


Figure 4.7: Combination Of LCD And LED Image

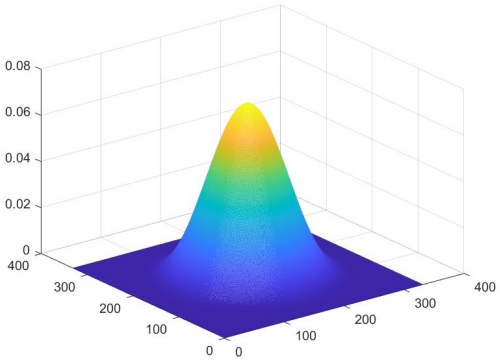
To solve this problem, my strategy is to firstly subtract the LED brightness map from the LCD brightness map. If there are many LEDs have been turned on, then the LED brightness map would be subtracted from the LCD brightness map layer by layer until the value of the LCD brightness map becomes zero. Secondly, apply the subtracted brightness map to the input image. Thirdly, instead of correcting the distortion and brightness of the original image, I will process and project the subtracted image.

In Figure 4.8 we could see the simulation of the brightness map for a single LED light, the size of the map is the size of the LED light spot. But the LED light and

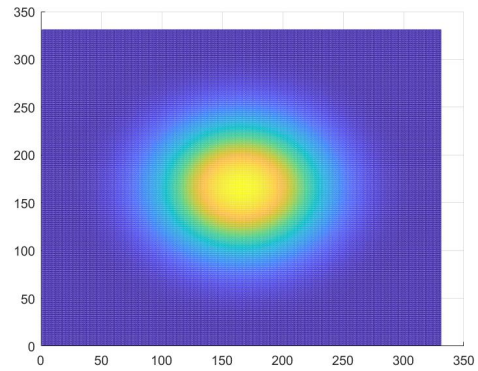
the LCD screen are different and how could we compare the brightness map from different light sources. The key point is to depict the brightness of the LED image and the LCD image with the same criteria. The brightness map of the LED and the LCD here are all normalized map, the values in the map are the amplification factors. For LED light, the simulation of the brightness could be depicted by the function.

$$I = a * \cos^b(\frac{\pi r}{2r_{max}})$$

a controls the magnitude of the brightness map for a single LED. b controls the gradient of the brightness map. r is the distance between the center of the light spot and any point is the region.



(a)



(b)

Figure 4.8: LED Brightness Map

To find out the best coefficients to depict the brightness map for the LED, I turned on a single LED and set it to white which is 255, and subtract the LED brightness map from the LCD brightness map. For the LCD brightness map all the values are set to 1, the size of the map is the size of the original input image. Then I will apply the subtracted brightness map to the original image, the subtracted region would be dimmer compared with other part. The color of the original image would also be white. Then I process the subtracted image, project it on the screen

and turn on the single LED, if the projected image looks smooth and natural, that means the coefficients we choose are correct.

Another thing need to consider is that the LED brightness map has nothing to do with the gamma correction, so before we apply the subtracted brightness map to the input image, the subtracted brightness map should be decoded ahead of time.

In Figure 4.9 we could see the subtracted LCD image.

Figure 4.9: Subtracted LCD Image

Here is the pseudo code for this part:

Algorithm 4 Generate Subtracted Image

function Generate Subtracted Image($LEDmask, Image, s, size, threshold$)
  $j \leftarrow 1$
  $i \leftarrow 1$
  $add_{mask} = zeros(s(1), s(2))$;
  $result \leftarrow double(Image)$
  $Image \leftarrow uint8(255 * (double(image)/255).^{2.2})$
  while $i < 31$ **do**
    while $j < 17$ **do**
      $y \leftarrow size(1)/8 + (j-1) * size(1)/4$
      $x \leftarrow size(2)/12 + (i-1) * size(2)/6$
      $temp \leftarrow image(y-50 : y+50, x-50 : x+50, :)$
      $deviation \leftarrow std2(temp(:,:,1)) + std2(temp(:,:,2) + std2(temp(:,:,3)$
      if $deviation < threshold$ **then**
        $add_{mask}(y-50 : y+50, x-50 : x+50) \leftarrow add_{mask}(y-50 : y+50, x-50 : x+50) + LEDmask$
      end if
      $i = i + 1$
      $j = j + 1$
    end while
  end while
  $add_{mask} \leftarrow 1 - add_{mask}$
  $add_{mask} \leftarrow add_{mask}^{\frac{1}{2.2}}$
  $result \leftarrow uint8(result \odot add_{mask})$
  return $result$
end function

Chapter 5

Conclusion

From Figure 5.1(a) we could see the projected LCD image without LED. Figure **??** is the combination of the projected LCD and LED images. From this comparison we could see the color of Figure **??** is more uniform and the brightness is more smooth. For this image it is very representative, because there a a lot of white region and there are also a lot of details reight next to the white region; therefore, the brightness of the a single LED in the white region is maximum. It is harder to combine the LED image and the subtracted image together. Once these two images could not blend together seamlessly, the edges would be very obvious.

(a) (b)

Figure 5.1: Result One

For the processing speed, I process 100 different images and take the average time as the result, the resolution of the processed image is 1920×1200, the average processing speed is 0.06321 seconds.

For the resolution of the screen, after the projection, for a single module the image is amplified so the PPI would be the PPI of the LCD screen divided by the

amplification value.

$$PPI = \frac{PPI_{LCD}}{amplificationvalue} = \frac{283}{2.2} = 129$$

In Figure 5.2, I combined four different pictures together and we could see the three problems have been solved and for this type of screen, the resolution and size have been improved greatly.

Figure 5.2: Result Two

Bibliography

[1] J. Andrade and L. J. Karam. Robust radial distortion correction based on alternate optimization. In: *2016 IEEE International Conference on Image Processing (ICIP)*. 2016, 2956–2960.

[2] D. Bailey, Y. Chang, and S. Le Moan. Lens Distortion Self-Calibration Using the Hough Transform. In: *2018 International Conference on Field-Programmable Technology (FPT)*. 2018, 385–388.

[3] F. Bukhari and M.N Dailey. Automatic Radial Distortion Estimation from a Single Image (2013), 31–45.

[4] C.Portalés et al. An interactive cameraless projector calibration method (2020), 109–121.

[5] C. Chen et al. Sequential Color LED Backlight Driving System for LCD Panels. *IEEE Transactions on Power Electronics* 22(3) (2007), 919–925.

[6] R. Chen, J.-M. Xue, and M.T. He. Seamless Multi-Projector Displays Using Nonlinear Edge Blending (2018), 764–778.

[7] R. Choy and A. Edelman. Parallel MATLAB: Doing it Right. *Proceedings of the IEEE* 93(2) (2005), 331–341.

[8] I. Garcia-Dorado and J. Cooperstock. Fully automatic multi-projector calibration with an uncalibrated camera. In: *CVPR 2011 WORKSHOPS*. 2011, 29–36.

[9] R. Hartley and S. B. Kang. Parameter-Free Radial Distortion Correction with Center of Distortion Estimation. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 29(8) (2007), 1309–1321.

[10] S. Kameyama and Y. Miura. Research for High Speed Image Processing Programming Method on Combined Environment of CUDA and OpenCV. In: *2018 IEEE International Conference on Consumer Electronics-Taiwan (ICCE-TW)*. 2018, 1–2.

[11] K. Liao et al. DR-GAN: Automatic Radial Distortion Rectification Using Conditional GAN in Real-Time. *IEEE Transactions on Circuits and Systems for Video Technology* 30(3) (2020), 725–733.

[12] A. Majumder et al. Achieving color uniformity across multi-projector displays. In: *Proceedings Visualization 2000. VIS 2000 (Cat. No.00CH37145)*. 2000, 117–124.

[13] P. Prentašić et al. A method for projector brightness calibration for tiled displays. In: *The 33rd International Convention MIPRO*. 2010, 1379–1383.

[14] Rong Jiangpeng et al. Radial Lens Distortion Correction Using Convolutional Neural Networks Trained with Synthesized Images. In: *Computer Vision – ACCV 2016*. 2017, 35–49.

[15] S.Rahman et al. An adaptive gamma correction for image enhancement (2016), 35.

[16] B. Sajadi et al. Color Seamlessness in Multi-Projector Displays Using Constrained Gamut Morphing. *IEEE Transactions on Visualization and Computer Graphics* 15(6) (2009), 1317–1326.

[17] D. Santana-Cedrés et al. Estimation of the Lens Distortion Model by Minimizing a Line Reprojection Error. *IEEE Sensors Journal* 17(9) (2017), 2848–2855.

[18] G. Sharma and J. Martin. MATLAB®: A Language for Parallel Computing (2009), 3–36.

[19] M. A. Tehrani, M. Gopi, and A. Majumder. Auto-calibration of multi-projector systems on arbitrary shapes. In: *2016 IEEE Applied Imagery Pattern Recognition Workshop (AIPR)*. 2016, 1–3.

[20] F. Teubl et al. FastFusion: A Scalable Multi-projector System. In: *2012 14th Symposium on Virtual and Augmented Reality*. 2012, 26–35.

[21] A. Wang, T. Qiu, and T.Shao. A Simple Method of Radial Distortion Correction with Centre of Distortion Estimation (2009), 165–172.