Investigating Computational Approaches and Proposing Hardware Improvement to the Vision Correcting Display

Zehao Wu

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2016-67
http://www.eecs.berkeley.edu/Pubs/TechRpts/2016/EECS-2016-67.html

May 12, 2016

Copyright © 2016, by the author(s).
All rights reserved.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission.

Acknowledgement

This research was supported in part by the National Science Foundation at the University of California, Berkeley under grant number IIS-1219241, "Individualized Inverse-Blurring and Aberration Compensated Displays for Personalized Vision Correction with Applications for Mobile Devices."

Investigating Computational Approaches and Proposing Hardware Improvement to the Vision Correcting Display

by Zehao Wu

Research Project

Submitted to the Department of Electrical Engineering and Computer Sciences, University of California at Berkeley, in partial satisfaction of the requirements for the degree of **Master of Science, Plan II**.

Approval for the Report and Comprehensive Examination:

Committee:

Professor Brian A. Barsky
Research Advisor

(Date)

* * * * * * *

Professor Carlo H. Sequin
Second Reader

(Date)

Investigating Computational Approaches and Proposing Hardware Improvement to the Vision Correcting Display

by

Zehao Wu

A thesis submitted in partial satisfaction of the

requirements for the degree of

Master of Science

in

Engineering - Electrical Engineering and Computer Sciences

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Professor Brian A. Barsky, Chair
Professor Carlo H. Sequin

Spring 2016

Investigating Computational Approaches and Proposing Hardware Improvement to the Vision Correcting Display

Copyright 2016
by
Zehao Wu

Abstract

Investigating Computational Approaches and Proposing Hardware Improvement to the Vision Correcting Display

by

Zehao Wu

Master of Science in Engineering - Electrical Engineering and Computer Sciences

University of California, Berkeley

Professor Brian A. Barsky, Chair

The purpose of the vision correcting display is to enable the viewer who has eye aberrations to see the display in sharp focus without requiring the use of eyeglasses or contact lenses. Fu-Chung Huang et al. proposed two different solutions to this problem. In the first one, they use a multilayer display [10], and in the second one they utilize light field technology to generate a sharp image from the display plane [11].

In this paper, we present three major software improvements that have been made upon the light field idea, including the forward method, optimization of the forward method and the improvement on projection and prefiltering. One major hardware improvement has been proposed, which replaces the pinhole array to a microlens array for the purpose of creating the light field. We develop a simulation program to better evaluate the result.

Contents

Contents i

List of Figures iii

List of Tables v

1 Introduction 1

2 Related Work 3
2.1 Multilayer Display . . . 3

3 Aberrations of the Eye 5
3.1 Zernike Polynomials . . . 5
3.2 Challenge of the Higher Order Aberrations . . . 6

4 Software Improvement for Light Field Displays 8
4.1 Introduction to Algorithms . . . 8
4.2 Original Algorithm using Light Field . . . 10
4.3 Forward Method . . . 12
4.4 Optimization of the Forward Method . . . 12
4.5 Improvement on Projection and Prefiltering . . . 14

5 Hardware Improvement for Light Field Displays 18
5.1 Prototype Using iPhone 6 . . . 18
5.2 Pinhole Array to Microlens Array . . . 19

6 Simulation 21
6.1 Simulation Algorithm . . . 21
6.2 Simulation Speedup . . . 22
6.3 Performance Analysis . . . 26
6.4 Limitations and Future Improvement . . . 31

7 Evaluation 32

7.1 Simulation Results 32
7.2 Visual Metrics 32
7.3 RGB Effect 33

8 Future Work **36**
8.1 One Eye to Two Eyes 36
8.2 Larger Display 36
8.3 Eye Tracking 36

9 Conclusion **37**

Bibliography **38**

List of Figures

2.1 Prototype multilayer display, front view and side view. 3

3.1 Visual aberrations can be described by the eye's point-spread function (PSF), often represented using wavefront maps. Left side is a PSF, and right side is a wavefront map. Data used in the plotting is collected from the author using a Shack-Hartmann wavefront sensor. 6

4.1 This is a figure illustrating defined symbols. (left) The input image is s_w by s_h pixels. (middle) p and t will specify a light ray, and o_s, o_d are positions where pt hits the display and the sensor respectively. (right) There are 5 pixels under each pinhole. h is the depth distance between the pinhole and the display plane. . . . 9

4.2 Evaluation of vision-correcting displays. We compare simulations both qualitatively and quantitatively using contrast and quality-mean-opinion-square (QMOS) error metrics. A conventional out-of-focus display always appears blurred (second column). Multilayer displays with prefiltering improve image sharpness but at a much lower contrast (third column). Light field displays without prefiltering require high angular resolutions, hence provide a low spatial resolution (fourth column). The proposed method combines prefiltering and light field display to optimize image contrast and sharpness (right column). The QMOS error metric is a perceptually linear metric, predicting perceived quality for a human observer. We also plot maps that illustrate the probability of an observer detecting the difference of a displayed image to the target image (bottom row). Our method performs best in most cases. (Source images courtesy of flickr users Jameziecakes, KarHan Tan, Mostaque Chowdhury, Thomas Quine (from top)) This figure is from [5]. 16

4.3 Comparison of the prefiltered images generated by two algorithms described in optimization of the forward method (left) and overall improvements (right). The two prefiltered images are generated with the same parameters. We can notice that there is the strange bright and dark pattern in the left image. The same artifact does not show in the right image. 17

5.1 Left side is the iOS App Demo. In the middle, it is the top view for the current prototype. Right side is the side view for the same prototype. 18
5.2 Light ray path going through one pinhole. 19
5.3 Light ray path going through one lenslet. 20

6.1 GPUs have thousands of cores to process parallel workloads efficiently. [15] . . 24
6.2 Compute with GPU. 25
6.3 Run with a sample time of $100 \times Area(Aperture)$. 27
6.4 Runtime of aperture sampling times using the original display. 28
6.5 Runtime of aperture sampling times using the original display. 29
6.6 Effect of OpenCL on prefiltering time. 30

7.1 Simulation result. 33
7.2 DRIM results for the letter E. 34
7.3 HDR-VDP2 results for the letter E. 35

List of Tables

6.1 Specifications of the benchmark computer. 27
6.2 Runtime for 400px original display 28
6.3 Runtime for Different Pixel Sizes. 30

Acknowledgments

This research was supported in part by the National Science Foundation at the University of California, Berkeley under grant number IIS-1219241, “Individualized Inverse-Blurring and Aberration Compensated Displays for Personalized Vision Correction with Applications for Mobile Devices.”

Chapter 1

Introduction

Vision is the primary channel through we acquire information about the surrounding world, and we rely heavily on vision for almost all the tasks that are required in daily life. This is a personal experience, and people with different visual imperfections, caused by genetic inheritance, disease, or age, will have difficulty in finishing many of the jobs.

The most common ways to solve vision problems are eyeglasses and contact lenses. However, these methods may not always be the best solution in all circumstances. Consider people with presbyopia, the only time they need the eyeglasses is when they read at the close distance, for example their mobile devices. Because they have "blurred" vision, when they look at a sharp image on the screen, they will see a "blurred" one. What if we "blur" the image on the screen according to their vision in a specific way that compensates for visual imperfections? Ideally, they will perceive a sharp image on their retina. We can call such display a "vision correcting display".

Therefore, the work presented in this paper is an alternative to eyeglasses, contact lenses, and refractive surgeries for addressing the problem of blurred human vision. The idea is to "digitally" modify the content on a display device, so that when viewed by a particular user, it will appear in sharp focus for this individual. The amount of blur in the vision of that particular user can be retrieved using an eyeglasses prescription [16] or aberration measurements from a Hartmann-Shack wavefront aberrometer [1, 2] to identify the Point Spread Function (PSF) of the user's eye. The process that "digitally" modifies the content on a display device comprises both algorithmic operations that are functions of the particular user's optical aberrations and modified display optics at a hardware level that are the same for all users. Once the display device is built, the refractive errors are corrected digitally; no further adjustment to the optical hardware component is required for different users. We correct for myopia or hyperopia and also consider more complicated blur induced by higher order aberrations.

Fu-Chung Huang et al. proposed two different solutions in this field. In the first one, they use a multilayer display [10] and in the second one they utilize light field technology to generate sharp image out of the display plane [11]. From this starting point, we have made to three major software improvements on the light field idea, including the forward method,

optimization of the forward method, and the improvement on projection and prefiltering. The detailed algorithm explanations and analysis are presented in Section 4. In Section 5, one major hardware improvement is being proposed, which changes the pinhole array to a microlens array for the purpose of creating the light field. Section 6 presents a simulation program that has been developed to better evaluate the results.

Chapter 2

Related Work

2.1 Multilayer Display

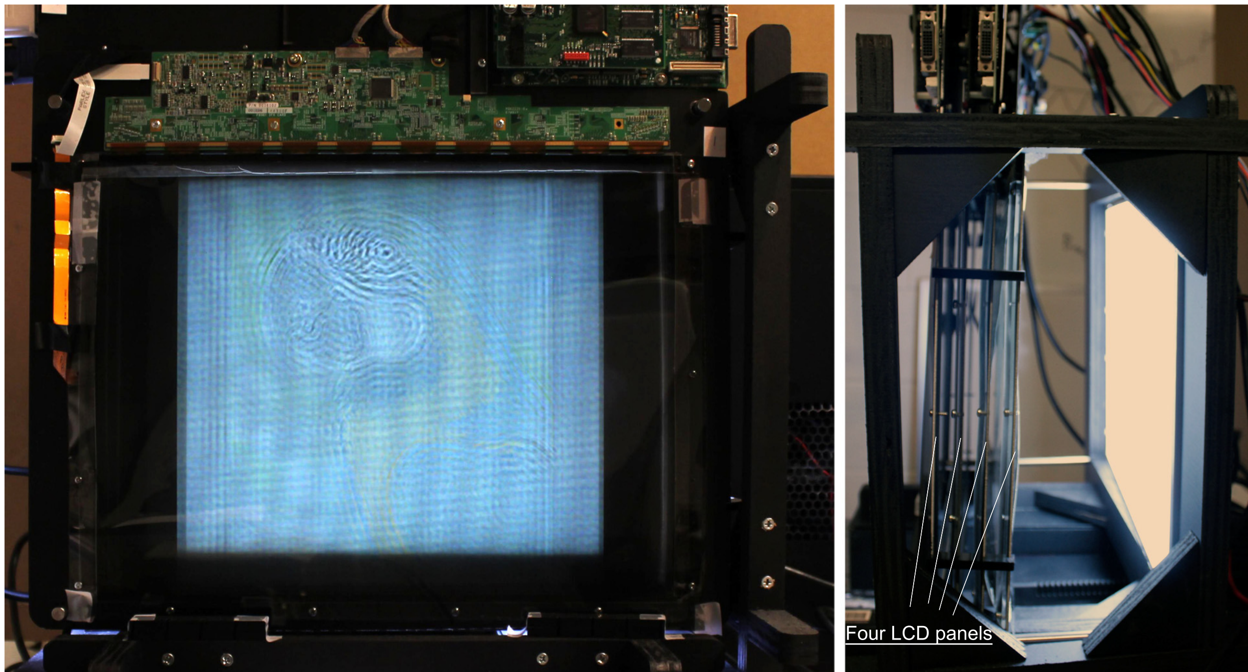

Figure 2.1: Prototype multilayer display, front view and side view.

Fu-chung Huang et al. proposed inverse prefiltering for emerging multilayer displays [10]. In order to solve the limitations on image prefiltering using a conventional display, the multilayer display enables an "all-pass kernel": there will be no zero-valued frequency responses [9].

In this work, Huang et al. have optimized the Michaelson contrast and the dynamic

range of the received image, as measured in a linear radiometric domain. Following Grosse et al. [7], incorporating the human contrast sensitivity function (CSF) [12] may allow further perceived gains in contrast.

Theory and experiment show that multilayer prefiltering achieves the primary goal: mitigating contrast loss and eliminating ringing artifacts observed with single-layer prefiltering. Yet, multilayer prefiltering comes at a cost of added components, increased computational complexity, and expanded display thickness, as shown in Figure 2.1. However, the introduction of the multilayer partition function is the first avenue to allow demonstrable increases in the contrast of images presented with prefiltered displays [5].

Chapter 3

Aberrations of the Eye

Eye aberrations can be categorized into two categories: lower order and higher order. Lower order aberrations, such as myopia, hyperopia, astigmatism, and presbyopia, can be described in terms of sphero-cylindrical values and can be corrected with the use of eye glasses or contact lenses. They contribute to 90% of loss of vision acuity [6]. The remaining 10% loss is due to a combination of particular eye imperfections, known as higher order aberrations, such as coma, trefoil, quatrefoil, and spherical aberration.

3.1 Zernike Polynomials

The wavefront of an eye can be measured using a Shack-Hartmann wavefront sensor. By analyzing the wavefront with Zernike polynomial functions, we can understand how the visual imperfections are formed.

A Shack-Hartmann wavefront sensor is an optical instrument used for characterizing an imaging system. It consists of an array of microlens with the same focal length, and a camera that is a CCD array or a CMOS array serving as the photon censor. In the system, each lenslet is focused onto the camera. In a Shack-Hartmann system, a laser creates a virtual light source in the retina. When a wavefront enters the lenslet array, a spot field is created on the camera. After that, each spot is analyzed for both intensity and location. The local tilt of the wavefront across each lens can then be calculated from the position of the focal spot on the sensor. Any phase aberration can be approximated by a set of discrete tilts. We can approximate the wavefront from the tilts calculated and measured from the samples of lenslets.

Zernike polynomials [17] are a set of basis functions used to describe a circular domain, $Z = W(x, y)$, a function useful to describe the wavefront surface of rays entering or exiting the circular pupil of the eye. In Figure 3.1, the wavefront map is generated by a sets of Zernike coefficients. The key property of Zernike polynomials is the decomposition of the shape function $W(x, y)$ into a frequency-series expansion of linear independent basis functions. Using a series expansion, different kind of aberrations can be expressed in terms of

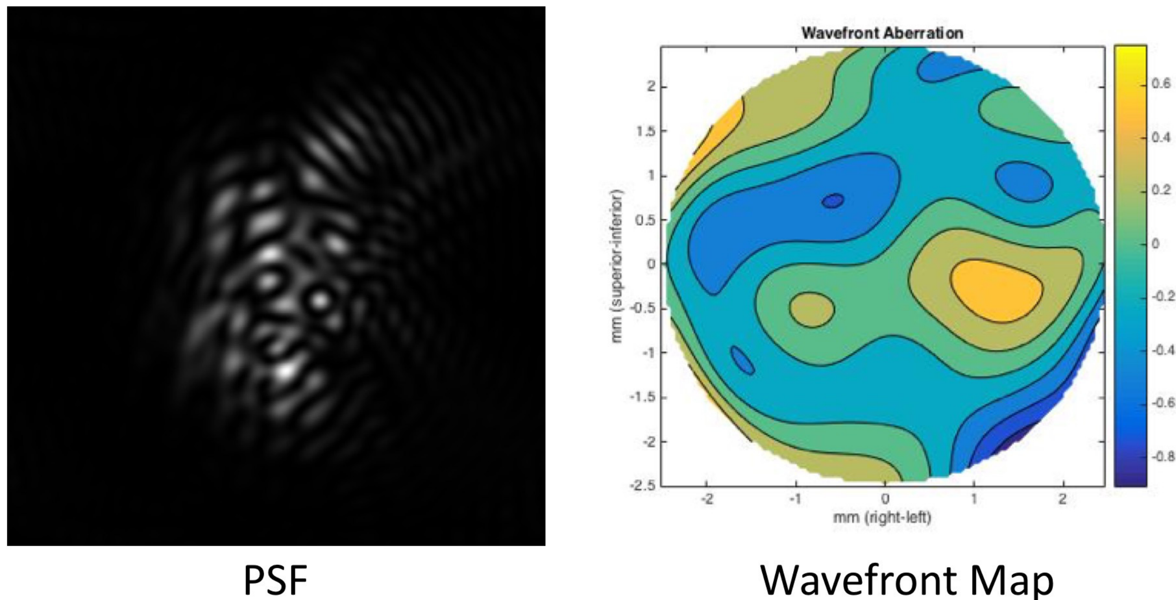

Figure 3.1: Visual aberrations can be described by the eye's point-spread function (PSF), often represented using wavefront maps. Left side is a PSF, and right side is a wavefront map. Data used in the plotting is collected from the author using a Shack-Hartmann wavefront sensor.

frequency coefficients. In the Listing 3.1, a sample calculation for the wavefront is achieved by using the Zernike coefficients.

3.2 Challenge of the Higher Order Aberrations

The higher order aberration terms are those Zernike polynomials with order 3 and above. These are more complicated, including coma, trefoil, quatrefoil, spherical aberration as well as many others that do not have names but are represented only by mathematical expressions. Higher order aberrations are traditionally hard to measure, and impossible to correct with eyeglasses.

In the technical report “A Framework for Aberration Compensated Displays”, authors identify some chanllenges of the higher order aberrations. “Although contact lenses have the potential to correct higher order aberrations, many eyes that exhibit higher order aberrations have corneas with an irregular surface shape that can prevent the successful fitting of a contact lens. Also, patients with presbyopia, that is, who require reading glasses, present

```
double x2 = x * x;
double y2 = y * y;

double W_x =
  c[1] * sqrt(4) +
  c[2] * 2 * sqrt(6) * y +
  c[3]* 4 * sqrt(3) * x +
  c[4]* 2 * sqrt(6) * x +
  c[5]* 6 * sqrt(8) * x * y +
  c[6]* 6 * sqrt(8) * x * y +
  c[7]* sqrt(8) * (9 * x2 + 3 * y2 - 2);

double W_y =
  c[0] * sqrt(4) +
  c[2] * 2 * sqrt(6) * x +
  c[3] * 4 * sqrt(3) * y +
  c[4] * (-2) * sqrt(6) * y +
  c[5]* sqrt(8) * (3 * x2 - 3 * y2) +
  c[6]* sqrt(8) * (3 * x2 + 9 * y2 - 2) +
  c[7]* 6 * sqrt(8) * x * y;
```

Listing 3.1: Code used in the prefiltering software described in Section 4.5 to generate the wavefront given the position and Zernike coefficients. In the code, only the first 8 terms are used.

still further challenges for satisfactory contact lens wear. For people whose vision involves higher order aberrations, it can be an ongoing struggle attempting to attain adequate vision correction." [9]

Chapter 4

Software Improvement for Light Field Displays

4.1 Introduction to Algorithms

Fu-Chung Huang et al. proposed to utilize light field technology to generate a sharp image beyond the display plane [11]. After that, three major software improvements have been made upon the light field idea, including the forward method in Section 4.3, optimization of the forward method in Section 4.4, and the improvement on projection and prefiltering in Section 4.5. We will compare the four algorithms in the following sections.

Projection and Prefiltering

All the algorithms presented in the later sections can be divided into two separate parts: The first part takes charge of building the **projection** relationship, and the second part does **prefiltering**.

The **projection** relationship is the mapping relationship from the point on the display plane to the point on the sensor (retina). Due to the eye aberrations, each display point can be mapped into an area on the sensor plane. This projection relationship depends greatly on the experiment settings, the eye condition and the physical light field display we use.

Prefiltering is the actual step to digitally modifying the content of a display. We take the projection relationship and an image as input, and output a transformed image that should be displayed. The viewer should see a sharp image because this transformed image is carefully created for the viewer.

Definition of Symbols

To facilitate our following discussion and prevent possible misunderstanding, we define following symbols. Symbols defined below are shown in Figure 4.1 as well.

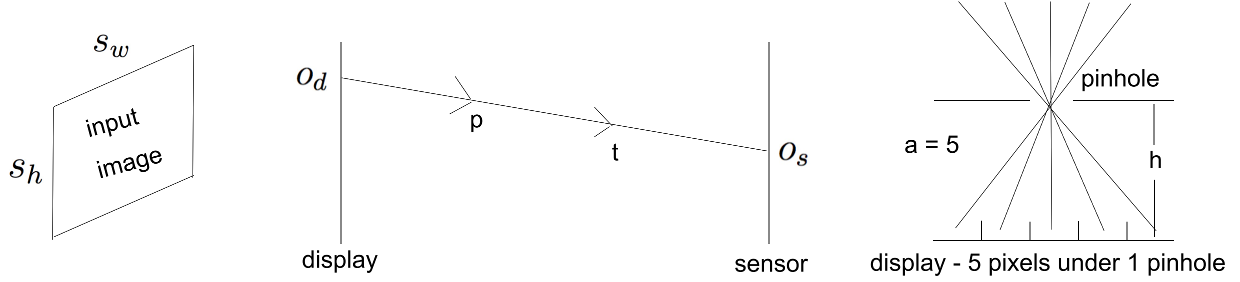

Figure 4.1: This is a figure illustrating defined symbols. (left) The input image is s_w by s_h pixels. (middle) p and t will specify a light ray, and o_s, o_d are positions where pt hits the display and the sensor respectively. (right) There are 5 pixels under each pinhole. h is the depth distance between the pinhole and the display plane.

The size of the input image is s_w by s_h pixels. The input image is the original one that normal people should see in the normal display.

We define two functions f_s and f_d, both of them have the same parameters:

- p and t both stand for position of an arbitrary point in real three-dimensional space. So p and t will specify a light ray.
- e is the experiment settings. If we only consider the defocus error for the eye, it will only contain the physical distance d between the display and the lens(eye).
- c is the condition of the camera (human eye). If we only consider the defocus error for the eye, this parameter will only contain the object distance u from the camera (human eye). The object will create a sharp image at the sensor (retina) if the distance between that and the lens is the object distance.

So f_s will receive those parameters and compute the position of the point o_s where the light-ray pt hit the sensor. And, accordingly, f_d will return the position of the point o_d where the light-ray pt hits the display.

$$f_s(p, t, e, c) \rightarrow O_s$$
$$f_d(p, t, e, c) \rightarrow O_d$$

Note that f_s and f_d take experiment settings and eye conditions as input. Higher order aberrations support should be only related to these two functions. There are several ways to implement this function, and in Huang's version he uses Zernike polynomials [4]. And we pack all the aberration-related computation into this function, so it becomes an API and is decoupled from our main algorithm.

We define two other functions m_s and m_d to transform an arbitrary position into a discrete two-dimensional index. They both take a parameter p which stands for an arbitrary

point either in the sensor plane or in the display plane. m_s and m_d will transform them into the discrete two-dimensional index respectively.

$$m_s(p) \rightarrow (a_s, b_s)$$
$$m_d(p) \rightarrow (a_d, b_d)$$

Angular resolution a is defined to be the ratio of screen resolution to the pinhole mask or microlens array mask resolution. For example, if each pinhole and its surrounding cover 5 by 5 screen pixels, we say that $a = 5$ for this device.

Depth h is defined to be the distance from the surface of pinhole mask or microlens array mask to the actually display plane.

4.2 Original Algorithm using Light Field

Projection

The projection relationship is presented by a matrix in Huang's algorithm. He assumes that the sensor of the camera has the same resolution as the pinhole mask, which is (s_h/a) by (s_w/a). Then he transforms the display and the sensor pixels into two one-dimensional arrays with length $L_d = s_h \cdot s_w$ and $L_s = s_h \cdot s_w/a^2$, respectively.

Next, the algorithm builds a projection matrix. Firstly, an all-zero matrix P with the dimension of L_d by L_s is created. Secondly, for every sensor pixel p_s, the algorithm randomly samples n points $\{o_n\}$ on the aperture, and for each sample point o_i, it constructs a light ray $p_s - o_i$. Here the algorithm treats the sensor pixel as a point locating in the center of the pixel. Thirdly, the algorithm applies the function f_d we defined in Section 4.1 to get the arbitrary point position p_d on the display. Next function m_d is called to convert p_d to 1-d index I_d. Lastly, the algorithm adds the element in the corresponding position of that projection matrix by $1/n$.

Prefiltering

This projection matrix P is saved after the projection phase. We expands the input image G to a one-dimensional vector b and we denote the transformed image to be x. So the prefiltering can be described by the following equation:

$$P \cdot x = b$$

Since P is not a square matrix, it cannot be adapted to some numerical methods. So the Huang's algorithm transforms P into a square matrix by following step. The algorithm multiplies the left-hand and right-hand of the equation both by P^T:

$$(P^T \cdot P) \cdot x = P^T \cdot b$$

Define $H = (P^T \cdot P)$. It is very likely that H is singular and this linear system does not have a possible solution. In order to solve the linear system, a small positive value called λ is defined and added into the system where I is the identity matrix:

$$H' \leftarrow H + \lambda \cdot I$$

λ would introduce error into the system but this error is negligible. H will not be singular anymore.

Instead of solving the system directly, Huang's algorithm changes it into a optimization problem with the target:

$$\textbf{Minimize: } (H'x - P^T b)^T \cdot (H'x - P^T b)$$

Huang's algorithm uses a method called L-BFGS [13] to solve this optimization problem. Since no restriction is added to this optimization problem, the result may contain large positive values or even negative values. In order to make it capable to be displayed, normalization is applied to the result image. The max value is mapped to 255 and min value is mapped to 0 respectively. Other values are computed by linear interpolation.

This process is repeated three times for RGB three channels seperately. In the end, RGB channels are combined into one complete output image.

Analysis

Advantages

- The result in Figure 4.2 [5] shows that Huang's algorithm achieves superior contrast and resolution compared to all other implementations such as multilayer display.
- It can accommodate the situation where light rays emitted by one screen pixel that go through several different pinholes can all be received by the sensor (retina).
- It has higher resolution when the image is monochrome.

Disadvantages

- All the computation is based on the assumption that the sensor resolution is $1/a$ of the original display, which is certainly not true because the sensor resolution is commonly much larger than the display resolution, let alone the viewer's retina resolution.
- Huang's algorithm assumes that any point of one screen pixel can emit all the possible colors. However, that is not true and each point can only emit one type of color and the type depends on the location of that point.
- It uses L-BGFS to solve that optimization problem. This makes the prefiltering speed depends greatly on the environment settings and viewer's eye condition, and the complexity is large compared to other method.

- In order to stabilize the prefiltering speed, systematic error is introduced.

4.3 Forward Method

The differences between the forward method and Huang's original algorithm are mainly in the projection part. In the forward method, sampling starts from the display instead of from the sensor. There is an important modification compared to the original one:

The sensor has the same resolution as the display.

Therefore, the dimension of the projection matrix in the forward method becomes $s_h \cdot s_w$ by $s_h \cdot s_w$ and it is square in natural. For every display pixel p_d, a sample light ray is constructed by picking up the center position o_c of the pinhole that covers it. In the forward method, one display pixel is still treated as one point that locates at the center of that pixel for simplicity. Then function f_s is applied to get the position that this light ray hits the sensor plane. In this way, the total number of sampling rays is greatly reduced. Thus, the time needed to construct that projection matrix is greatly reduced as well.

Analysis

Advantages

- Huang's algorithm spends a lot of time doing unnecessary sampling. For example, when a pinhole mask is placed on the top of the display with angular resolution of 5, only 1/25 of all the light rays can pass this pinhole mask. The forward method cuts a lot of such samplings in order to reduce a lot of computations. A 10 times performance improvement can be achieved.

Disadvantages

- The forward method assumes for simplicity that only the light rays that go though the designated top pinhole can be received by user's eye. However, this may lead to severe ghost image issues without a careful choice of the depth value.
- Starting from display cannot measure how much light one sensor pixel receives.

4.4 Optimization of the Forward Method

In order to improve the speed of the forward method, the following steps were performed in modifying the code:

- Translation of codebase from Matlab to C++.

- Reducing the number of linear systems to solve from 3 to 1 per image.
- Computing the pseudoinverse of the projection matrix and caching it to improve the speed of solving each image.
- Changing the computation from matrix multiplication to direct assignment.

Translate from Matlab to C++

For the programming language of the program, Matlab was originally used but is slow for programs that involve a large number of computations, so we switched to C++. In order to make translation from Matlab to C++ easier and reduce the number of new bugs, we used the software packages Armadillo and Eigen which provided similar linear algebra capabilities to those of Matlab, as well as FreeImage to facilitate working with images. Testing showed that the C++ version of the original method took about 0.3 seconds per image.

Reducing the Number of Linear Systems

As before, the technique for solving the corrected image from the original image was to find the least squares solution $(P^T \cdot P) \cdot x = P^T \cdot b$, where P is the projection matrix which represents the alterations applied to the original image x which causes the viewer to perceive the image as b.

In order to understand the structure of the images x and b, we must examine how images are laid out in the computer. An image is laid out as a grid of pixels, each with some color associated with it, and each pixel is comprised of 3 component values: usually the primary colors red, green, and blue, which together can comprise most of the colors available on displays. Therefore, an image is equivalent to a set of 3 images which are its red, green, and blue components.

Originally, we solved the system separately for each color component, resulting in 3 linear systems per image. To reduce overhead of separating the image and recombining the image, we modified the software to work on the image's original form, which has a pixel's red, green, and blue components adjacent to each other by performing a simple trick, replicating the entries in matrix P so that P's dimensions are now $3\times$ larger as well as the dimensions of vectors x and b.

Compute and Cache

We notice that for a given projection matrix P, we may want to solve $(P^T \cdot P) \cdot x = P^T \cdot b$ for many different images b. We made this process more efficient by finding the pseudoinverse $(P^T P)^{-1} P^T$. This allows us to find the solution x by just multiplying the saved pseudo-inverse by any new image b, which is much faster than using our previous iterative methods of solving the system. The pseudoinverse can be saved until a new projection matrix P is to be used. After this change, performance was further improved to 0.015 seconds per image.

Change Multiplication to Direct Assignment

Further testing showed that using matrix multiplication to find the solution $x = (P^T P)^{-1} P^T b$ was unnecessary and speedup could be achieved by noticing that the pseudoinverse $(P^T P)^{-1} P^T$ was very sparse. Each row i in the pseudoinverse had very few non-zero values and corresponded to which pixel(s) in the original image b mapped to pixel i. Therefore, we could just go through each pixel i in the output image x, use the pseudoinverse to look up which pixel in b mapped to this location, and set the color. Directly assigning pixel values to the solution image using the pseudoinverse as a lookup table for pixel mappings reduced computation time to 0.002 seconds.

Analysis

Advantages

- If we can get pseudoinverse successfully, we can save it for future usage. And since the it is only a matrix-vector multiplication, the complexity is very low and the speed is reasonably fast and constant for all situations.

Disadvantages

- This method introduce unavoidable error in calculation. Although λ is a small amount, but this may have non-negligible impact on the final result.

4.5 Improvement on Projection and Prefiltering

Projection

The idea of forward sampling is still used for faster speed, but instead of using a projection matrix to represent this projection relationship, a one-to-one pairing is used. The overall improvements assume that all the light rays emitted by one screen pixel that can be received by our eyes go through only one pinhole.

Therefore, there will be only one continuous region on the sensor that can be illuminated by one screen pixel. This is a comparatively loose condition that will hold true for most experiment settings. In this way we choose the "center" of this continuous region to be the partner of this screen pixel. In fact, it is generally hard to find this center, so the following steps are performed to find this mapping relationship.

For every subpixel p_d, we pick its center position as the position of that subpixel. Then, we enumerate all the pinholes to find the one that light rays goes through this pinhole can be received by user's eye. If this pinhole was found, we pick its center o_h, and use f_s to compute the position p_s on the sensor plane that light ray p_d-o_h hits. This mapping pair (p_d, p_s) is saved in a vector, and then goes through all the screen subpixels.

Prefiltering

After the Projection Phase, a vector of mapping pairs is saved. And for each pair (a, b) in this vector, a stands for a screen subpixel, and b stands for a arbitrary point on the sensor plane. And we compute the value for each color by the following method. Given an arbitrary position on the sensor, we can use bilinear interpolation to compute the color for that arbitrary position using the peripheral pixel's color information. Then we pick the specific color value for that subpixel.

Analysis

Advantages

- The project speed is roughly the same as forward method.
- The light ray computation is corrected. Display arrangement is taken into consideration, so the color separation effect is reduced.
- The prefiltering speed is fast and steady, roughly 0.03 second per image. In Figure 4.3, less artifacts are observed in the prefiltered image.

Disadvantages

- Resolution is not as good as with Huang's algorithm when the image is monochrome.
- The center position computation part is not precise.
- Cannot handle the situation when the one-to-one pairing assumption does not stand.

Figure 4.2: Evaluation of vision-correcting displays. We compare simulations both qualitatively and quantitatively using contrast and quality-mean-opinion-square (QMOS) error metrics. A conventional out-of-focus display always appears blurred (second column). Multilayer displays with prefiltering improve image sharpness but at a much lower contrast (third column). Light field displays without prefiltering require high angular resolutions, hence provide a low spatial resolution (fourth column). The proposed method combines prefiltering and light field display to optimize image contrast and sharpness (right column). The QMOS error metric is a perceptually linear metric, predicting perceived quality for a human observer. We also plot maps that illustrate the probability of an observer detecting the difference of a displayed image to the target image (bottom row). Our method performs best in most cases. (Source images courtesy of flickr users Jameziecakes, KarHan Tan, Mostaque Chowdhury, Thomas Quine (from top)) This figure is from [5].

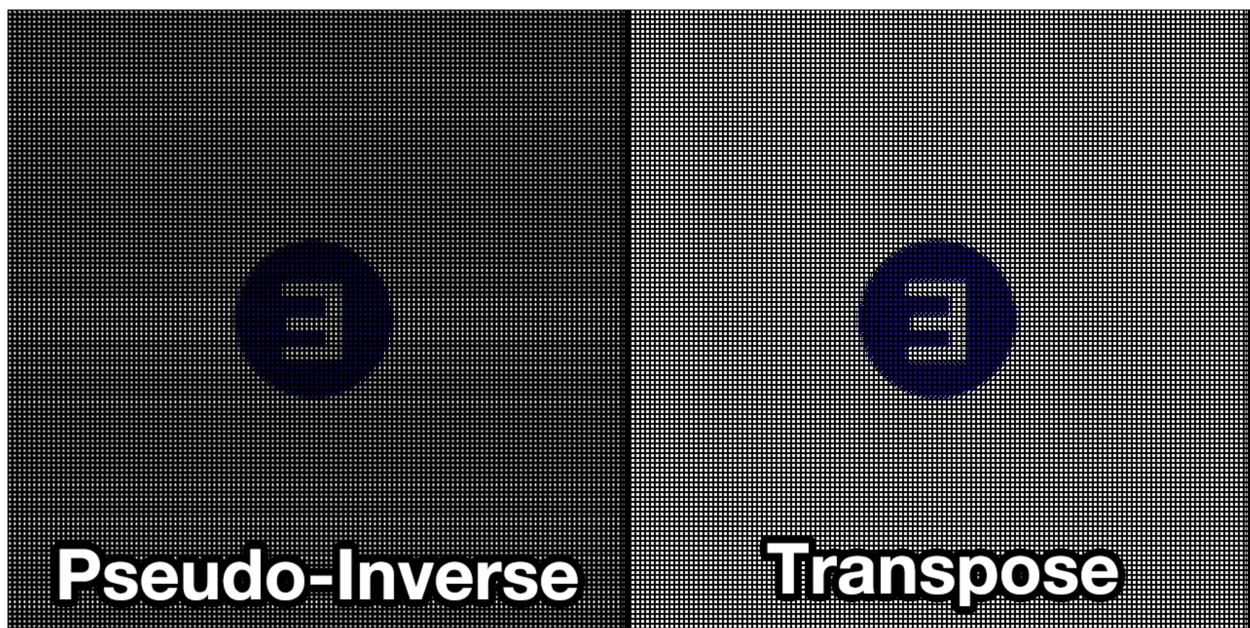

Figure 4.3: Comparison of the prefiltered images generated by two algorithms described in optimization of the forward method (left) and overall improvements (right). The two prefiltered images are generated with the same parameters. We can notice that there is the strange bright and dark pattern in the left image. The same artifact does not show in the right image.

Chapter 5

Hardware Improvement for Light Field Displays

5.1 Prototype Using iPhone 6

Figure 5.1: Left side is the iOS App Demo. In the middle, it is the top view for the current prototype. Right side is the side view for the same prototype.

Our current implementation of the prototype is a light field display using a conventional parallax barrier - pinhole mask. When driven by the algorithm in Section 4.5, the light field

display achieves good sharpness and contrast. After installation, the pinhole mask stays on the iPhone 6 screen. Eye focus distance (focus), pupil size (diopter), and viewing distance (distance) can be adjusted by sliders at the bottom of the app. These coefficients are used to specify different eye aberrations and to construct the projection matrix. Our iOS app displays a prefiltered static image (left) and corresponding rotating image (right) for tests and experiments. The projection matrix maps original images (two "E"s in the middle) to be prefiltered images (two "E"s at the top). The times needed to calculate the projection matrix and prefilter an image are indicated below the sliders accordingly. The time for pre-filtering an image given the projection matrix takes around 0.05sec, which is near real time rendering. When environmental setting changes, the program takes around 0.05sec to produce the new projection matrix. The algorithms are optimized so that mobile phones can perform prefiltering efficiently.

5.2 Pinhole Array to Microlens Array

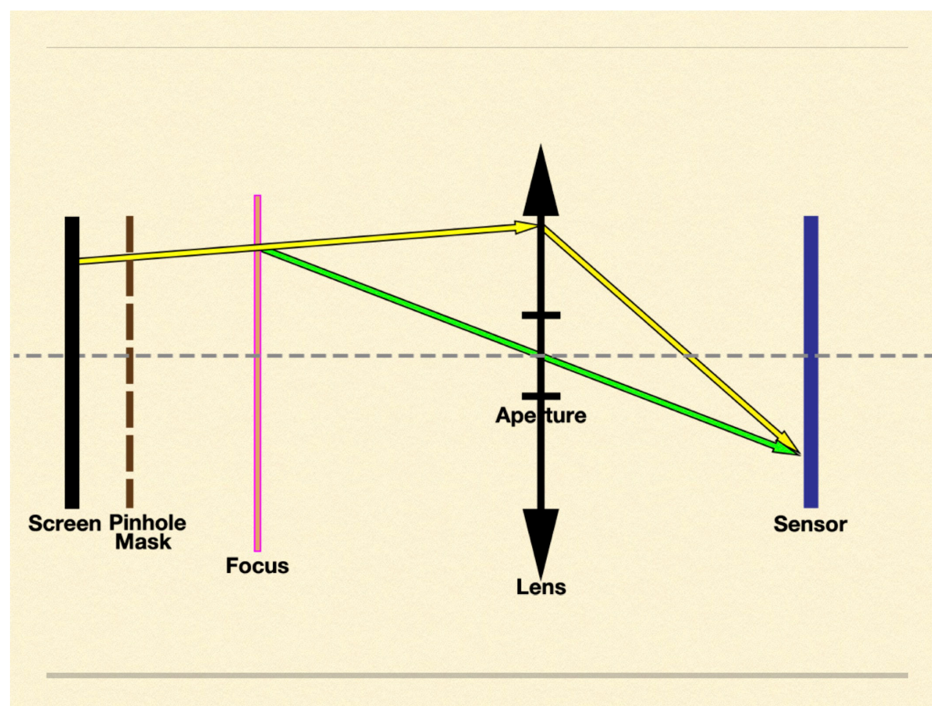

Figure 5.2: Light ray path going through one pinhole.

A microlens is a small lens, generally with a diameter less than a millimetre and often as small as 10 micrometres. As shown in Figure 5.3, we place the microlens on top of the pixels to construct the light field. The depth between the microlens and the pixel plane is equal to the focal length of the microlens, so a parallel beam will be emitted for each individual pixel.

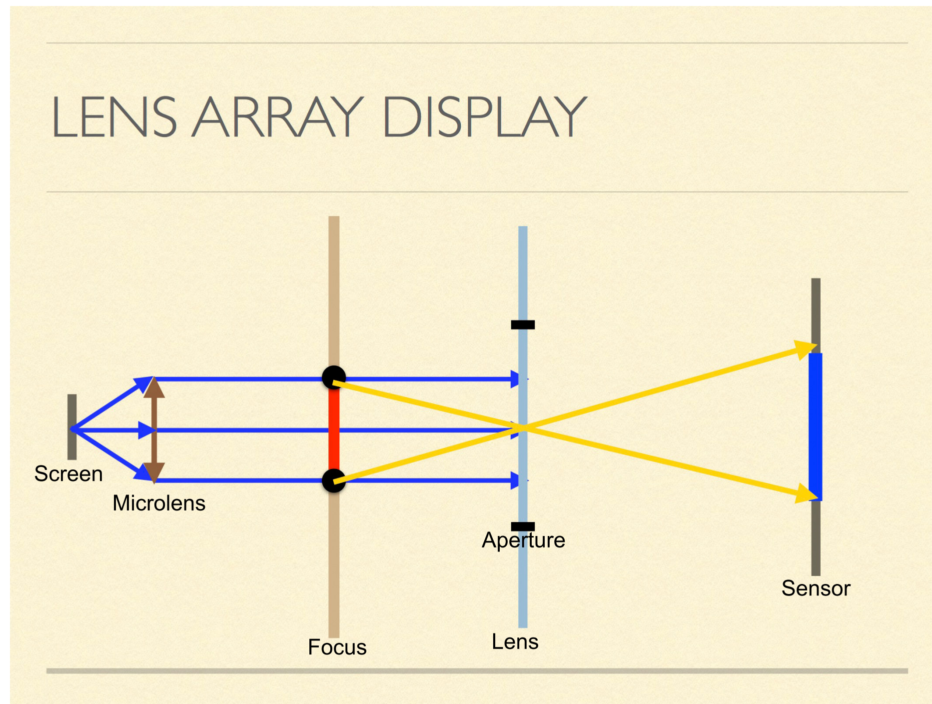

Figure 5.3: Light ray path going through one lenslet.

Like the pinhole, each microlens will correspond to 5 by 5 pixels. Given the pixel pitch for iPhone 6 is $0.0779mm$, the diameter for microlens will be $0.0779mm \times 5 = 0.3895mm$.

Microlens array offers a great advantage of a significant increase in brightness. For pinhole array, the pinhole filters out a significant number of light rays. If the pinhole size is similar to one pixel, theoretically, the microlens array will increase the brightness by 25 times given 5 by 5 pixels under each pinhole.

Chapter 6

Simulation

A computer simulation was developed to model the effectiveness of various light field displays. In this research, we have different physical models of display methods. We found that the correctness and the efficiency of them are not the same. We want to evaluate the correctness and quantify their efficiency to have a better understanding of the physical models and to decide which display method to choose. Simulation is essential for making an accurate design of the physical model, because the experimental setting in the simulation can be easily changed for testing.

There are several motivations for simulation:

- More efficient than physical experiments; allows for quick tuning of experimental parameters.
- Avoids the high cost of expensive microlens arrays with inaccurate parameters.
- Lenses for certain kinds of eye aberrations are not readily available.

6.1 Simulation Algorithm

The algorithm of the simulation is mainly based on ray tracing. It is generated from the physical model explaining how the light rays travel through camera (eye) and lenses. In the physical world, a pixel of light can be seen because of the light is sent from the surface of an object, going through the air straight and finally reaching the eye. Multiple light rays travel in this way and finally reach human eye to form projected image in the retina. Our task is to simulate this process but in a more sophisticated way by incorporating the pinhole array and the microlens array.

The ray tracing algorithm is a backward tracing, where light rays are emitted from the sensor plane to the display plane. For the projective image we need to generate, we want to find the corresponding pixel of the original image for each pixel in the projected pixel knowing the image size, the screen size, parameters of the lens, the depth from the lens to the screen. The process starts from the input image.

```
1  void thread_work(int begin, int end,
2    const LightFieldDisplay* display) {
3    //ITERATE OVER SCREEN (OUTPUT) PIXELS
4    for (int iy = begin; iy < end; iy++) {
5      for (int ix = 0; ix < screen_size; ix++) {
6        //ITERATE OVER A SINGLE SENSOR PIXEL (INPUT)
7        for (int dy = 0; dy < sensor_sample_time_y; dy++) {
8          for (int dx = 0; dx < sensor_sample_time_x; dx++) {
9            //ITERATE OVER RANDOM APERTURE POINTS
10           for (int i = 0; i < aperture_sample_time; i++) {
11             //SIMULATION COMPUTATIONS CALCULATING COLOR
12           }
13         }
14       }
15     }
16     //STORE CALCULATED COLOR IN OUTPUT IMAGE AT (ix, iy)
17   }
18 }
```

Listing 6.1: Examining the structure of the CPU-based simulation code.

- First, scale the input image to fit the size of the screen.
- Second, send rays from the screen pixel to the sensor (eye) and calculate the position on the sensor. We need to calculate the position that a ray going through a pinhole and a lens.

6.2 Simulation Speedup

Speedup Motivation

The simulation is used to experiment with a variety of parameters (e.g. the configuration of the display) and their effects on the perceived image quality. Therefore to allow for effective experimentation, the simulation needs to run within a reasonable amount of time on a personal computer.

However the original CPU-based implementation shown in Listing 6.1, even when taking advantage of multiple threads, proved to be too slow.

Since the code in Listing 6.1 is a multi-nested for-loop, it is computationally expensive. The structure of the simulation is described as follows:

- The outer two for-statements iterate over the “screen’s” pixels (the pixels of our output image).
- The next two for-statements iterate over a single sensor pixel (our input image), sampling it in a grid pattern with dimensions (sensor_sample_time_y) × (sensor_sample_time_x).

- The most inner for-loop iterates over randomly generated aperture points.

In order to generate a realistic image, a large number of aperture points need to be used (aperture_sample_time will be fairly large). Recall that our aperture is the opening through which rays are drawn from our screen to our sensor. In a physical system, there are an infinite number of light rays from one point on the sensor through a finite-sized aperture. Thus, to accurately model the physical system, the simulation must randomly sample a large number of aperture points per sensor position (around 1000 points). Thus, the CPU-based simulation scales as $O((a^s)^n)$, where a = number of aperture points sampled per sensor position and s = number of sensor positions sampled per sensor pixel and n = number of screen pixels sampled. If we are not tuning the number of sensor positions (for all simulations thus far we have kept a constant s = 4), then we can simplify the runtime to $O(a^{4n})$.

Speedup Using GPU

Examining the simulation ray-tracing algorithm, it becomes apparent that computing the output color of one screen pixel is independent of the calculations for the others. Also, the calculations per pixel are a series of relatively simple operations. So ideally if we could ray-trace every screen pixel in parallel then we should be able to reduce our runtime complexity from $O(a^n)$ to $O(a)$, which would drastically improve the simulation's runtime performance.

This is where the GPU is introduced. The GPU's main task is to render images on our computer screens with minimal latency. Given that a screen has many thousands of pixels, this is a computationally intensive task (analogous to our own simulation, and image processing in general). However the GPU successfully accomplishes this by making use of its massively parallel architecture. Of course each GPU core is not nearly as complex and versatile as a CPU core, however these light-weight cores are particularly efficient with simple calculations.

Although the GPU is typically used for rendering pixels on screens, its speed can also be leveraged for general purpose computations (GPGPU). There are several GPGPU libraries available, most notably OpenCL and nVidia's CUDA. For our simulation, we make use of OpenCL which can be run with any graphics card that supports OpenCL (most modern graphics cards) whereas CUDA is solely used on nVidia's graphics cards.

OpenCL Basics

For the task of writing code that makes use of OpenCL and the GPU, there are two things to consider: host code and kernel code.

Writing Host Code: This is code run on the CPU used to prime the GPU for us. Although there are many steps in the setup code, most of it can be reused from application to application. We only need to change application specific details (e.g. input/output buffers, kernel args, etc.). The following is a general list of what needs to be done:

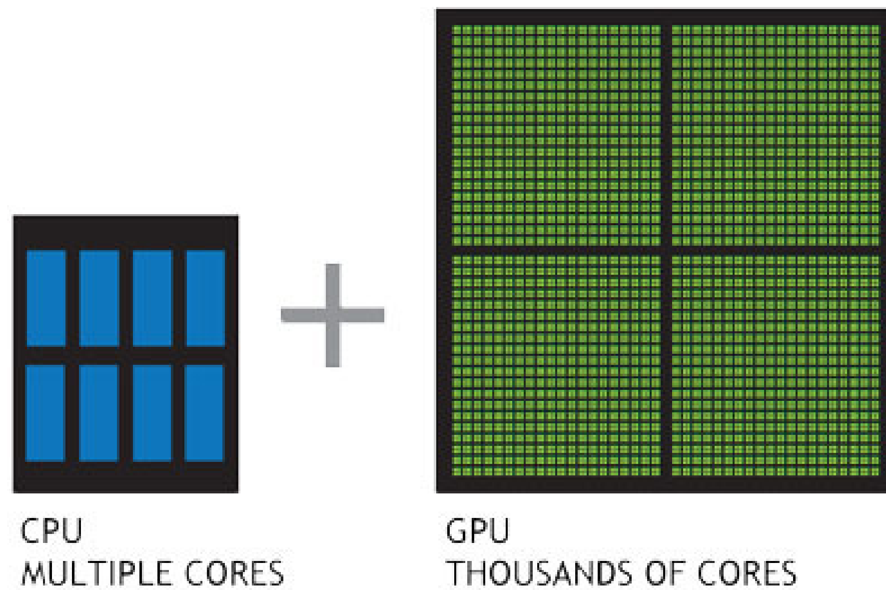

Figure 6.1: GPUs have thousands of cores to process parallel workloads efficiently. [15]

- Setup the environment - We need to discover and create a context for available OpenCL compatible platforms (GPUs, CPUs, etc.).
- Declare input and output buffers and prepare move of data buffers to GPU.
- Runtime Kernel Compilation - The kernel is the source code for the GPU, need to compile it before running
- Run the Program
  - Set kernel arguments; the arguments for "__kernel void funcName() {}"
  - Set size of global and local work groups; essentially laying out how the computation should be partitioned for parallelization as shown in Figure 6.2
  - Run
- Read result back to host

Writing Kernel Code: This is code to be run on the platform (e.g. GPU). This is where the logic of our program (the parallelized parts) should be.

- Written in OpenCL C, which is essentially C99
- The starting point for a kernel program (the main function) is given the __kernel void funcName() {} signature

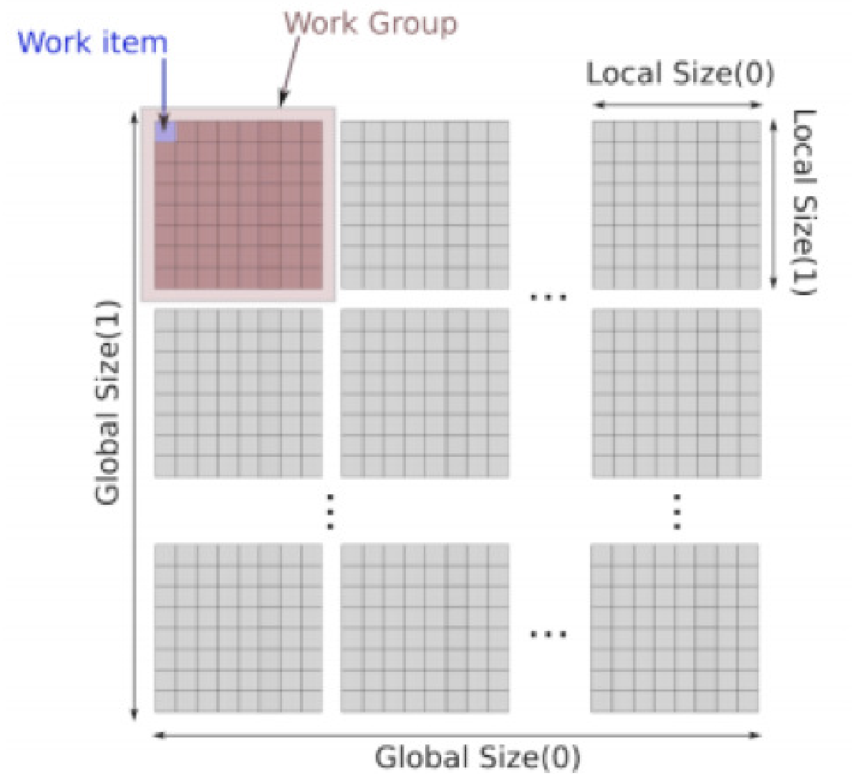

Figure 6.2: Compute with GPU.

- Use get_global_id(⟨*dim*⟩) to determine where the current process is in the global workspace in the specified dimension (we set the dimensions of the work groups in our host code). Similarly get_local_id(⟨*dim*⟩) is used to find where the process is in the local workspace. This is important because the core needs to know what set of the data it should work on.

OpenCL Implementation

As shown in the Listing 6.2, the structure of our kernel code is exactly the same as before, minus the outer loop over the image pixels. This is because we want to parallelize our code so that each core works on a designated subset of pixels (its work group).

Note that the __global tag means that this is data that has been buffered into the GPU by the host and is globally available on GPU memory.

```
1  __kernel void originSim(__global int* result_arr,
2                          __global float* aperture_samples,
3                          __global uchar* image,
4                          int sensor_size, float sensor_pixel, float rgb_pixel,
5                          int sensor_sample_time_y, int sensor_sample_time_x,
6                          float screen_offset_x, float screen_offset_y,
7                          int screen_size, float screen_pixel,
8                          float focusObj, float disObj,
9                          float disImg,
10                         int aperture_sample_time)
11 {
12     // (ix, iy) are used to determine what pixel in IMAGE is being worked on
13     const int ix = get_global_id(0);
14     const int iy = get_global_id(1);
15     //ITERATE OVER A SINGLE SENSOR PIXEL (INPUT)
16     for (int dy = 0; dy < sensor_sample_time_y; dy++) {
17         for (int dx = 0; dx < sensor_sample_time_x; dx++) {
18        //ITERATE OVER RANDOM APERTURE POINTS
19             for (int i = 0; i < aperture_sample_time; i++) {
20     //SIMULATION COMPUTATIONS CALCULATING COLOR
21             }
22         }
23     }
24    //store calculated color in RESULT_ARR AT (ix, iy)
25 }
```

Listing 6.2: OpenCL kernel code.

6.3 Performance Analysis

Analysis of OpenCL Speedup Over Different Screen Sizes, Sampling Times, and Displays

We investigated the speedup due to OpenCL over three different parameters: screen sizes, sampling times, and displays. We wanted to examine the rate at which run time increases with respect to screen size and see all screen sizes within the range of 240 pixels to 800 pixels can run in a short amount of time. We wanted to examine the relationship between runtime and sampling time to find the upper bound in which the simulation can run quickly with a large aperture sampling time. Originally, our team had a tradeoff issue in which the simulation could run quickly with a small aperture sampling rate, which yielded less accurate results, or run slowly with a large aperture sampling rate, which yielded more accurate results. The goal of using OpenCL was to allow the simulation to run at a large aperture sampling rate and run in less than five minutes. The third parameter that was examined was the display that was simulated (original display, pinhole array display, microlens array display). Each display uses a different algorithm for its computations, and the simulation

team compared the algorithms to see which ones were more efficient or less efficient and determine whether the one or more of the display algorithms should be improved.

Table 6.1: Specifications of the benchmark computer.

Computer System:	Windows 10
GPU	
Max clock frequency:	1070 MHz
Max memory allocation:	3019898880 Bytes

Table 6.1 displays basic statistics about the computer used in the analysis of OpenCL run time. The OpenCL code was run on a Windows 10 machine with a maximum clock frequency of 1070 MHz and a maximum memory allocation of approximately 3 Gigabytes. The computer is relatively new and therefore produces runtimes that are expected to be much faster than other machines.

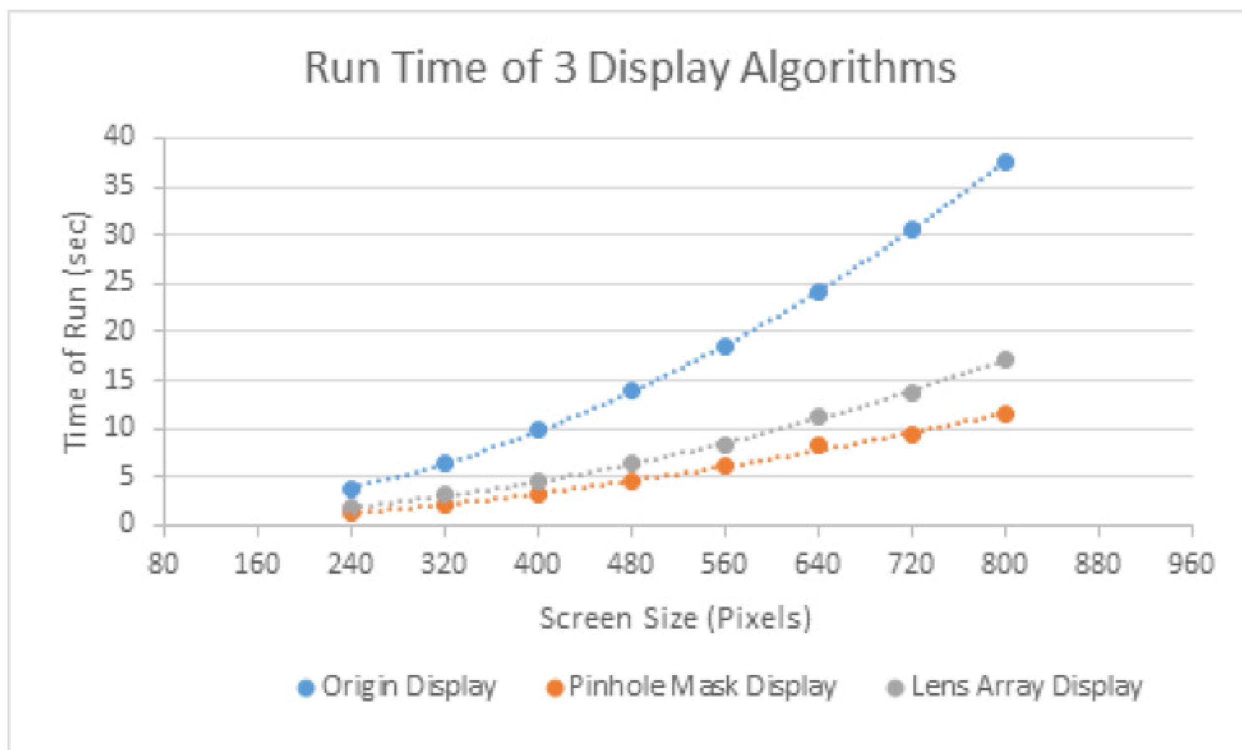

Figure 6.3: Run with a sample time of $100 \times Area(Aperture)$.

According to Figure 6.3, the original display takes approximately three times as much time as the pinhole array display and twice as much time as the microlens array display. A screen size of 800×800 pixels takes under 40 seconds for the original display, under 20 seconds for the microlens array display, and under 15 seconds for the pinhole array display.

For all three displays, run time increases quadratically ($O(n^2)$ time) with respect to screen size.

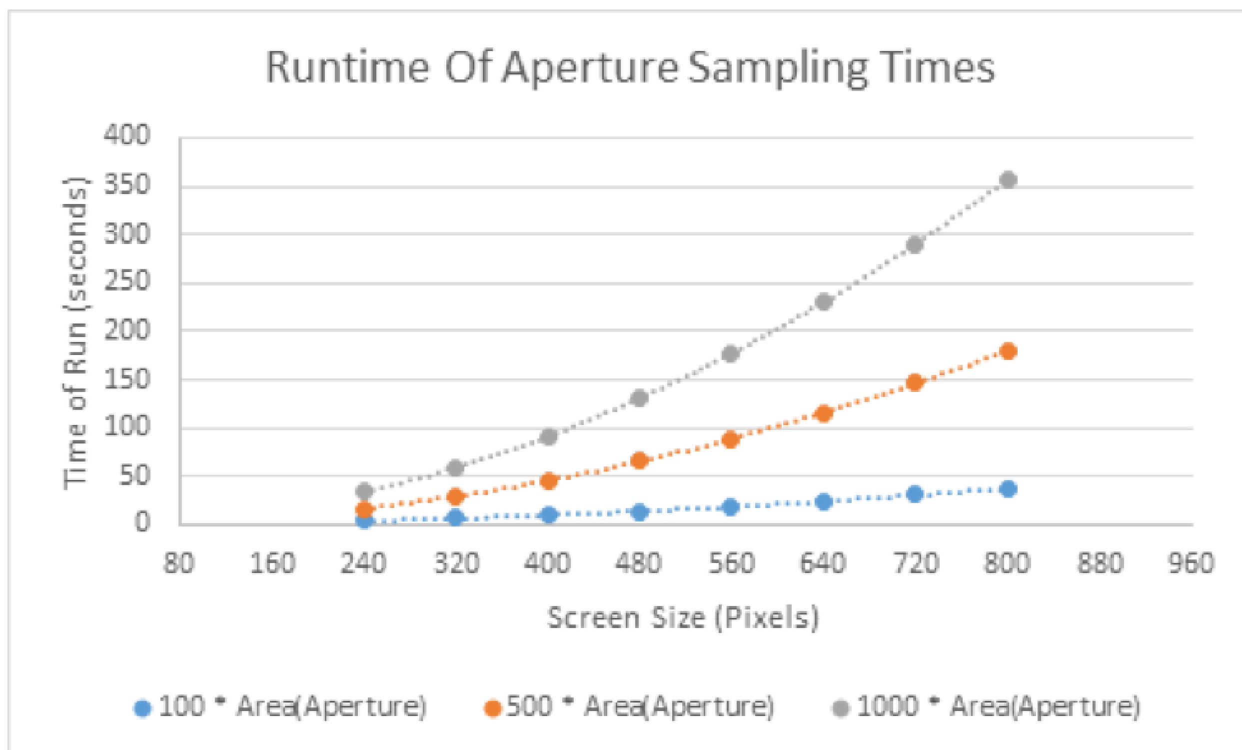

Figure 6.4: Runtime of aperture sampling times using the original display.

According to Figure 6.4, for the same aperture sampling time, run time increases quadratically with screen size. An aperture sampling time of 100 can run under 50 seconds for screen size of 800 pixels or less; a sampling time of 500 can run under 200 second; a sampling time of 1000 can run under 400 seconds. These times are for an original display. (Figure 6.4)

According to Figure 6.5, for the same screen size, run time increases linearly with aperture sampling time. This means a run with a sampling time of 1000 takes 2 times longer than one with a sampling time of 500, and 10 times longer than one with a sampling time of 100. (Figure 6.5).

Table 6.2: Runtime for 400px original display

Aperture Sampling Time	Runtime (Seconds)
5000 * Area(Aperture)	448.041
10000 * Area(Aperture)	897.879

Table 6.2 displays the runtime of the simulation if one used extremely large aperture sampling times. A run for an original display, screen size of 400 pixels, and a sampling time

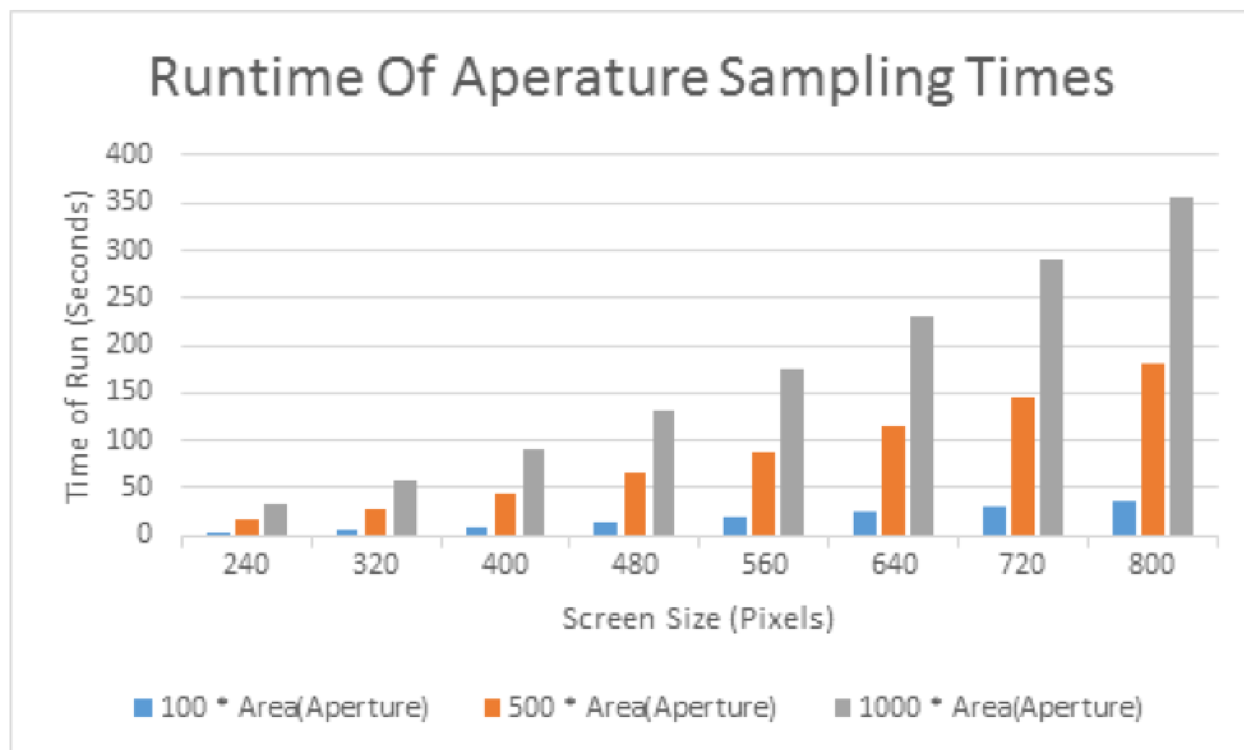

Figure 6.5: Runtime of aperture sampling times using the original display.

of 5000 $\times$ $Area(Aperture)$ should run in 448.041 seconds (about 7.5 minutes). The same run except with a sampling time of 10000 $\times$ $Area(Aperture)$ should run in 897.879 seconds (about 15 minutes).

Paralleled simulation code through OpenCL gives much faster running times than non-parallelized code. When the simulation code was not parallelized, a simulation with the original display with a screen size of 400 pixels and aperture sampling time of 100 took exceeded hour. That same simulation takes under two minutes. If we consider feasibility as running under three minutes, then we can conclude that OpenCL is feasible with an original display for a screen size of up to 800 pixels and a sampling time of up to 500*Area(Aperture), or a screen size of 560 pixels and a sampling time of 1000 $\times$ $Area(Aperture)$. We also note that pinhole array and the microlens array display model would run even faster.

Effect of OpenCL On RunTime of Prefiltering Algorithm

Before a simulation of a vision correcting display is run, the image needs to be prefiltered on a screen. The role of prefiltering is to show what an image looks like to a user who does not use a vision correcting display. We want to test whether OpenCL can improve the runtime of the prefiltering algorithm on the iPhone and other electronic screens.

Table 6.3: Runtime for Different Pixel Sizes.

Screen Size (Pixels)	Method	Trial 1	Trial 2	Trial 3	Trial 4	Average
80	Without OpenCL	1.208	1.267	1.174	1.177	1.2065
	With OpenCL	1.402	1.401	1.46	1.471	1.4335
160	Without OpenCL	1.242	1.242	1.239	1.222	1.23625
	With OpenCL	1.441	1.436	1.432	1.437	1.4365
240	Without OpenCL	1.351	1.342	1.344	1.346	1.34575
	With OpenCL	1.407	1.448	1.424	1.419	1.4245
320	Without OpenCL	1.458	1.437	1.454	1.433	1.4455
	With OpenCL	1.412	1.428	1.464	1.425	1.43225
400	Without OpenCL	1.567	1.557	1.618	1.597	1.58475
	With OpenCL	1.414	1.445	1.452	1.41	1.43025

Table 6.3 shows the run time of the prefiltering algorithm for different screen sizes (80 pixels, 160 pixels, 240 pixels, 320 pixels, and 400 that apply or do not apply the OpenCL modifications. We used four trials for each screen size and method combination in order to prevent inconsistencies and took the mean of all trials.

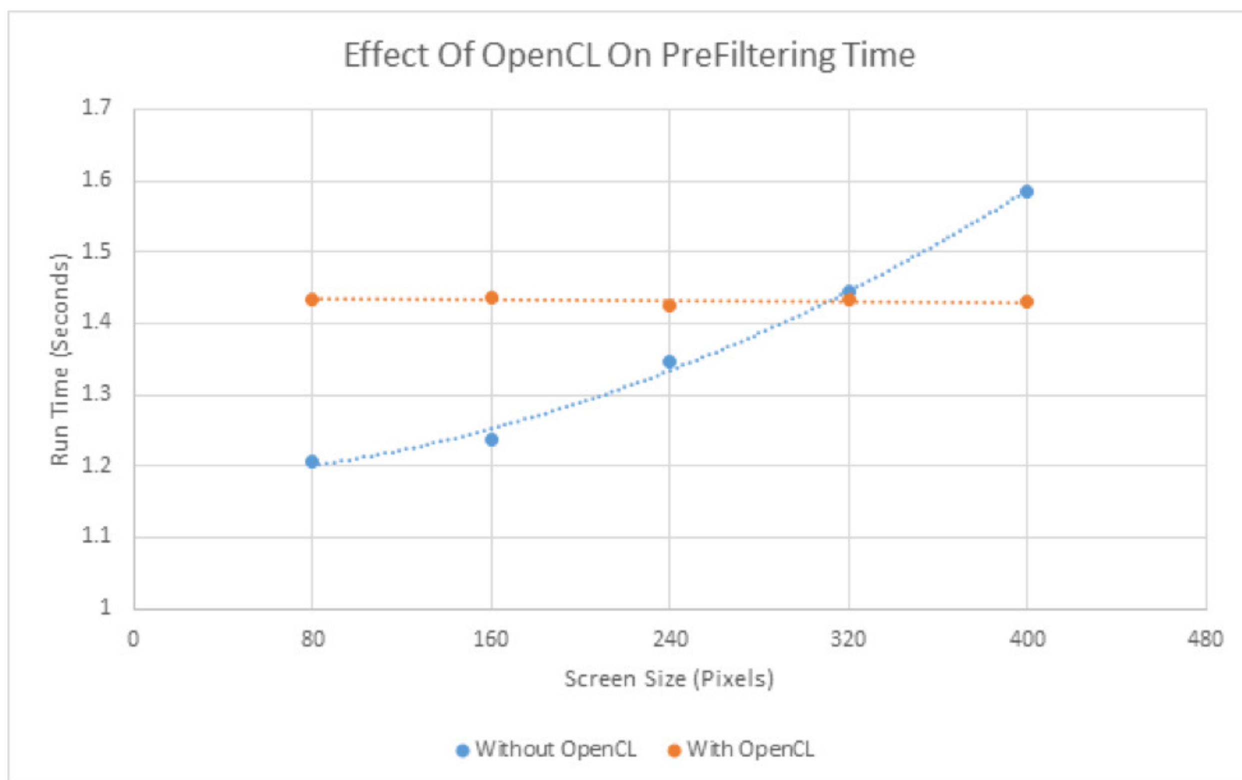

Figure 6.6: Effect of OpenCL on prefiltering time.

Figure 6.6 shows that without OpenCL, prefiltering time runs in quadratic ($O(n^2)$ time) with respect to the screen size. With OpenCL, prefiltering times runs in constant time of around 1.43 seconds regardless of screen size. For screen sizes smaller than 300 pixels, running prefiltering normally is faster. For screen sizes larger than 300 pixels, running prefiltering with OpenCL is faster.

For small screen sizes, running prefiltering normally is faster, but for large screen sizes, using OpenCL is faster. The overhead of OpenCL may contribute to the slower runtimes for small screen sizes. If we extrapolate the run time values of the prefiltering algorithm with and without OpenCL, we would find that running the prefiltering code with OpenCL would be much faster on a computer or television than without OpenCL.

6.4 Limitations and Future Improvement

Limitations

For simulation general limitations:

- In our current simulation, we only model light as light rays. The propagation of light through space is usually sufficiently modeled as rays. However, this is a simplification. Light is electromagnetic radiation and has both wave and particle properties. To generalize the simulation, we should account for the wave properties.
- Currently diffraction is being ignored. The effect on image quality due to diffraction need to be furthur evaluated.

For OpenCL specific limitations:

- The biggest drawback for our OpenCL implementation is that we are unable to use OOP practices. There are “structs” in that can be used to group variables however that is the extent of OOP.

Future Improvement

First, our OpenCL simulation is only able to model focus/defocus. To model the result for higher order aberrations, we need to incorporate the wavefront model for aberrations into our simulation.

Second, the display setup results in very noticeable RGB effects (periodic R, G, and B stripes in the perceived image). We believe that these effects can be minimized if a different RGB pixel subarrangement were used by the sensor pixels. To run experiments to determine the optimal RGB arrangements we would need to modify the simulation, which as of now implements a fixed configurations (three vertical bars corresponding to R, G, B).

Chapter 7

Evaluation

7.1 Simulation Results

We set up our simulation with the camera focused at 400 mm, while placing the display at 350mm. In this case, presbyopia is simulated. We prefiltered our images with respect to this setting. In Figure 7.1, final results are collected from the simulation program.

The original display assumes that there is no aberration compensation to enhance the image. In the pinhole array display, light is filtered through square shaped pinholes. In the microlens array display, light travels through small microlenses organized into a matrix structure. The goal of the simulation is to produce a resulting image to appear as close as possible to the original image in Figure 7.1, taking into account resolution, contrast, and brightness.

7.2 Visual Metrics

Two visual metrics are used here to evaluate the results. DRIM [3] is used to evaluate the contrast, and HDR-VDP-2 [14] is used to evaluate the overall visual quality.

In Figure 7.2, both the pinhole array display and the microlens array display show an improvement over the original display in terms of contrast. The microlens array display has better visual quality according to HDR-VDP-2 in Figure 7.3 and the pinhole array display has better contrast.

The microlens array display is brighter than the pinhole array display because the pinhole filters out a significant amount of light. In Figure 7.1, the brightness of the pinhole array images has already been increased by 25 times to better compare the contrast and visual quality. Therefore, in reality, the microlens array image will be much brighter.

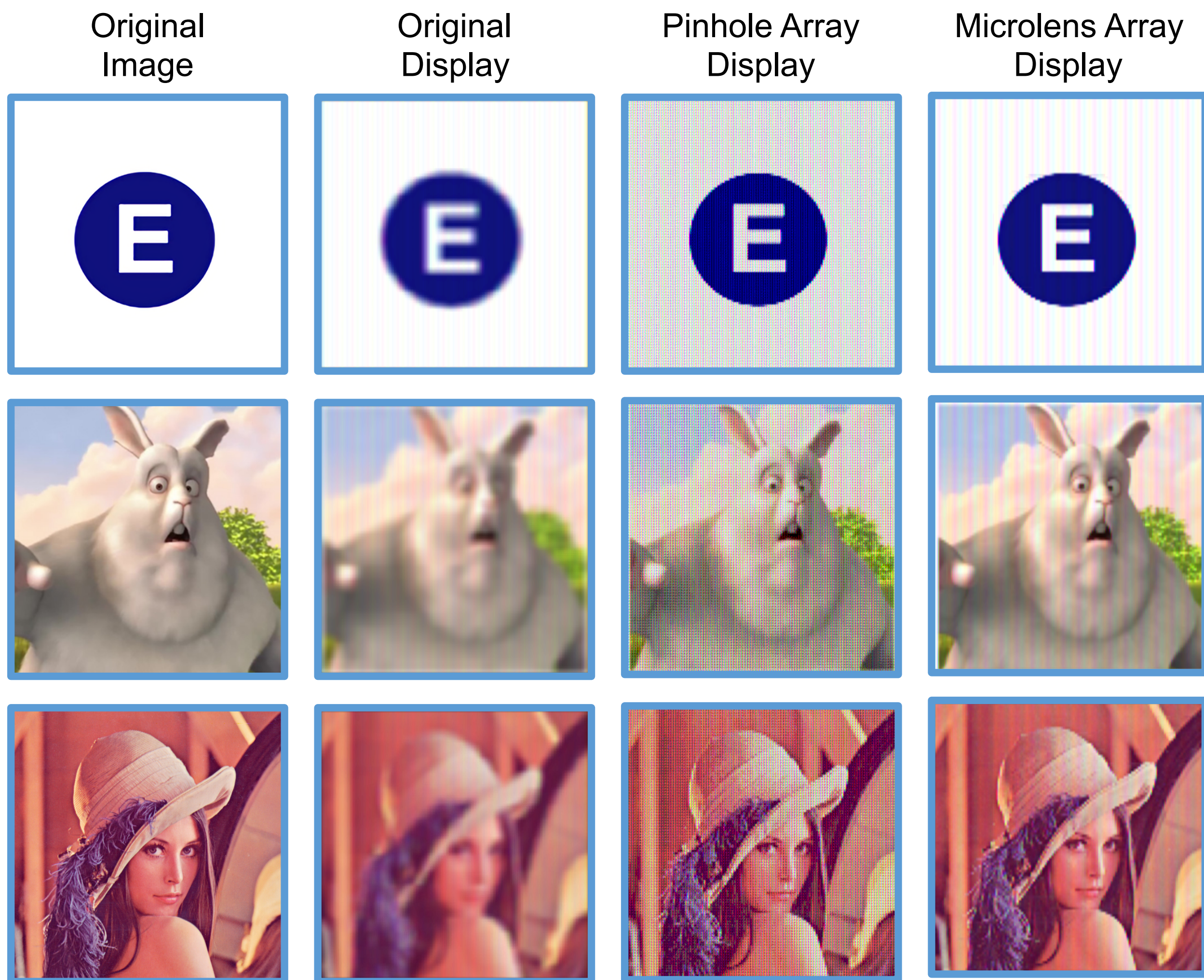

Figure 7.1: Simulation result.

7.3 RGB Effect

The artifact of color stripes on the simulated result is called the RGB effect, because red, green, and blue sub-pixels are not located in the same position. Some preliminary studies have been conducted on different screen RGB sub-pixel arrangements. These studies show that different arrangements will lead to different sharpness and contrast when the vision correcting display is viewed. Future studies are needed to explore the relationship and ideally determine the best arrangement for our purposes.

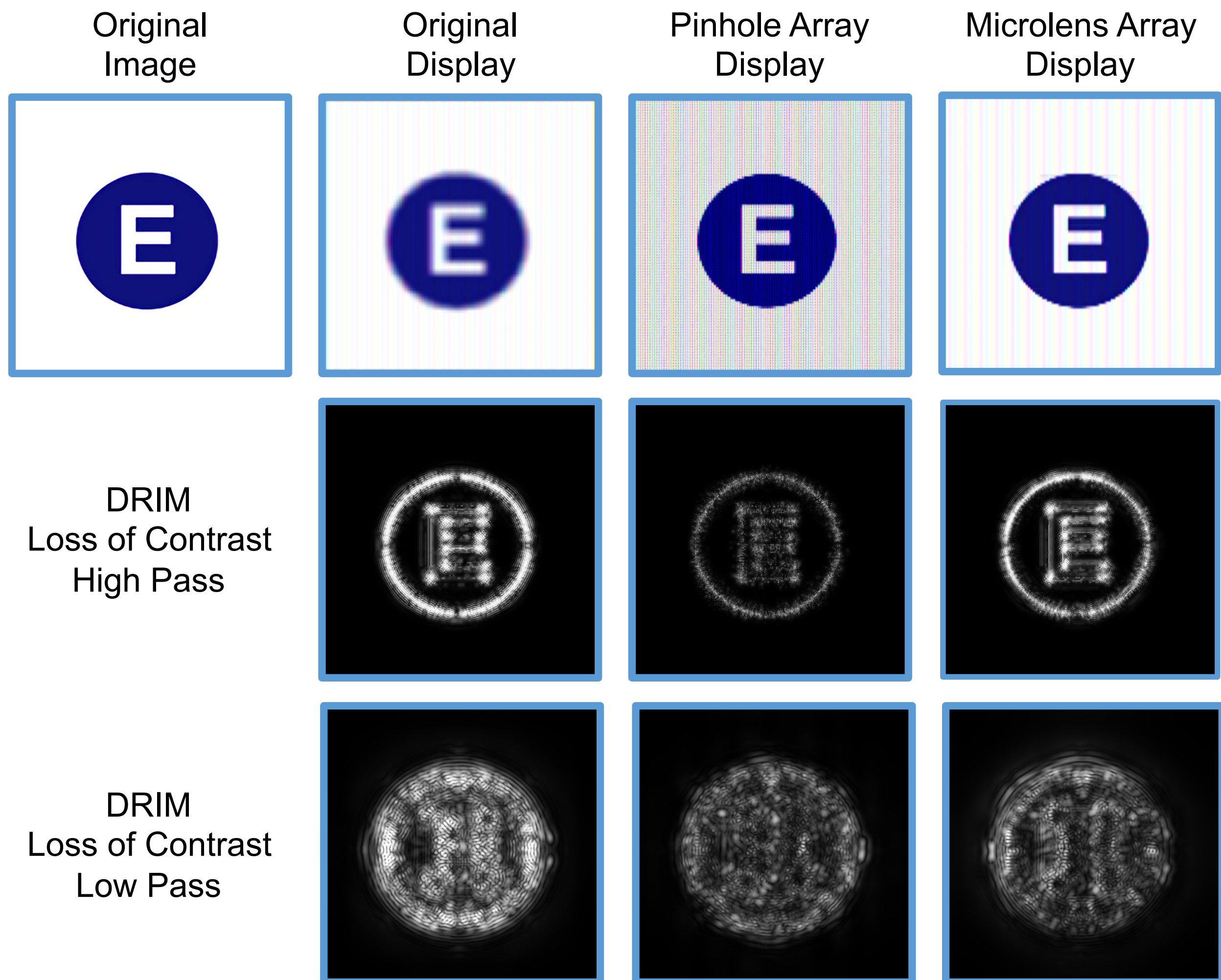

Figure 7.2: DRIM results for the letter E.

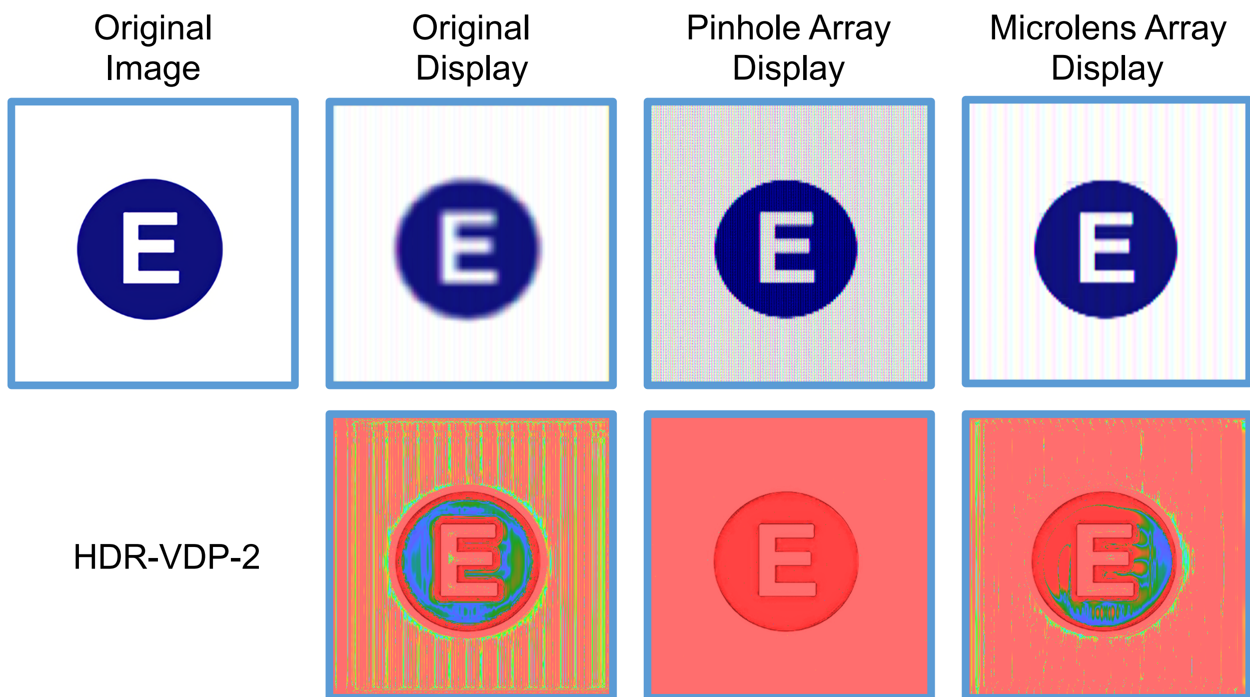

Figure 7.3: HDR-VDP2 results for the letter E.

Chapter 8

Future Work

8.1 One Eye to Two Eyes

Our current implementation only accommodates one eye for the vision correcting display. In reality, people have two eyes with slightly different angles towards the screen. In the next step, our algorithm needs to accommodate two eyes and take into account the distance between the eyes, and even possible different aberrations.

8.2 Larger Display

Our current prototype is focused on mobile devices with relatively small screen size. When considering larger display, for example, computer monitors, many assumptions may not stand. Larger screens have another set of constrains as well. For example, larger displays do not need a thin mask since the thickness does not matter. A prototype could be built to better study the feasibility of the vision correcting display on a larger screen.

8.3 Eye Tracking

For our current implementation, we assume the eye is perfectly perpendicular to the display plane. However, this is not true in reality. Some eye tracking functionalities could be added to evaluate the position of the eye relative to the display, so that a more precise projective relationship could be established.

Chapter 9

Conclusion

In conclusion, results presented in this paper show that the visual correcting display improves the quality of vision for people with eye aberrations. People nowadays spend significant amount of time interacting with digital devices. Therefore, the vision correcting display may have a profound impact on their life. The three major software improvements presented in this report bring the vision correcting technology to the real-time era, making the technology a lot more practical. The major hardware improvement proposed in this report will significantly improve the user experience. In the future, the vision correcting display will become a practical alternative to eyeglasses and contact lenses and play an important role in people's everyday life.

Bibliography

[1] Miguel. Alonso Jr. and Armando B. Barreto. “Pre-compensation for high-order aberrations of the human eye using on-screen image deconvolution”. In: *IEEE Engineering in Medicine and Biology Society*. Vol. 1. 2003, pp. 556–559.

[2] Pierre Archand, Eric Pite, Herve Guillemet, and Loic Trocme. *Systems and Methods for Rendering a Display to Compensate for a Viewer’s Visual Impairment*. International Patent Application PCT/US2011/039993. 2011.

[3] Tunç Ozan Aydin, Karol Mantiuk Rafałand Myszkowski, and Hans-Peter Seidel. “Dynamic Range Independent Image Quality Assessment”. In: *ACM Trans. Graph.* 27.3 (Aug. 2008), 69:1–69:10. ISSN: 0730-0301. DOI: `10.1145/1360612.1360668`. URL: `http://doi.acm.org/10.1145/1360612.1360668`.

[4] Brian A. Barsky. “Vision-realistic rendering: simulation of the scanned foveal image from wavefront data of human subjects”. In: *Applied Perception in Graphics and Visualization*. 2004, pp. 73–81. DOI: `10.1145/1012551.1012564`. URL: `http://doi.acm.org/10.1145/1012551.1012564`.

[5] Brian A Barsky, Fu-Chung Huang, Douglas Lanman, Gordon Wetzstein, and Ramesh Raskar. “Vision Correcting Displays Based on Inverse Blurring and Aberration Compensation”. In: *Computer Vision-ECCV 2014 Workshops*. Springer. 2014, pp. 524–538.

[6] Arnaldo Dias-Santos, Rita Rosa, Joana Ferreira, Joapoundso P. Cunha, Cristina Brito, Ana Paixapoundso, and Alcina Toscano. “Higher order aberrations in amblyopic children and their role in refractory amblyopia”. en. In: *Revista Brasileira de Oftalmologia* 73 (Dec. 2014), pp. 358–362. ISSN: 0034-7280. URL: `http://www.scielo.br/scielo.php?script=sci_arttext&pid=S0034-72802014000600358&nrm=iso`.

[7] Max Grosse, Gordon Wetzstein, Anselm Grundhöfer, and Oliver Bimber. “Coded aperture projection”. In: *ACM Trans. Graph.* (3 2010). ISSN: 0730-0301. DOI: `http://doi.acm.org/10.1145/1805964.1805966`. URL: `http://doi.acm.org/10.1145/1805964.1805966`.

[8] Fu-Chung Huang. “A Computational Light Field Display for Correcting Visual Aberrations”. PhD thesis. EECS Department, University of California, Berkeley, Dec. 2013. URL: `http://www.eecs.berkeley.edu/Pubs/TechRpts/2013/EECS-2013-206.html`.

[9] Fu-Chung Huang and Brian A. Barsky. *A Framework for Aberration Compensated Displays*. Tech. rep. UCB/EECS-2011-162. EECS Department, University of California, Berkeley, Dec. 2011. URL: http://www.eecs.berkeley.edu/Pubs/TechRpts/2011/EECS-2011-162.html.

[10] Fu-Chung Huang, Douglas Lanman, Brian A. Barsky, and Ramesh Raskar. "Correcting for optical aberrations using multilayer displays". In: *ACM Trans. Graph.* 31.6 (Nov. 2012), 185:1–185:12. ISSN: 0730-0301. DOI: 10.1145/2366145.2366204. URL: http://doi.acm.org/10.1145/2366145.2366204.

[11] Fu-Chung Huang, Gordon Wetzstein, Brian A Barsky, and Ramesh Raskar. "Eyeglasses-free display: Towards correcting visual aberrations with computational light field displays". In: *ACM Transactions on Graphics (TOG)* 33.4 (2014), p. 59.

[12] Paul L. Kaufman and Albert Alm. *Adler's Physiology of the Eye (Tenth Edition)*. Mosby, 2002.

[13] Dong C. Liu and Jorge Nocedal. "On the Limited Memory BFGS Method for Large Scale Optimization". In: *Math. Program.* 45.3 (Dec. 1989), pp. 503–528. ISSN: 0025-5610. DOI: 10.1007/BF01589116. URL: http://dx.doi.org/10.1007/BF01589116.

[14] Rafat Mantiuk, Kil Joong Kim, Allan G. Rempel, and Wolfgang Heidrich. "HDR-VDP-2: A Calibrated Visual Metric for Visibility and Quality Predictions in All Luminance Conditions". In: *ACM Trans. Graph.* 30.4 (July 2011), 40:1–40:14. ISSN: 0730-0301. DOI: 10.1145/2010324.1964935. URL: http://doi.acm.org/10.1145/2010324.1964935.

[15] NVIDIA. *What is GPU Accelarated Computing?* http://www.nvidia.com/object/what-is-gpu-computing.html/. [Online; accessed 19-April-2016]. 2016.

[16] John I. Yellott and John W. Yellott. "Correcting spurious resolution in defocused images". In: *Proc. SPIE* 6492 (2007).

[17] Frits Zernike. "Beugungstheorie des schneidenverfahrens und seiner verbesserten form, der phasenkontrastmethode". In: *Physica* 1 (May 1934), pp. 689–704. DOI: 10.1016/S0031-8914(34)80259-5.