Towards a Real-Time Vision Correcting Display

Shiyun Xu

Electrical Engineering and Computer Sciences
University of California, Berkeley

Technical Report No. UCB/EECS-2022-160
http://www2.eecs.berkeley.edu/Pubs/TechRpts/2022/EECS-2022-160.html

May 20, 2022

Towards a Real-Time Vision Correcting Display

by

Shiyun Xu

A report submitted in partial satisfaction of the

requirements for the degree of

Master of Science

in

Electrical Engineering and Computer Science

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Professor Brian Barsky, Chair
Professor Boubacar Kanté, Second Reader

Spring 2022

Towards a Real-Time Vision Correcting Display

Abstract

Towards a Real-Time Vision Correcting Display

by

Shiyun Xu

Master of Science in Electrical Engineering and Computer Science

University of California, Berkeley

Professor Brian Barsky, Chair

Visual aberrations, such as farsightedness and nearsightedness, have been a prevalent problem around the world for many years. A common way to correct these problems is wearing eyeglasses, which can be cumbersome or infeasible for certain groups of people. An alternative approach is to utilize specialized hardware to convert a normal display to a light-field display and do computation in a specific way such that the displayed image will become clear when it reaches the user's retina.

Recent studies have shown various computing methods on different hardware. However, most of them focus on the quality of the perceived image. In this report, we want to examine these algorithms from a practical point of view, focusing on the tradeoff between runtime and quality. We first propose our design and implementation of two microlens-based prefiltering algorithms and then speed up a few selected algorithms via parallelism to achieve real-time performance.

Contents

List of Figures

List of Tables

Acknowledgments

I would like to thank my advisor, Professor Brian Barsky, for his mentorship and guidance during my three years in the lab. I would also like to thank Professor Boubacar Kanté for reviewing this report and providing valuable feedback. I also want to express my gratitude towards all of my collaborators. VCD 3D Group: Yizhen Ding, Jacob Holesinger, Sundi Xiao, Xiaoyu "Evelyn" Yang, Zhuoming Zhang, and Yaying Zhao. Microlens Array Group: Angela Chen, Chun-Ning Tsao, and Eric Ying. Parallelization Group: Shuwei "Waller" Li, Yu Long, Haohua Lyu, Kaiwen "Kaia" Yu, and Siyu Zhang. Finally, I would like to thank my parents for their continuous support throughout my years abroad.

Chapter 1

Introduction

1.1 Vision Correcting Display

Visual aberrations have been a prevalent problem around the world for many years, affecting people of all ages, genders, and races. For example, there are about 80 million children with myopia, or nearsightedness, in the world [10]. The traditional method to mitigate such aberration is to wear a pair of corrective lenses; however, it can be cumbersome or infeasible for certain groups of people. It can even be impossible for people with higher-order aberrations.

Vision correcting display (VCD) uses an alternative approach to correct such aberrations. The idea is to utilize specialized hardware to convert a normal display to a light-field display and do computation in a specific way such that the displayed image will become clear when it reaches the user's retina.

1.2 Parallelism

Historically, computers execute programs in a serial fashion. In recent years, as Moore's Law diminishes, we can no longer rely on the increase of CPU frequency over time to reduce the runtime of our program. Meanwhile, manufacturers have been increasing the number of cores per CPU in not only PCs but also mobile devices. Therefore, to speed up programs, it is paramount to adapt them to be executed on multiple cores concurrently.

For VCD to be practical, it is essential to make short the processing time of the algorithms of converting an image (or a frame of a video) to be displayed on a VCD. Our approach is to take advantage of the prevalent multi-core CPUs and parallelize these algorithms.

1.3 Organization of This Report

In Chapter 2, we go over some background information and related work in the field of vision correcting display. In Chapter 3, we examine some existing algorithms and propose a few microlens-based algorithms. We then conduct some experiments on the runtime and quality of them and compare the results with other existing algorithms in Chapter 4. In Chapter 5, we present our approaches to optimize a handful of methods to allow processing in real time via algorithm modification and parallelism. Finally, in Chapter 6, we perform various experiments to examine the performance of parallelized algorithms on different platforms.

Chapter 2

Background

In this chapter, we introduce some concepts as well as prior work in the field of vision science and parallel computing to give a context of the vision correcting display project and facilitate understanding later chapters of this report.

2.1 Visual Aberrations of the Eye

Diopter

Diopter is a unit for measuring the refractive power of a lens. For our purposes, it refers to the ability of the lens of a human eye to bend the light rays passing through it. Diopter is equal to the reciprocal of the eye's focal length.

Typical relaxed human eyes have 60 diopters [11]. As diopters increase, the focal length decreases, and light rays passing through the lens will converge to a point closer to the lens. As diopters decrease, the focal length increases, and light rays will converge to a point further away from the lens. A normal human eye adjusts its diopter dynamically depending on the distance between the focused object and the eye.

Myopia and Presbyopia

If a human eye fails to make the light rays focus on the retina, we call such a condition as a visual aberration. The two most common ones are myopia, commonly known as nearsightedness, and presbyopia, commonly known as farsightedness.

In myopia, the human eye has an abnormally high diopter when relaxed, causing it to lose the ability to focus on far objects. The light rays will converge to a point before the retina and creating circular blur. In presbyopia, the human eye loses the ability to accommodate when focusing on close objects, thus losing diopters.

Nearsightedness	Far Point (mm)	Farsightedness	Near Point (mm)
-2D (Myopia)	500	+1D (Presbyopia)	330
-3D (Myopia)	330	+2D (Presbyopia)	500
-5D (Myopia)	200	+3D (Presbyopia)	1000

Table 2.1: Diopters and corresponding far or near points

For the purpose of simulating these aberrations, a single plane (or point) is used to separate the objects in focus versus the objects out of focus. In the case of myopia, there exists a **far plane** such that objects further away are out of focus. In the case of presbyopia, there exists a **near plane** such that objects closer are out of focus. Table 2.1 shows the relationship between diopters and their corresponding far or near points [8].

Higher-Order Aberrations

In addition to the relatively common myopia and presbyopia, other types of visual aberrations exist. Higher-order aberrations refer to those aberrations that have a third or higher Zernike term [5]. Zernike polynomials are used to describe the aberrations of the wavefront of a lens.

In this report, however, we focus on myopia and presbyopia, and more considerations and modifications are needed for the algorithms to work with high-order aberrations.

2.2 Vision Correcting Display

In a conventional display, each pixel emits light in all directions, and we can only change its color. In contrast, a light field display can control both the color and the direction of the light ray emitting from each pixel. The additional direction information is vital to our implementation of vision correcting display.

Previous research groups used mostly pinhole-based light field displays. A pinhole mask is a sheet with arrays of openings. It covers a conventional screen such that the pixels below each pinhole can only go through the pinhole in a certain direction, creating a light field. A microlens array is a plane made up of arrays of tiny lenslets. The pixels below each lenslet will be bent in a certain direction, creating a light field.

The microlens array allows more light to pass through but is significantly more expensive than pinhole mask. Figure 2.2 shows the difference between the two types of display [4].

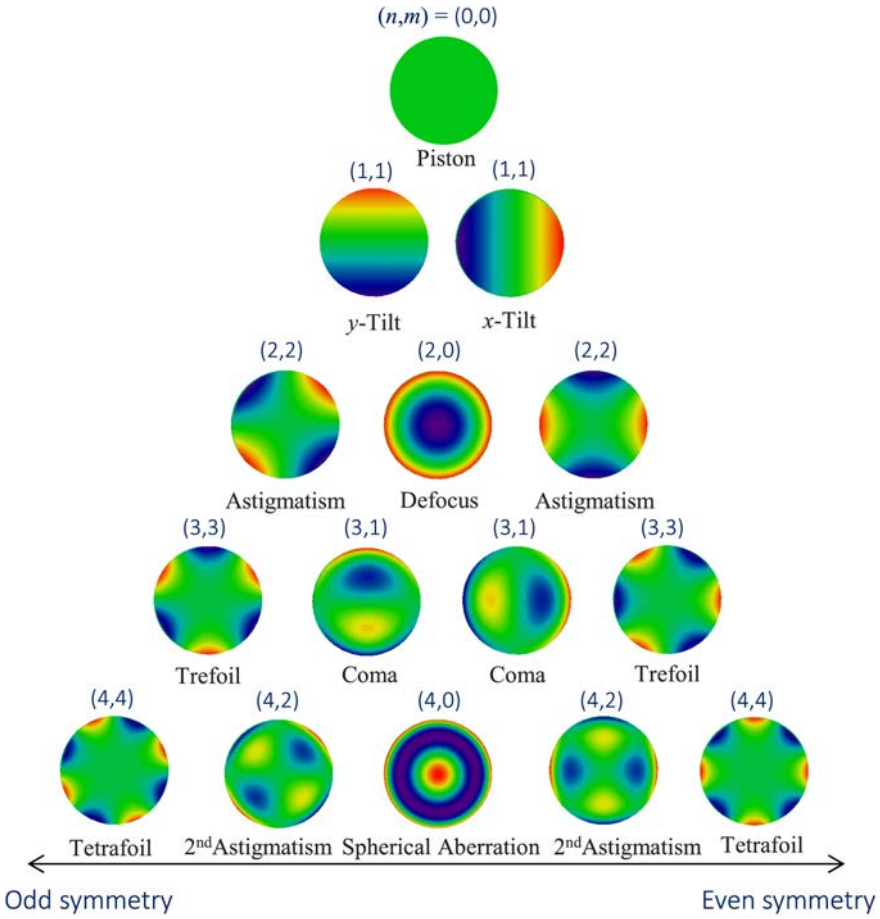


Figure 2.1: The fourth order and lower Zernike polynomials

2.3 Eye Model

The human eye is a very complex biological and optical system. To simplify calculations, we use a simplified eye model (thin lens) to simulate the propagation of the light emitted from the display to the retina.

Thin Lens

A thin lens is a lens in which the thickness of the lens is much smaller than its diameter. We consider the thickness negligible when performing ray tracing, which significantly simplifies computation without causing too much error. As a result, images are formed based on the rules shown in Figure 2.3 [7]:

1. A ray parallel to the principal axis is refracted through the focal point.

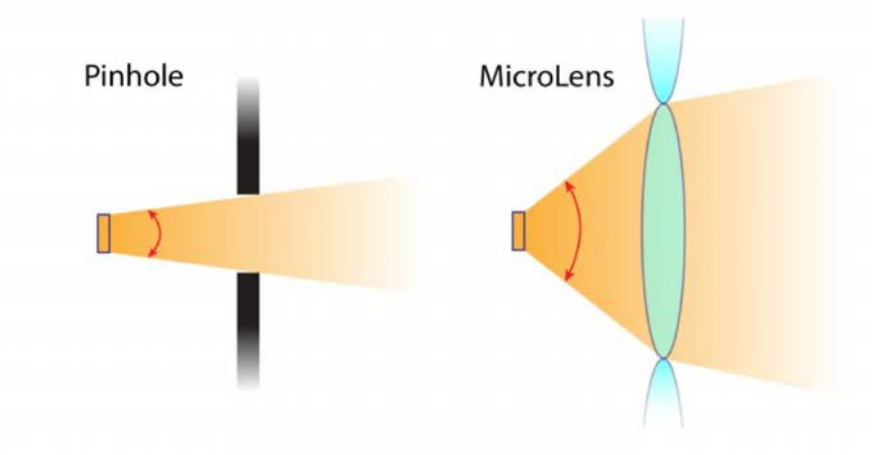


Figure 2.2: llustration of the light from a pixel going through a pinhole and a lenslet

2. A ray through the center of the lens is not refracted.
3. A ray through the focal point is refracted parallel to the principal axis.

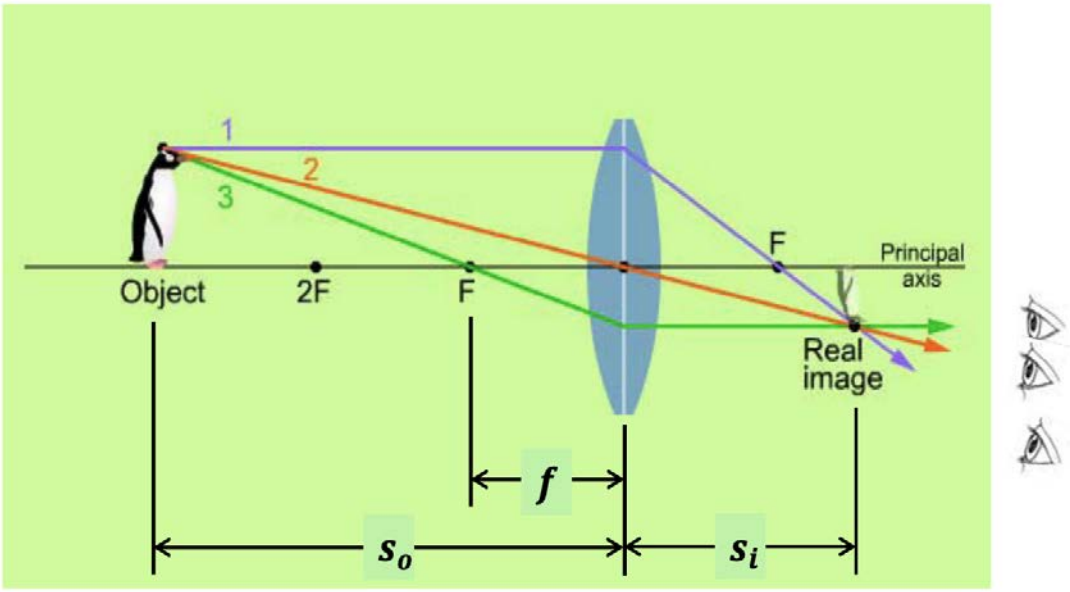


Figure 2.3: Image formation of a thin lens

2.4 Parallel Programming

Parallel programming refers to the approach of solving a problem with multiple processors in parallel instead of a single processor. There are many paradigms to write a parallel program. For example, there are some significant differences between writing a parallel program to be executed on a cluster with shared memory and doing that for one without shared memory. Figure 2.4 shows three commonly seen patterns. In this report, we focus on the first pattern: shared memory with multicores on CPU and GPU. The second one can scale to multiple nodes but is slower than the first one if we only have one node. The third one is not applicable to this project since it requires contiguous read and write of memory.

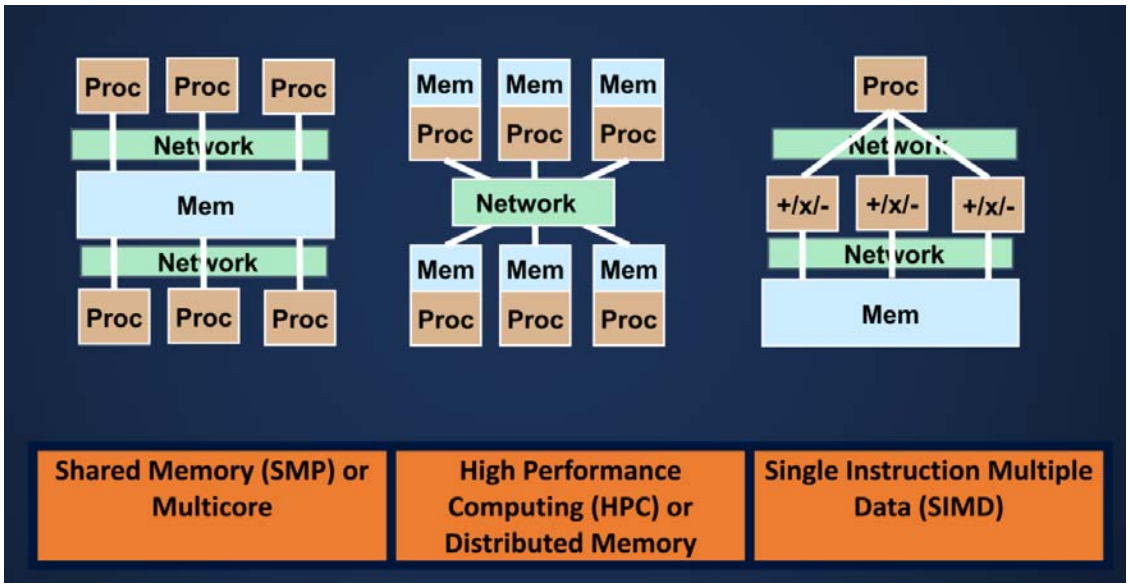


Figure 2.4: Parallel Computers [2]

Shared Memory Parallelism

In shared memory parallelism, processors are all connected to a large shared memory. Typically, each processor reads a task from the shared memory, does some computation, communicates with other processor via shared variable if necessary, and writes back to the shared memory.

GPU

A graphics processing unit, or GPU, can be used for shared memory parallelism. While each thread of a GPU may be slower than a core of a CPU, a GPU typically has a much higher level of parallelism than a CPU. For example, for double precision computation, a Nvidia

Tesla V100 has 163,840 threads while an Intel Skylake only has 80 threads (640 if AVX-512 is included) [1].

Chapter 3

Prefiltering Algorithms

In this chapter[1], we present a survey of the existing algorithms with a focus on its assumptions, runtime, and quality of the resultant images. We then present our design and implementation of the Microlens Naive Algorithm, Microlens Backward Algorithm, and the Microlens Generalized Algorithm. The implementation of the algorithms is based on the codebase in [8]. While Pamplona et al. and Zhao propose an alternative prefiltering algorithm for microlens array-based VCD, we choose to based our algorithms on the one proposed by Zhen for its efficiency and simplicity [12] [14] [15].

3.1 Assumptions

One-to-one Assumption

We assume there exists only one pinhole or microlens that the light from each pixel will go through and reach the eye. This assumption holds true in general. Figure 3.1 shows the proof and the condition of the one-to-one assumption in a microlens-array-based VCD.

Parallel Coaxial Planes Assumption

We assume the display plane and the retina plane are parallel to each other and are coaxial. We believe this assumption simplifies computation and is practical for most use cases. Holesinger shows necessary adaptions if this assumption no longer holds true in [4].

Thin Lens Assumption

As noted in Chapter 2, we use a simplified eye model to simulate the behavior of lights traveling through the eye. We assume the lens of the eye is a thin lens. We believe this simplifies computation without losing too much accuracy.

[1]This chapter is adapted from an unpublished project report by Angela Chen, Chun-Ning Tsao, Shiyun Xu, and Eric Ying in 2021 [3].

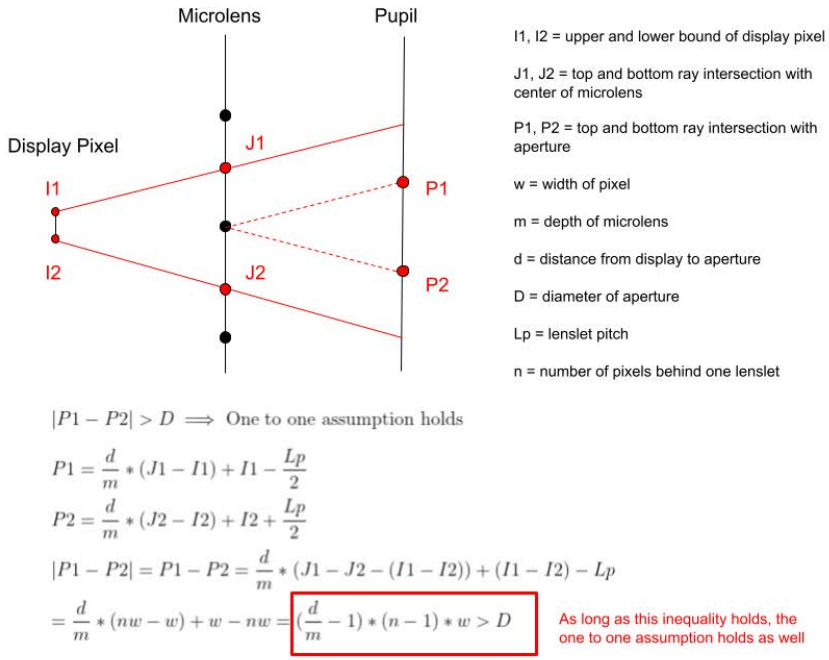


Figure 3.1: Proof of the one-to-one assumption in microlens-array-based VCD

3.2 Workflow

Figure 3.2 shows the workflow of our experiments. Prefiltering refers to the process of converting an image or a frame of a video to be displayed on a VCD so that it becomes clear when reaching the retina of the user. Simulation refers to the process of simulating the image received by the retina when viewing a VCD.

We also have a configuration file that specifies various parameters of the display and the user, such as the width and height of the display, the degree of the visual aberration (in diopters), the distance between the user and the display, etc.

3.3 Definitions

Forward vs Backward Algorithm

The prefiltering process can be categorized as a "forward" approach or "backward" approach, based on whether the rays are traced from the display to the retina or vice versa, respectively.

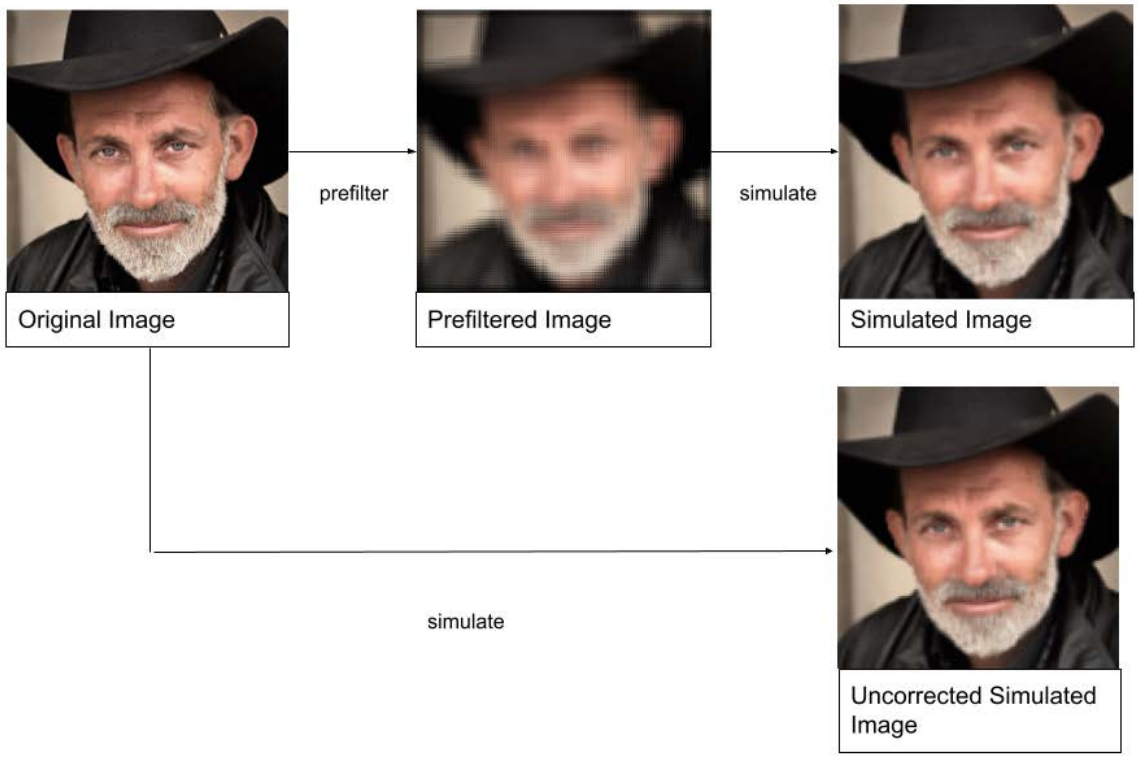


Figure 3.2: Workflow of Vision Correcting Display

Depth of Microlens Array

The depth of a microlens array is defined as the distance between the microlens and the display.

3.4 Previous Algorithms

Forward Optimization

Huang proposed an algorithm in which he formulates the light field projection as a matrix and runs least squares optimization to minimize the difference between the prefiltered image and the original image [6]. Al-Kazily et al. proposed a similar algorithm by changing the direction when constructing the matrix from backward to forward [8].

As solving a least squares optimization problem takes more than linear time, we choose to focus on other algorithms for real time performance. In [6], Huang's algorithm took about 3 minutes to construct a matrix and 20 seconds to solve the optimization problem.

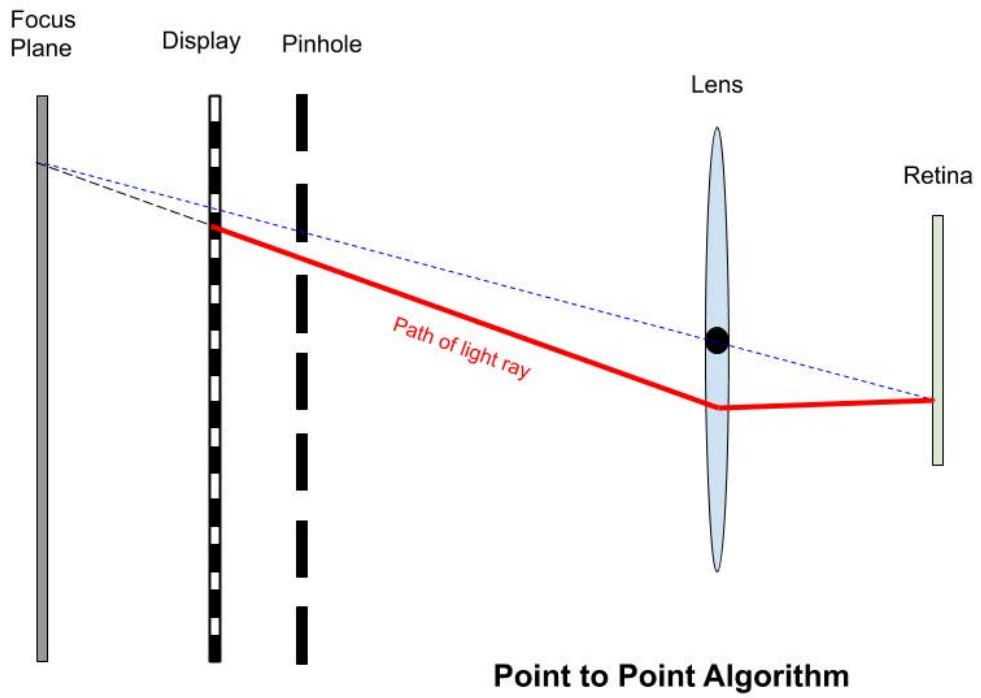


Figure 3.3: Illustration of the point to point algorithm [13]

Point-to-point Algorithm

The point-to-point algorithm is a forward algorithm. We first place the ideal image on the retina. For each pixel on the screen, we find its corresponding pinhole (note that the one-to-one assumption allows us to simply find the closest pinhole) and trace its corresponding place on the retina. We then do bilinear interpolation and assign the color to the pixel on the screen. The time complexity is $O(N)$ where N is the number of pixels on the display. Variants of the point to point algorithm can be found in [15].

Pinhole Backward Algorithm

The pinhole backward algorithm is also a ray-tracing-based algorithm but traces light rays from the retina to the display. It takes $O(N^2)$ time to generate the prefiltered image [15].

Pinhole Simulation Algorithm

The pinhole simulation algorithm uses the same idea as the pinhole backward algorithm. We place the prefiltered image on the display, and compute the color for each pixel on the retina. We then output the image on the retina.

3.5 Microlens Naive Algorithm

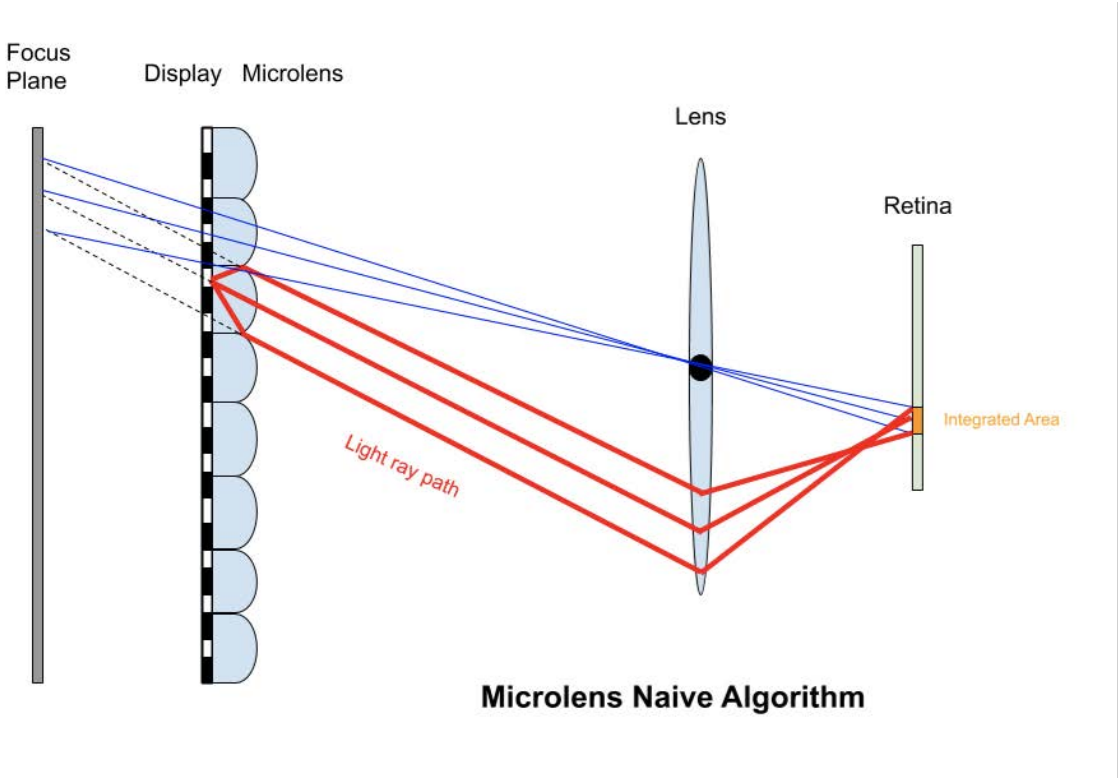


Figure 3.4: Illustration of the microlens naive algorithm

The first algorithm we propose is the microlens naive algorithm. It is a forward algorithm. Similar to the pinhole based forward algorithms (point to point), this algorithm traces rays from a display pixel to the retina and maps the colors of the retina pixels back to the display pixel. The microlens naive algorithm assumes that the focal length of each individual lenslet in the microlens array is equal to the depth of the microlens array (i.e. the distance). This assumption allows us to assume that each lenslet in the microlens array outputs a bundle of parallel light rays (derived from thin lens assumption). Thus, only the top and bottom bounding rays need to be traced in order to encompass the entire bundle of light rays, simplifying ray tracing calculations greatly and reducing the time to execute the algorithm.

Figure 3.4 shows the process of the microlens naive algorithm. Note that the rays exiting the microlens mask are parallel, simplifying the number of rays that need to be traced to just the top and bottom bounding rays. The bounded area on the retina can then be uniformly integrated to return an averaged color.

The pseudocode is as follows:

```
for each display_point:
  find the lenslet it is behind, determine lenslet center
  get ray_a from display_point to center of lenslet
  trace ray_a back to to the focus plane
    call the intersection with focus plane inter_a
  get top and bottom rays on the focus plane (±lenslet_pitch / 2 around inter_a)
    with the same direction as ray_a
  trace the top and bottom rays to an area on retina,
    similar to methodology in point_to_area and area_to_area
  integrate the pixels of ideal image at the area,
    get averaged color
  assign color back to the display_point
```

In addition, we verified the one to one assumption for the microlens forward algorithm. The one to one assumption states that light rays from one display pixel can only pass through the pupil through one possible lenslet in the microlens array. If this assumption is broken, then we would have to consider all lenslets in the microlens mask when tracing rays from a particular display pixel, dramatically increasing the computation time of the naive algorithm by a factor n^2 (n is the dimension of the microlens mask). The one to one assumption allows us to circumvent this by selectively choosing one lenslet to project light rays through. Figure 3.1 shows the proof.

3.6 Microlens Backwards Algorithm

After implementing the prefiltering algorithm, we implemented the microlens based simulation. The purpose of the simulation software is to simulate how a user who suffers from a specific case of myopia or presbyopia would perceive a prefiltered image on the vision correcting display. Due to the lack of physical hardware, only a software based simulation can be used to test the quality of the vision correcting display.

Previously, the simulation software targeted only pinhole based vision correcting displays. As a result, we had to extend the simulation software to target microlens based vision correcting displays. Like the pinhole simulation, the microlens based simulation relies on a backwards ray tracing algorithm. For each pixel on the retina (which we can think of as the image that the user "sees"), multiple rays are traced backwards towards the prefiltered display. The colors of the display pixels hit are averaged together and mapped back to the retina. During this process, we make sure not to rely on any underlying assumptions on the user or display parameters, since the simulation must be generalized enough to apply to all possible user and display configurations.

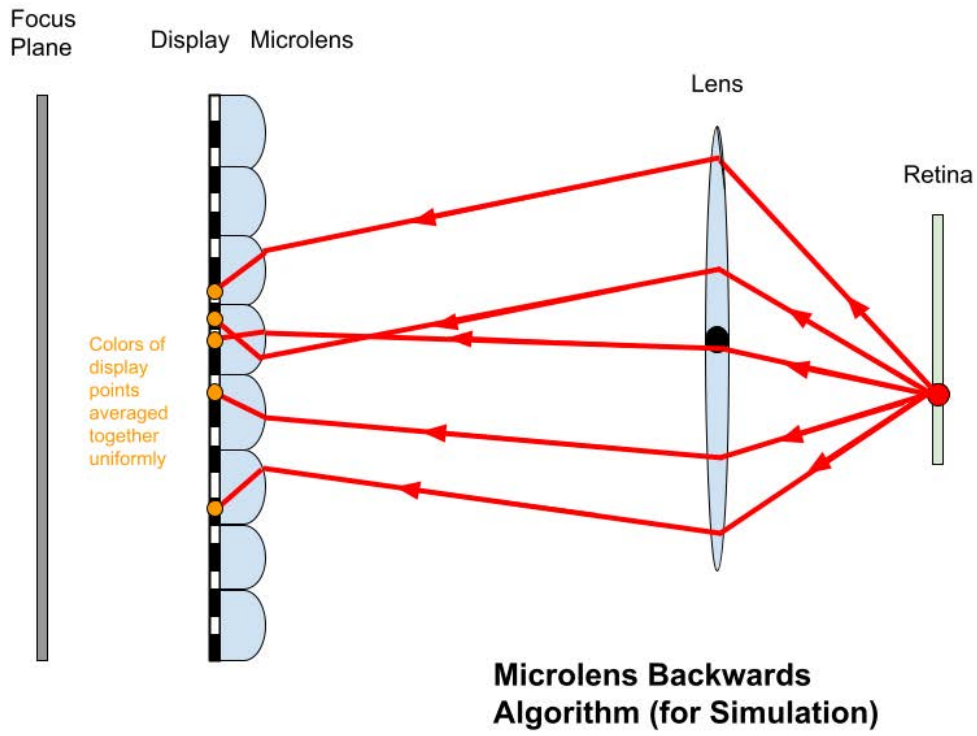


Figure 3.5: Illustration of the microlens backwards algorithm

3.7 Microlens Generalized Algorithm

We propose another microlens forward algorithm that breaks the assumption that the depth of the microlens array is equal to the focal length of lenslets. We believe there are many theoretical benefits to doing this. Firstly, breaking this assumption allows us to reduce the thickness of the display for practical purposes. Secondly, changing the depth to be different from the lenslet focal length allows us to tune the parameters such that bundles of light exiting the display converge on the retina rather than scatter in an area, potentially mitigating focus blur.

This motivates the creation of a new generalized microlens forward algorithm, which relies on a refraction function that maps input light rays hitting a lens to outgoing light rays exiting it. With this function, we can map light rays from the display plane to the microlens layer, refract towards the aperture/pupil layer, and then refract onto the retina. Since we can no longer assume that the bundle of light rays is parallel, we must discretely sample multiple light rays from a display pixel in order to retain as much information as possible. The final runtime of the algorithm is slower than the naive algorithm by a factor of k^2, where k is the

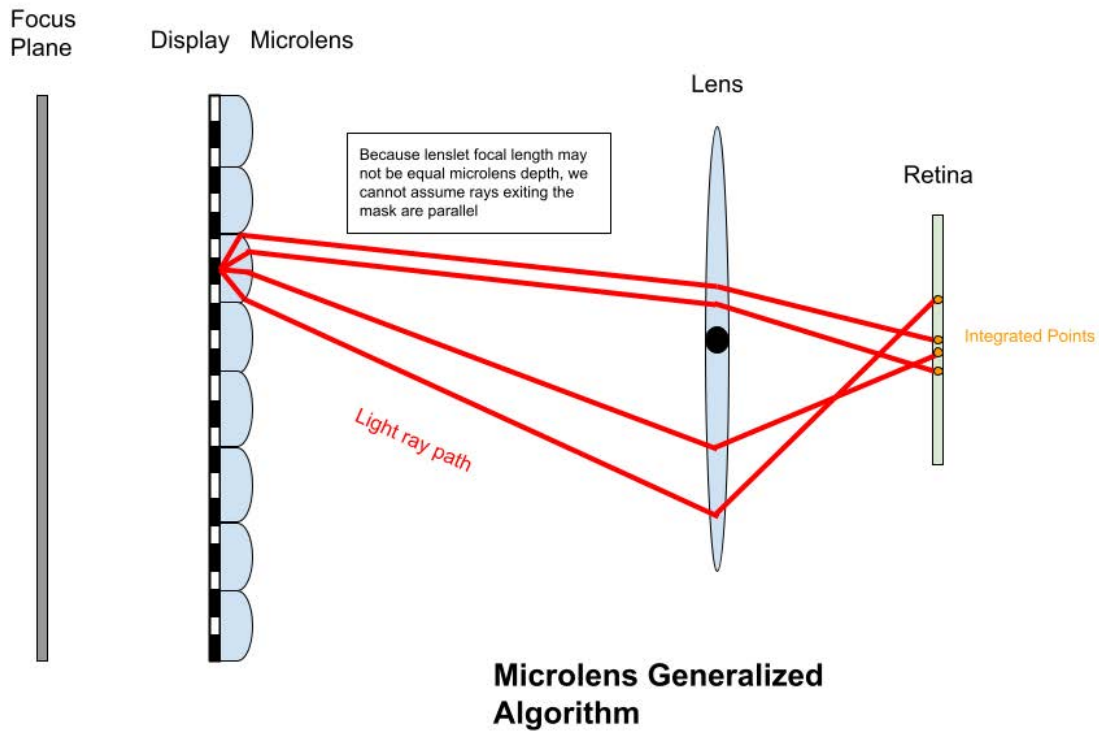


Figure 3.6: Illustration of the microlens generalized algorithm

sampling rate on both the display and microlens layer.

The pseudocode is as follows:

```
for each pixel (display_point) in display:
  select the lenslet that intersects the pixel and the center of the eye
  sample multiple points on the display and lenslet:
    get the ray from display_point to microlens_point
    refract the ray across the lenslet,
      determine where ray hits on pupil plane (aperture_point)
    refract the ray across the pupil,
      determine where the ray hits on sensor plane (sensor_point)
    get the color of the pixel of the image at retina, aggregate into a vector
  perform uniform integration for the colors of pixels in the vector
  set display pixel to that color
```

Figure 3.6 shows the process of the microlens generalized algorithm. Because the rays exiting the graph are no longer parallel, multiple rays must be discretely sampled from each display

pixel. Also note that there are two refractions being performed per light ray, one at the microlens layer and one at the aperture layer.

Chapter 4

Prefiltering Algorithms Experiments

In this chapter[1], After examining several existing algorithms and implementing a few new ones, our next objective is to verify the results of the new algorithms as well as comparing the quality of the images generated by every algorithm. Each algorithm is run on the same set of stock photos while varying a few hyperparameters.

4.1 Metrics

We use the following image quality metrics. The higher the metric is, the better quality the image has.

PSNR

PSNR (or Peak Signal to Noise Ratio), is a simple metric that takes the ratio of the highest signal (in the case of most images, this would be 255 for the maximum RGB value) and the mean squared error noise (calculated by taking the square of the difference between a reference and test image). PSNR is typically measured logarithmically in decibels. It is the least complex one among the three metrics.

SSIM

SSIM (Structural Similarity Index Measure) is a metric that takes into account luminance (mean intensity of the signal), contrast (variance of the signal normalized with luminance), and structural similarity (correlation of the signal normalized with luminance and contrast). These three comparisons are combined together to form the final similarity measure. This metric is more complex than PSNR.

[1]This chapter is adapted from an unpublished project report by Angela Chen, Chun-Ning Tsao, Shiyun Xu, and Eric Ying in 2021 [3].

HDRVDP-2

HDRVDP-2 (High Dynamic Range Visual Difference Predictor) is the most complicated metric of the three. It attempts to model the human visual system as closely as possible. HDRVDP-2 is composed of two metrics, the visibility metric, which measures how likely a human will notice the differences between two images, and the quality metric, which attempts to subjectively gauge how the visual differences between two images will affect the final visual quality perceived by humans. For the purposes of Vision Correcting Displays, we only concern ourselves with the quality metric, since we are interested in how well a user with a specific case of nearsightedness or farsightedness is able to perceive an image on the Vision Correcting Display.

4.2 Results

All following experiments use the 128x128 cowboy reference image below. This image is scaled up to 640x640 on the display during prefiltering and scaled back down to 128x128 on the retina during simulation.

Figure 4.1: Cowboy: Reference image

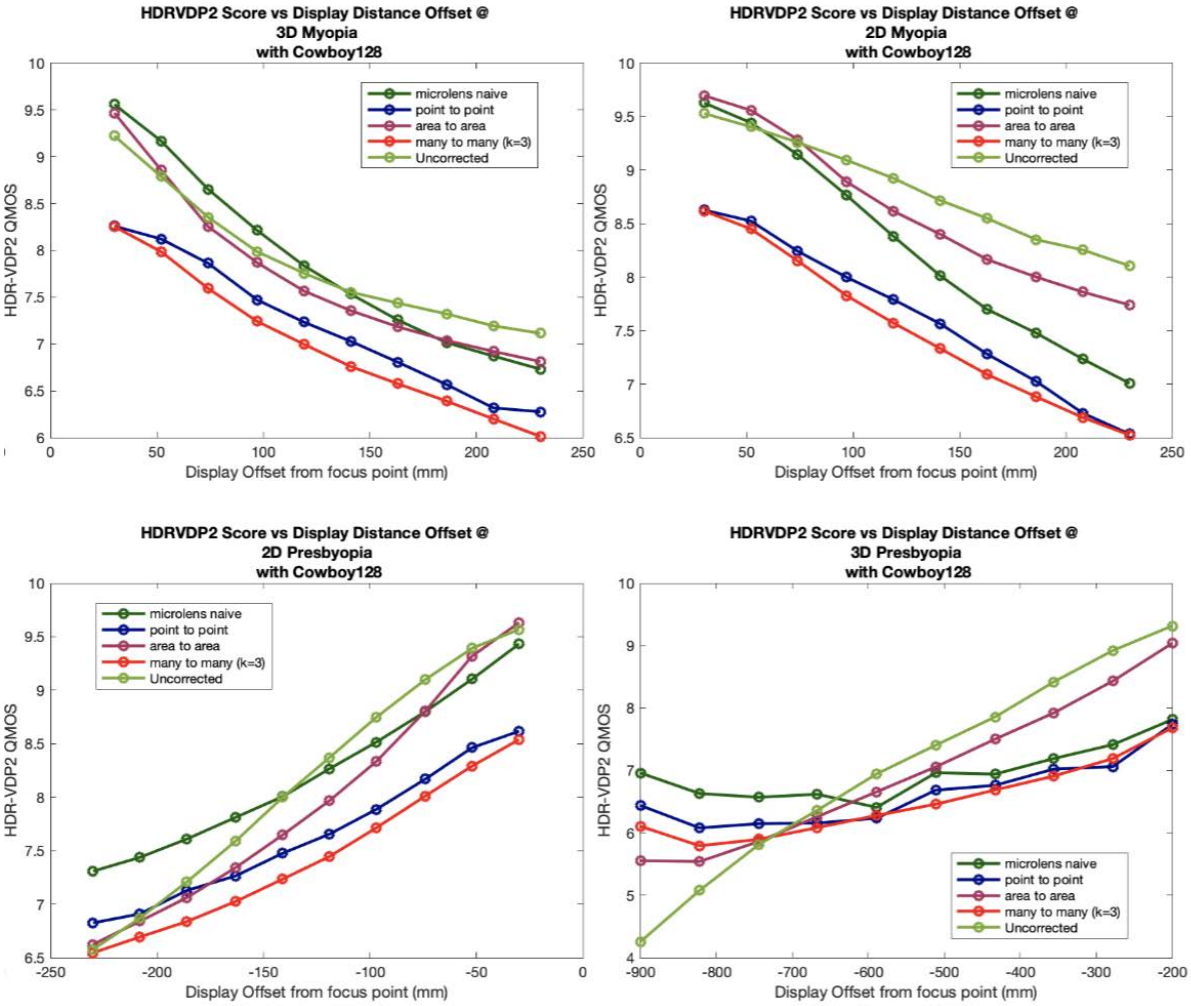


Figure 4.2: Comparison of the microlens naive algorithm with respect to previous pinhole based algorithms

Comparison between Microlens-Based and Pinhole-Based Algorithms

We first compare the quality of the simulated images from the naive microlens algorithm with the simulated images from the pinhole based algorithms (which includes point to point, area to area, and many to many algorithms). Note that the pinhole based forward optimization algorithm, which essentially solves the vision correcting problem posed as a least squares problem, was not included in the test suite due to its slow runtime. In addition, only the HDRVDP-2 results were included in this report, but the PSNR and SSIM results all followed the same pattern as the HDRVDP-2 results.

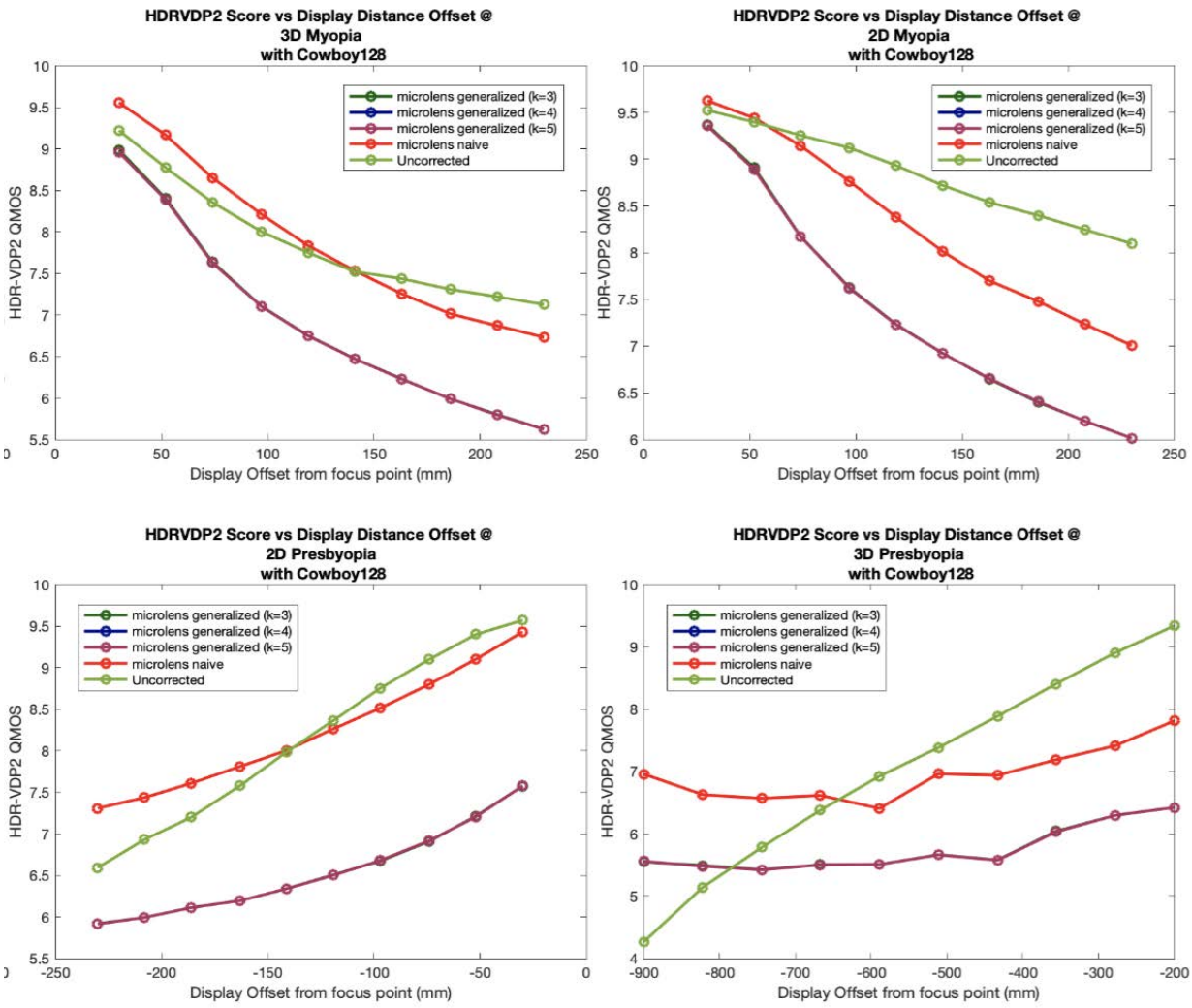


Figure 4.3: Comparison of the naive microlens vs generalized microlens algorithm

In Figure 4.2, the naive algorithm performs fairly well compared to the pinhole based area to area algorithm and the uncorrected algorithm in certain circumstances for the myopic and presbyopic cases.

In the myopic (nearsighted) cases, the microlens naive algorithm performs fairly well with respect to the uncorrected results when the display is closer to the focus plane. In the presbyopic (farsightedness) cases, the microlens naive algorithm performs better when the display is further away from the focus point. One possible explanation regarding this phenomenon is that during prefiltering, the image at the display plane is essentially projected and rescaled onto the focus plane. In myopia, the rescaled image is smaller than the original

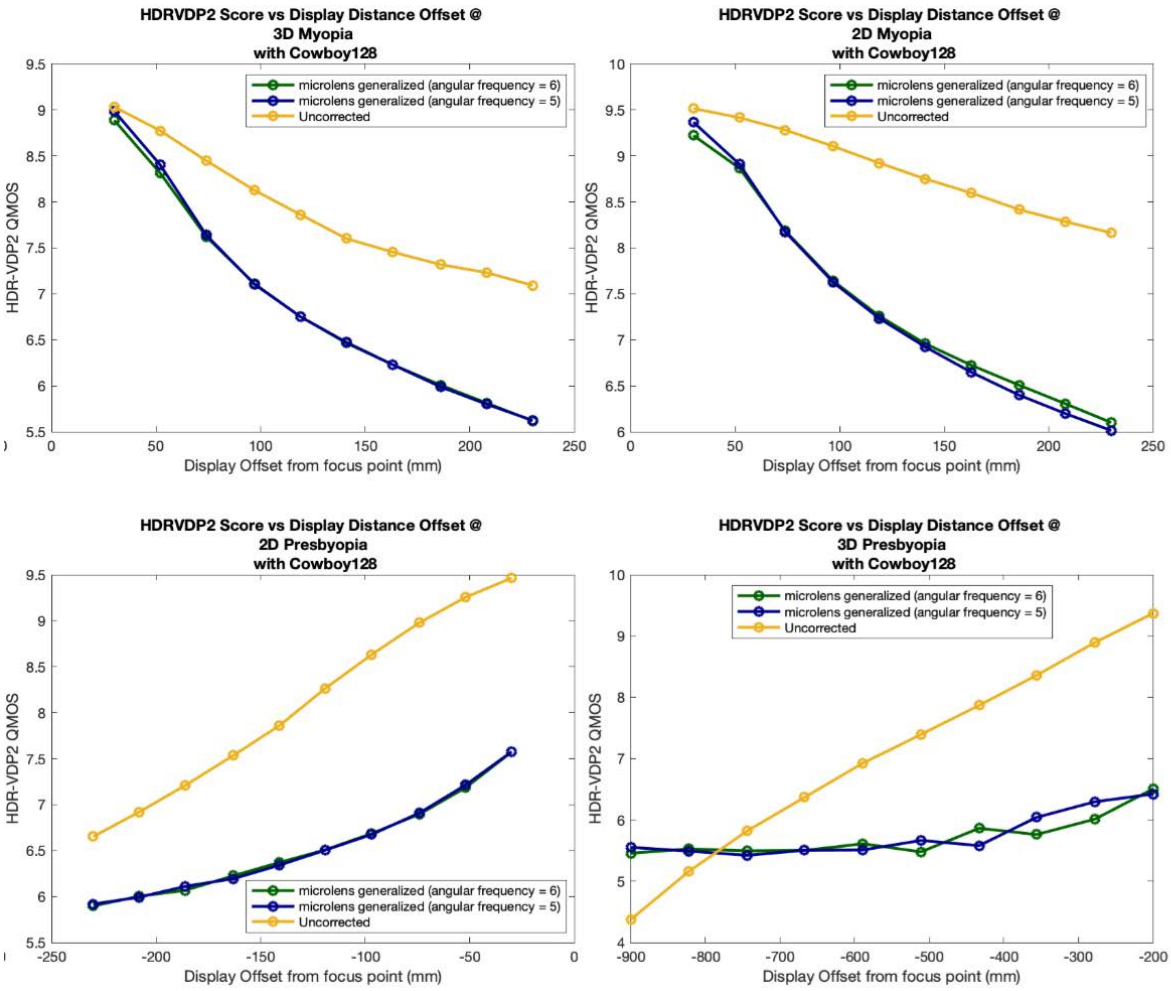


Figure 4.4: Comparison of generalized microlens algorithm with varying angular frequencies (or number of pixels behind each lenslet)

image. Thus, as the display moves further away from the focus plane, the rescaled image at the focus plane gets smaller, and less light rays contribute to the final prefiltering. In presbyopia, the rescaled image is larger. Thus, as the display moves further away from the focus plane, the rescaled image gets larger, and more light rays contribute to the prefiltering.

Comparison between Microlens Naive Algorithm and Microlens Generalized Algorithm

On the other hand, the generalized algorithm performs consistently worse than the naive algorithm. One potential hypothesis that was explored is that the sampling rate is simply not high enough, and there is not enough information to contribute to the prefiltering process.

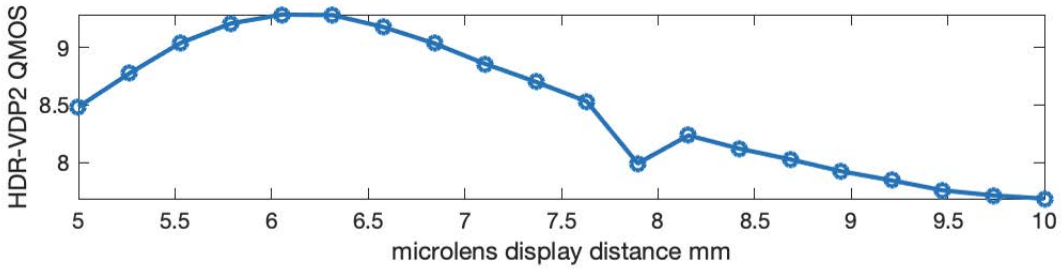


Figure 4.5: HDRVDP-2 quality of simulated image with varying microlens depth. The optimal depth to supposedly minimize focus blur is 7.5mm.

However, higher sampling rates had a very negligible effect on the image quality. Another potential hypothesis was that the angular resolution is not high enough, but varying the angular resolution also had a very negligible effect on the final results. All this suggests that discrete sampling (like in pinhole many to many and microlens generalized algorithms) as a whole performs worse than continuous sampling (like in pinhole area to area and microlens naive algorithms). Further work needs to be done to verify this claim.

In Figure 4.3, the generalized algorithm performs significantly worse than the naive algorithm. In addition, also note that sampling has almost negligible effect on image quality. The curves representing the generalized algorithm with sampling rate k=3,4,5 are stacked on top of one another.

Microlens Generalized Algorithm with Varying Parameters

The final set of experiments performed was to investigate our earlier hypothesis regarding the optimal depth of the microlens mask with respect to the lenslet focal length, distance between display and eye, and eye focus point. This was done by running the image quality metrics while fixing the far point, display distance, and lenslet focal length and varying the microlens depth. From the preliminary results, the quality of the image peaks at 6mm, the same distance as the lenslet focal length. While the current results do not support our hypothesis regarding the optimal microlens depth, they do reveal the robustness of the naive microlens algorithm (since the assumption that is made in the naive algorithm does not seem to adversely affect the final image quality at all).

4.3 Discussion

In conclusion, all experimental results seem to indicate that the naive microlens algorithm performs better than expected, whereas the generalized algorithm performs worse. Of course,

further work needs to be done to verify the correctness of the generalized microlens algorithm and to investigate into the issue of focus blur.

Chapter 5

Parallelizing Prefiltering Algorithms

5.1 Overview

In this chapter[1], we focus on the techniques to reduce the runtime of various prefiltering algorithms. The serial implementation of the algorithms is based on the codebase in [8] with additions and modifications introduced in Chapter 3. Our goal is to enable VCD to achieve real-time performance (24 fps) on a modern display which has a resolution of 1920×1080.

5.2 Methods

Algorithms

We choose to spend most of time on speeding up the point to point algorithm and the microlens naive algorithm. The former one has the fastest serial runtime in pinhole-based algorithms while the latter one shows the most promising results except the forward optimization algorithm.

We decide not to focus on the forward optimization algorithm even though it has the best quality. Its polynomial runtime makes scaling very difficult, and as we have already used a well-tuned optimization problem solver, it is extremely difficult to further reduce the runtime.

Cache Projection Matrix

When the user is reading a book or watching a movie on a screen, the relative location of the user and the display is mostly fixed. This allows us to make an assumption that we only need to compute the projection matrix once (i.e. finding the corresponding pixel on the retina of a pixel on the display). While this does not yield a speed boost when prefiltering

[1]This chapter is done in collaboration with Shuwei Li, Yu Long, Haohua Lyu, Kaiwen Yu, and Siyu Zhang [9].

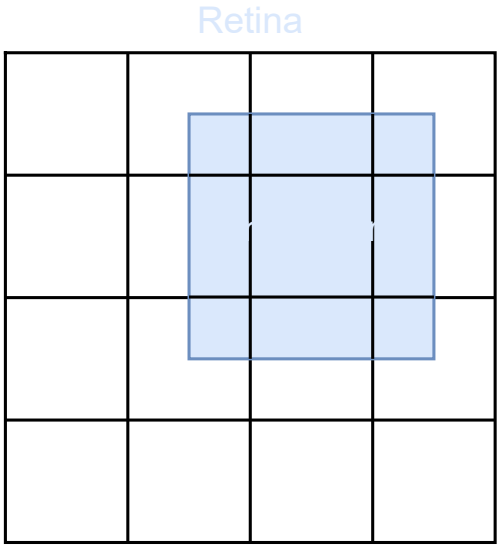


Figure 5.1: Illustration of Color Computation

an image, we observe a noticeable speed up after implementing functions to prefilter videos and live stream from cameras.

A potential improvement is to mitigate small movement of the user by re-computing the projection matrix every few frames. It makes the assumption more practical and improves the overall image quality.

Coarse Integration

Our original implementation of the microlens naive algorithm considers subpixel location when integrating the landing area of the rays on the retina. For each pixel or subpixel in the area, we get the color from the image and weight it by the area of the pixel. For example, in Figure 5.1, the pixel at the corners will be weighted by 0.25.

If the landing area is large, the subpixel-level accuracy is not very important, and we can avoid computing the area explicitly. So we would only sum up all the pixel colors and divide it by 9 in Figure 5.1. We use the built-in `cv::integral` to implement this approach.

OpenMP

Our first approach is to use OpenMP to parallelize the algorithm on CPU. We first read the input image and make it accessible (read-only) to all threads. We then create an empty output image. Next, we assign each pixel to each thread, and each thread does the ray

tracing and compute the color. It finally writes the color to the corresponding place of the output image. Note that there is no thread contention over writing to the same pixel.

GPU

The nature of the algorithms make it natural for us to choose to implement a GPU version. We choose OpenCL over CUDA for its compatibility across multiple platforms. The implementation is similar to the OpenMP in which we assign each GPU thread to a pixel, and the thread does everything to compute the color for the pixel.

After implementing the OpenCL version, we find that we have to convert the input image from BGR format to RGBA format to make it compatible with OpenCL. OpenCL also needs to read the code from the disk and compile it during runtime. These overhead may be overcome by switching to CUDA, which does GPU code compilation during compile time and can process BGR images without conversion.

Chapter 6

Paralleized Prefiltering Algorithms Experiments

In this chapter[1], we introduce the metrics we measure to assess the performance of a few prefiltering algorithms. We then present our results. We perform a series of experiments to assess the runtime and measure how well the algorithms scale as the number of processors increase.

6.1 Metrics

Problem Size

The problem size refers to the data to be processed. In our case, it refers to the number of pixels of the display.

Processor

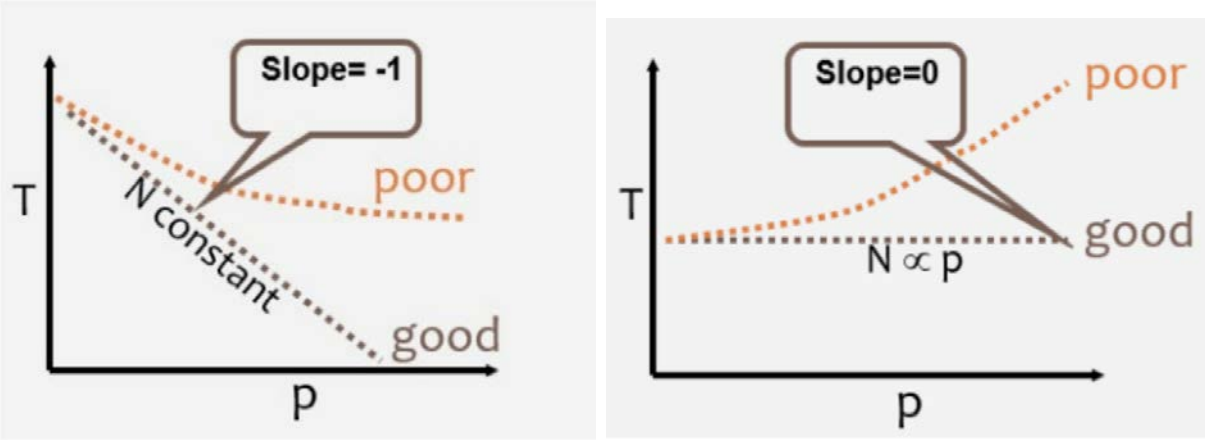
A processor, in our case, means a CPU thread or a GPU thread. In general, it can also mean a process or a node.

Parallel Speedup

Let $t(n, p)$ be the time for p processors to solve a problem of a size of n. The parallel speedup with p processors is defined as

$$\text{Speedup}(p) = \frac{t(n, 1)}{t(n, p)}$$

[1]This chapter is done in collaboration with Shuwei Li, Yu Long, Haohua Lyu, Kaiwen Yu, and Siyu Zhang [9].



(a) Ideal strong scaling graph

(b) Ideal weak scaling graph

Figure 6.1: Strong and weak scaling graphs

Strong Scaling

The strong scaling efficiency evaluates the case in which the problem size remains constant, but the number of processors increases.

$$\text{Efficiency}_{\text{strong}} = \frac{\text{Speedup}(p)}{p}$$

For example, suppose a prefiltering algorithm takes 1 second to process an image of 1000 pixels. If we have 100 processors to process the same image concurrently, we expect the process to get 100 times faster. However, the actual data shows that it only gets 60 times faster. The strong scaling efficiency is thus 60% for this algorithm.

Figure 6.1a shows a typical strong scaling graph.

Weak Scaling

The weak scaling efficiency evaluates the case in which the problem size increases proportionally to the number of processors so the workload per processor stays the same.

$$\text{Efficiency}_{\text{weak}} = \frac{t(n,1)}{t(np,p)}$$

For example, suppose a prefiltering takes 1 second with 1 processor on an image of 1000 pixels. Therefore, we expect it to take 1 second with 2 processors on an image with 2000 pixels. However, the actual data shows that it really takes 4 seconds with 2 processors on an image with 2000 pixels. Thus, the weak scaling efficiency is 25%.

Figure 6.1b shows a typical weak scaling graph.

Serial Slowdown

For an $O(n)$ algorithm, ideally, the runtime should increase linearly $O(n)$ as the problem size increases. In reality, due to the presence of various overheads, the runtime increases at the rate of $O(n^{1+\epsilon})$. This ϵ is called serial slowdown.

6.2 Results

To assess the performance of various prefiltering algorithms in the real world, we measure the runtime on a MacBook Pro that is equipped with an Intel i9-9880H (8 cores) and an integrated graphics card (Intel HD Graphics 630).

We also perform a series of experiments to measure how well the algorithms scale as the number of cores increases. The scaling experiments are conducted on a single KNL node on Cori at NERSC (National Energy Research Scientific Computing Center). A single KNL node has an Intel Xeon Phi Processor 7250 which has 68 physical cores and runs at 1.4 GHz.

For the GPU experiments, we conduct the experiments on an integrated GPU (Intel HD Graphics 630) which can be normally found on a consumer laptop. We also perform the same experiments on a single dedicated GPU, Nvidia Tesla V100, on Bridges-2 to assess the potential of the algorithms.

All experiments are performed on a set of images of varying sizes: $64 \times 64, 91, 128 \times 128, 181 \times 181, 256 \times 256, 362 \times 362, 512 \times 512, 724 \times 724, 1024 \times 1024, 1448 \times 1448, 2048 \times 2048, 2896 \times 2896, 4096 \times 4096$.

OpenMP Serial Slowdown

We run the OpenMP version of the microlens naive algorithm and the point to point algorithm on a single node with 68 cores and set the number of threads to 68. It takes 30 ms for the microlens naive algorithm and 27ms for the point to point algorithm to prefilter an image with a size of 1448×1448. Figure 6.2 shows that the runtime increases roughly linearly as the number of pixels increases.

OpenMP Strong Scaling

We run the strong scaling experiment by running the prefiltering algorithms on a 4096 by 4096 image and varying the number of threads (1, 2, 4, 8, 16, 32, 64, 68). For the microlens naive algorithm, the runtime decreases from 11808ms for one thread to 218ms for 68 threads,

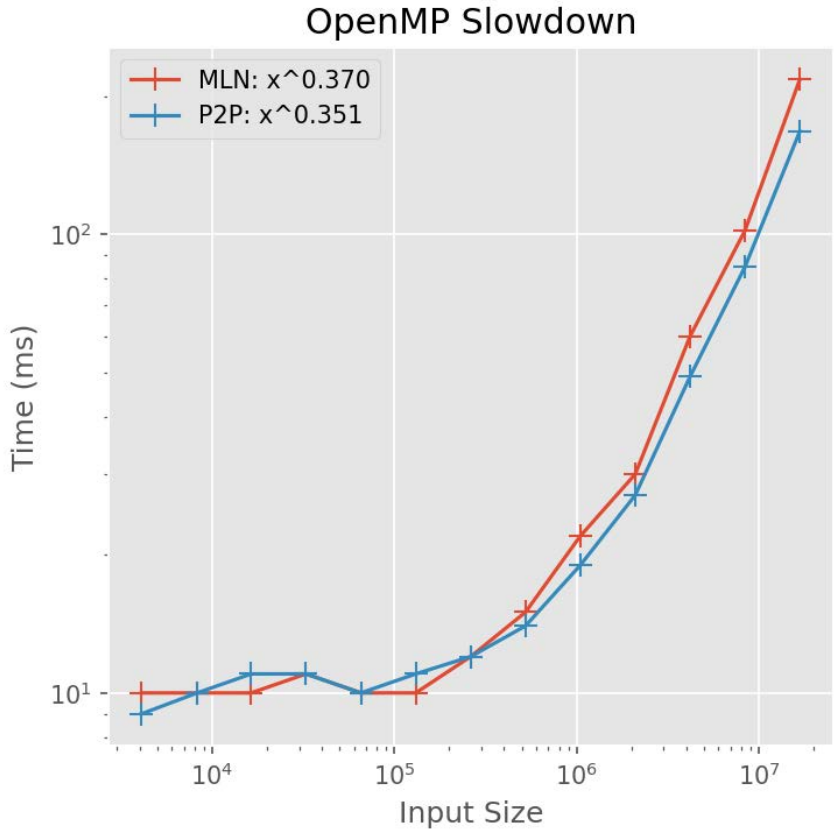
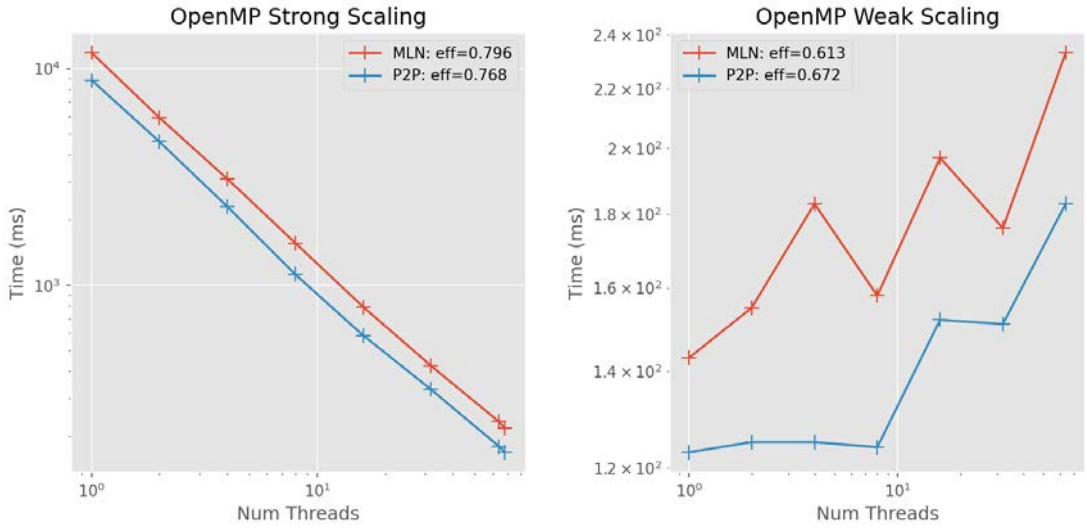


Figure 6.2: Serial slowdown of microlens naive algorithm and point to point algorithm implemented with OpenMP

yielding a strong scaling efficiency of 79.7%. For the point to point algorithm, the runtime decreases from 8826 for one thread to 169ms for 68 threads, yielding a strong scaling efficiency of 76.8%. Figure 6.3a shows the performance.

OpenMP Weak Scaling

We run the weak scaling experiment by running the prefiltering algorithms on a set of pair of (number of threads, input size): $(1, 512 \times 512), (2, 724 \times 724), (4, 1024 \times 1024), (8, 1448 \times 1448), (16, 2048 \times 2048), (32, 2896 \times 2896), (64, 4096 \times 4096)$. For the microlens naive algorithm, the runtime is 143ms for one thread and 233ms for 64 threads, yielding a weak scaling efficiency of 61.4%. For the point to point algorithm, the runtime is 123ms for one thread and 183ms for 64 threads, yielding a weak scaling efficiency of 67.2%. Figure 6.3b shows the performance.



(a) Strong scaling

(b) Weak scaling

Figure 6.3: Strong and weak scaling of microlens naive algorithm and point to point algorithm implemented with OpenMP

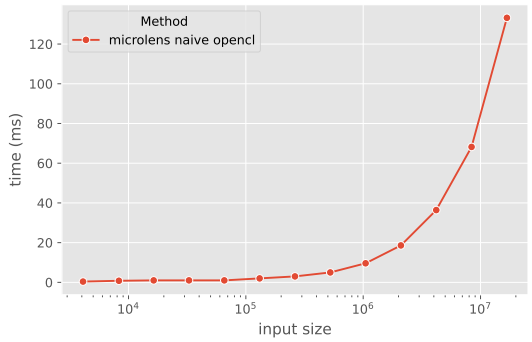
GPU Serial Slowdown

We run the OpenCL version of the microlens naive algorithm on a laptop with Intel HD Graphics 630 as well as on a single node with Nvidia Tesla V100 on Bridges-2. Figure 6.4 shows that the runtime increases roughly linearly as the number of pixels increases.

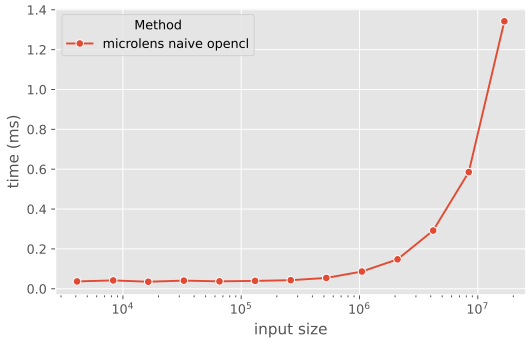
Note that in our implementation, we have to convert the input image from BGR format to RGBA format to make it compatible with OpenCL. Another overhead is OpenCL needs to read the code from the disk and compile it during runtime, causing significant overhead on Bridges-2 which has a slow hard drive. These overhead may be overcome by switching to CUDA, which does GPU code compilation during compile time and can process BGR images without conversion.

Comparison of Microlens Naive Algorithms

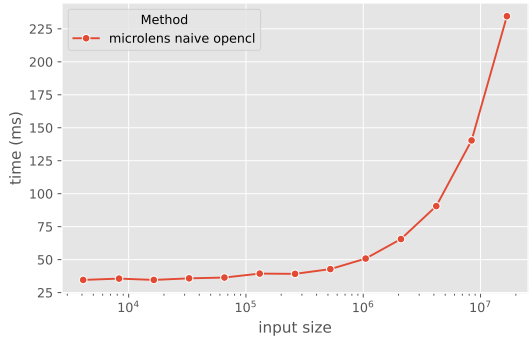
We run all versions of the microlens naive algorithms on a laptop with Intel i9-9880H and Intel HD Graphics 630. We only calculate the kernel runtime of the OpenCL version. Figure 6.5 shows the result. It turns out all parallelized algorithms exhibit a great speedup over the serial version, and the OpenCL version has the best runtime overall.



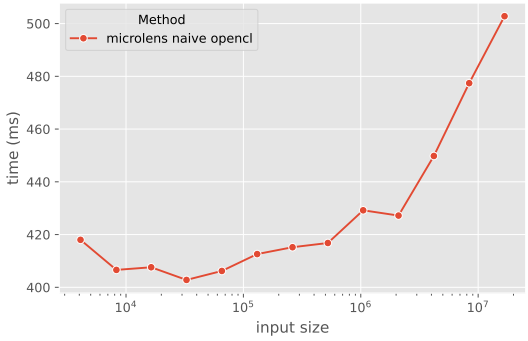
(a) Kernel Runtime on Intel HD Graphics 630



(b) Kernel Runtime on Nvidia Tesla V100



(c) Total Runtime on Nvidia Tesla V100



(d) Total Runtime on Nvidia Tesla V100

Figure 6.4: Runtime of microlens naive algorithm implemented with OpenCL

Comparison of Point to Point Algorithms

We run all versions of the point to point algorithms on a laptop with Intel i9-9880H. Figure 6.6 shows the result. It turns out all parallelized algorithms exhibit a great speedup over the serial version, and the cv parallel for version has the best runtime overall. Based on the experiment results on microlens naive algorithms, We believe if we were able to implement the OpenCL version, it would be even faster than the cv parallel for version.

6.3 Discussion

Overall, we are satisfied with the runtime and scaling efficiency of both the point to point algorithm and the microlens naive algorithm. Specifically, we are able to prefilter an image with a size of 1448×1448, which has roughly the same number of pixels as a full HD screen (1920×1080, in less than 33ms for both the CPU and GPU versions of both algorithms. That means we have achieved real-time performance (30fps) for VCD on a laptop with a

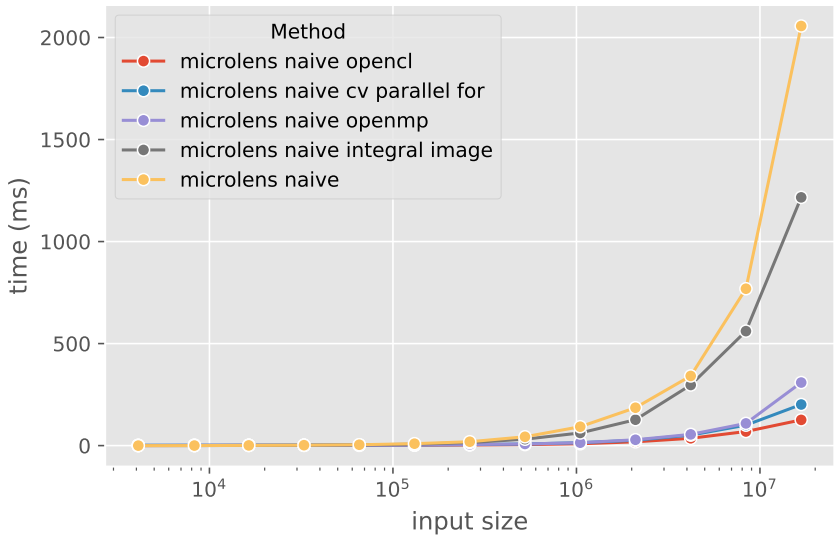


Figure 6.5: Comparison of microlens naive algorithms on Intel i9-9880H

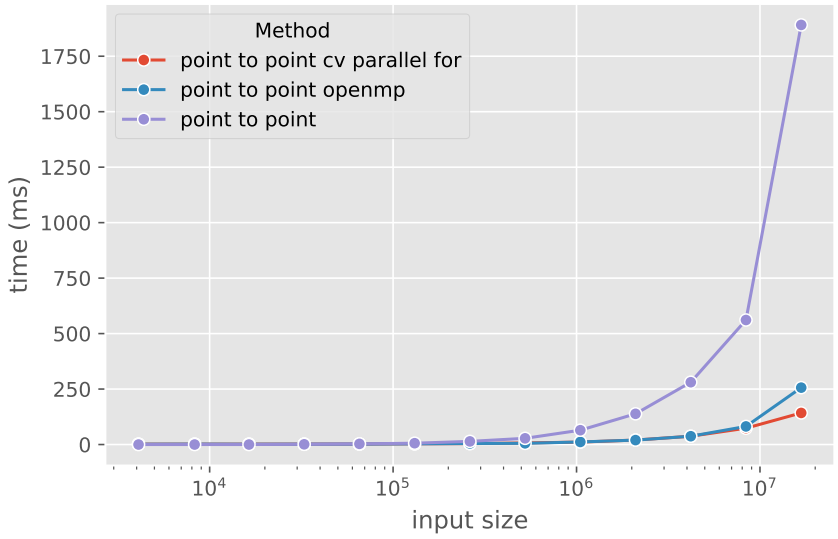


Figure 6.6: Comparison of point to point algorithms on Intel i9-9880H

reasonable CPU.

Bibliography

[1] Aydin Buluç. *An Introduction to CUDA and Graphics Processors (GPUs)*. 2022. URL: https://drive.google.com/file/d/1MAKMdsQEa0-pbYxOgSCzU1NDH72q3ieB/view.

[2] Aydin Buluç. *Introduction & Overview*. 2022. URL: https://drive.google.com/file/d/1cEe7IIiIqUtVX07rFJPZDHAviY0HP_z1/view.

[3] Angela Chen, Shiyun Xu, and Eric Ying. *Microlens Array Subgroup Fall 2020 - Spring 2021 Project Report*. URAP Report. EECS Department, University of California, Berkeley, May 2021.

[4] Jacob Holesinger. "Adapting Vision Correcting Displays to 3D". MA thesis. EECS Department, University of California, Berkeley, Sept. 2020. URL: http://www2.eecs.berkeley.edu/Pubs/TechRpts/2020/EECS-2020-181.html.

[5] Y. H. Hsieh, Y. T. Yu, Y. H. Lai, M. X. Hsieh, and Y. F. Chen. "Integral-based parallel algorithm for the fast generation of the Zernike polynomials". In: *Opt. Express* 28.2 (Jan. 2020), pp. 936–947. DOI: 10.1364/OE.380567. URL: http://opg.optica.org/oe/abstract.cfm?URI=oe-28-2-936.

[6] Fu-Chung Huang. "A Computational Light Field Display for Correcting Visual Aberrations". PhD thesis. EECS Department, University of California, Berkeley, Dec. 2013. URL: http://www2.eecs.berkeley.edu/Pubs/TechRpts/2013/EECS-2013-206.html.

[7] Matthew Jones. *Thin Lenses*. 2015. URL: http://www.physics.purdue.edu/~jones105/phys21900_Fall2015/Phys21900_Lecture19.pdf.

[8] Tarkan Al-Kazily, Melody Mao, Eric Liu, Chun-Ning Tsao, Wei-Chun Hwang, and Eric Ying. *Algorithms for Displays that Correct Nearsighted and Farsighted Vision*. M.Eng. Capstone Report. EECS Department, University of California, Berkeley, May 2020.

[9] Shuwei Li, Haohua Lyu, Siyu Zhang, and Kaiwen Yu. *Real-time Vision Correcting Displays Implemented with Parallel Computing*. M.Eng. Capstone Report. EECS Department, University of California, Berkeley, May 2022.

[10] Gregory I Ostrow and Laura Kirkeby. *Myopia*. Ed. by Sudha Nallasamy. 2021. URL: https://eyewiki.aao.org/Myopia.

[11] Daniel Palanker. *Optical Properties of the Eye*. 2013. URL: https://www.aao.org/munnerlyn-laser-surgery-center/optical-properties-of-eye.

[12] Vitor F. Pamplona, Manuel M. Oliveira, Daniel G. Aliaga, and Ramesh Raskar. "Tailored Displays to Compensate for Visual Aberrations". In: 31.4 (July 2012). ISSN: 0730-0301. DOI: 10.1145/2185520.2185577. URL: https://doi.org/10.1145/2185520.2185577.

[13] *VCD prefiltering repository*. https://github.com/vision-correcting-display/prefiltering.

[14] Yaying Zhao. MA thesis. EECS Department, University of California, Berkeley, 2020.

[15] Yirong Zhen. "New Algorithms for the Vision Correcting Display". MA thesis. EECS Department, University of California, Berkeley, 2019.