GPU-based Processing of Hartmann-Shack Images for Accurate and High-Speed Ocular Wavefront Sensing

Juan Mompeán[a,b,*], Juan L. Aragón[a], Pedro M. Prieto[b], Pablo Artal[b]

[a]*Dept. Ingeniería y Tecnología de Computadores, Universidad de Murcia, Spain*
[b]*Laboratorio de Óptica, IUiOyN, Universidad de Murcia, Spain*

Abstract

Hartmann-Shack aberrometry is a widely used technique in the field of visual optics but, high-speed and accurate processing of Hartmann-Shack images can be a computationally expensive/resource intensive task. While some advancements have been made in achieving high-performance processing units, they haven't been specifically designed for processing Hartmann-Shack images of the human eye with Graphics Processing Units. In this work, we present the first full-Graphics Processing Unit implementation of a Hartmann-Shack sensor algorithm aimed at accurately measuring ocular aberrations at a high speed from high-resolution spot pattern images. The proposed algorithm, called PaPyCS (*Parallel Pyramidal Centroid Search*), is inherently parallel and performs a very robust centroid search to avoid image noise and other artifacts. This is a field where the use of Graphics Processing Units have not been exploited despite the fact that they can boost Adaptive Optics systems and related closed-loop approaches. Our proposed implementation achieves processing speeds of 380 frames per second for high resolution (1280x1280 pixels) images, in addition to showing a high resilience to system and image artifacts that appear in Hartmann-Shack images from human eyes: more than 98% of the Hartmann-Shack images, with aberrations of up to to 4 microns Root Mean Square for a 5.12mm pupil diameter, were measured with less than 0.05 microns Root Mean Square Error, which is basically negligible for ocular aberrations.

Keywords: GPGPU; image processing; real time; tracking; Hartmann-Shack; wavefront sensing

1. Introduction

Optical aberrations are defined as the difference between the perfect (flat or spherical) wavefront for an ideal optical system and the bumpy wavefront generated by a real optical system. Optical aberrations (defocus, astigmatism, coma, etc.) cause deviations to the rays of the light beam, therefore preventing them from converging to a single focusing point and blurring the image. Wavefront sensing, i.e., optical aberration measurement, is routinely performed in a wide range of fields (e.g., Astronomy, Microscopy, Communications) for different purposes (e.g., optical quality determination, instrument calibration, optical design) [4]. Wavefront sensing is also pivotal in most Adaptive Optics (AO) systems, widely used in astronomy and vision science. AO aims to dynamically correct the fluctuating aberrations of a system in *real time* by means of a wavefront corrector (deformable mirror or liquid-crystal modulator) whose shape or refractive index distribution can be modified point-by-point to reshape the wavefront [8]. Due to the dynamics of the aberrations, an AO system requires very fast wavefront sensing and image processing, therefore, being a computationally intensive process. As wavefront sensing involves heavy image analysis and processing, it is an intrinsically parallelizable task which makes the use of General Purpose Graphic Processing Units (GPGPUs) a perfect candidate to achieve real time processing speeds.

The Hartmann-Shack (H-S) wavefront sensor, described in 1971 [29], which is based in the sensor proposed by Hartmann in 1900 [10], is the most widely used aberration measurement approach nowadays. The H-S sensor consists of an array of microlenses, which sample the aperture of the optical system, and an image detector that records the spot pattern generated (commonly called the H-S image). For an ideal system, each microlens focuses the collected wavefront into its focal point and a regular spot pattern is recorded (Figures 1a and 1c). For a real system, the wavefront irregularities cause local slopes over each microlens, resulting in a distorted spot pattern (Figures 1b and 1d). The displacement of each spot with respect to its ideal location (i.e., the focal point of each microlens) is related to the wavefront's local tilt, which in turn is related to the wavefront's local derivative. The H-S wavefront sensing operation, therefore, consists of recording the spot pattern, calculating the position of each spot (i.e., its center of mass or *centroid*), computing each centroid displacement in x and y directions and, finally, integrating these set of local derivatives to finally obtain the wave-

*Corresponding author
Email addresses: `juan.mompean@um.es` (Juan Mompeán), `jlaragon@um.es` (Juan L. Aragón), `pegrito@um.es` (Pedro M. Prieto), `pablo@um.es` (Pablo Artal)

front shape or aberration map (as that depicted in Figure 5).

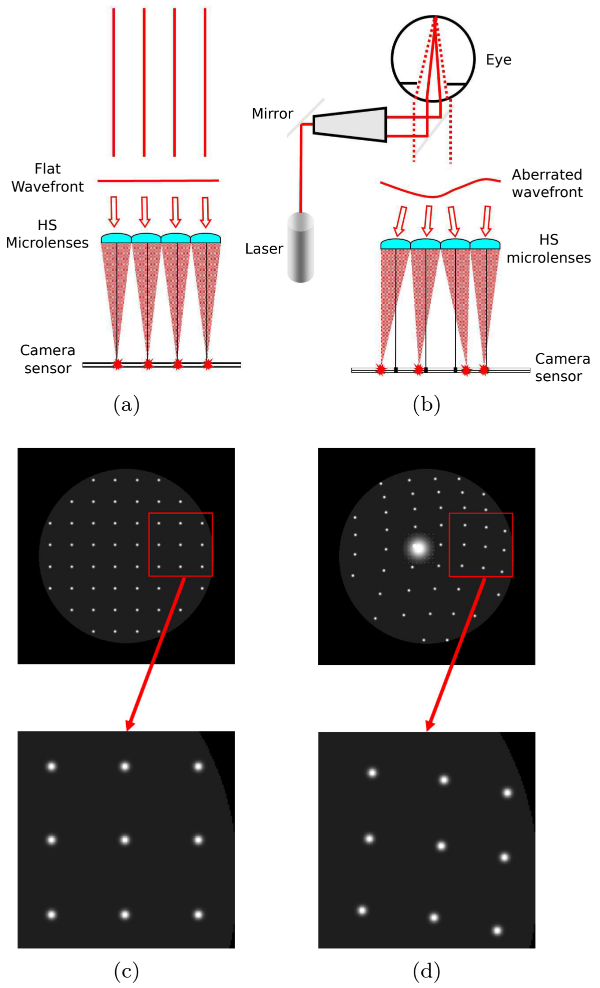


Figure 1: (a) H-S wavefront sensor diagram registering a flat wavefront. (b) H-S wavefront sensor diagram registering an aberrated wavefront. (c) Obtained H-S spot image for the perfect wavefront. The whole eye is shown on top and a zoomed area below. (d) Obtained H-S spot image for the aberrated wavefront, including a corneal reflection. Whole eye is shown on top, zoomed area below.

In this paper, our main focus is on measuring the aberrations of the human eye. Considering this special optical system is far from perfect, which results in a limited visual quality, measurement of the eye's aberrations, also known as *ocular aberrometry*, is a very important field of study. Over the last two decades, Ophthalmology and Vision Science applications have been a catalyst for wavefront sensor development [18]. There are several commercially available aberrometers for clinical use, most of them based on the H-S principle, and many other research apparatuses and prototypes, recently including binocular and open-view configurations [6]. In many cases, ocular aberrations are used for diagnosis and/or prescription of corrective optics but there are also Adaptive Optics systems for ocular applications [26, 3]. It is important to note that, given the fact that the human eye is a living system – closed, mobile, and fragile – *ocular aberrometry* is somewhat idiosyncratic and not completely interchangeable with other optics areas that deal with artificial systems (e.g., telescopes, microscopes, camera lenses). For example, H-S images from living human eyes suffer from corneal reflections (which severely degrade the spot images, as displayed in Figure 1, right) and brightness irregularities due to crystalline lens reduced transparency as an effect of aging, in addition to temporal fluctuations in spot intensity across the pupil due to the tear film and other factors.

In this paper we present a parallel Hartmann-Shack wavefront sensing algorithm for accurate yet high-speed ocular aberrometry. The proposed core algorithm, called PaPyCS (*Parallel Pyramidal Centroid Search*) parallelizes the centroid detection phase while performing a very robust centroid search, to make the algorithm immune to the aforementioned issues that degrade ocular H-S spot images. PaPyCS has been parallelized and optimized for GPGPUs as it will be detailed in Section 3. In addition to the spot detection and centroid search phase, polynomial fitting of the aberration and wavefront map calculation have been parallelized using the GPU as well. The pupil tracking algorithm, which is another crucial component in the process, as we will discuss later, has also been parallelized in the GPU. Experimental results show that our full approach achieves a speedup above 100x compared to its corresponding sequential implementation. This enables a high speed processing (up to 380 frames per second on 1280x1280 pixel images), while not sacrificing detection accuracy (more than 98% of the H-S images with aberrations up to 4 microns were measured with RMS-Error lower than 0.05 microns).

The key contributions of this paper are: developing a GPU-based high-speed implementation capable of processing H-S wavefront sensor images; creating a robust and accurate centroid detection algorithm based on dynamic pyramidal search; implementing a highly-parallel pupil tracking algorithm for Hartmann-Shack wavefront sensor images; and developing a parallel high-speed wavefront map calculation.

The remainder of this paper is organized as follows. Section 2 provides some background on H-S wavefront sensing and further motivates this work. In addition, Section 2 reviews the most relevant literature on real-time H-S image processing. Section 3 describes our parallel GPU-based implementation. In Section 4 we evaluate and report its performance and accuracy. Finally, Section 5 summarizes the main conclusions of the work.

2. Motivation and Related work

2.1. Background and Motivation

When measuring the wavefront and aberrations of a living optical system such as the human eye, H-S images must be processed to first detect the centroid of each spot of the microlens array, and then the wavefront aberration can be reconstructed from the set of spot displacements – similarly to other artificial optical systems such as telescopes.

However, as mentioned in the previous section, H-S spot images from a living human eye suffer from an important number of inherent problems that can seriously jeopardize the accuracy of the measured aberration. For example, instead of the typical single-pass arrangement, an *ocular* H-S always works in double-pass (i.e., a light beam is shined into the eye and its retinal reflection acts as actual source for sensing), and as a consequence, corneal and crystalline-lens reflections can fall into the H-S image, degrading the spot image. Furthermore, transparency may be reduced, especially in older eyes, producing irregularities in spot brightness. And retinal hazard severely limits light intensity for ocular uses, posing a constraint to the signal-to-noise ratio of the spot image. Even in modern sensors, noise is still a problem which degrades the quality of the images [12]. The position of the pupil is also critical for both processing the images and correctly presenting a stimuli or correcting the aberrations. Finally, when measuring the aberrations of living eyes along time, it is common to observe temporal fluctuations in spot intensity across the pupil due to inhomogeneities in the tear film and changes in the ocular media.

Furthermore, while highly distorted H-S images may be discarded in other applications, this is not typically the case for highly aberrated eyes, which must be dealt with, since they belong to real patients. In fact, subjects with pathologically high levels of ocular aberrations, seriously degrading their vision, are of a high interest from the point of view of ocular aberrometry since they have the largest room for improvement.

For all of the above reasons, it is crucial to rely on a very robust centroid search algorithm, capable of dealing with the aforementioned effects to produce highly accurate aberration measurements, yet able to perform at high speed in order to be effectively used in closed-loop Adaptive Optics systems. To that end, in this paper we propose PaPyCS, a very resilient spot search algorithm, as we will describe in Section 3.2.1, derived from the time tested algorithm that we proposed in the early 2000's [27, 11], capable of yielding very high accuracy for living optical systems, and as we will report in Section 4, achieving a high throughput as a result of the highly efficient parallelization performed in this work.

2.2. Related Work

The Hartmann-Shack wavefront sensor has been widely used in optics for a relatively long time and there are a number of published works aimed at speeding-up H-S image processing, specifically targeting, however, artificial optical systems, such as telescopes. To the best of our knowledge, no other previous work has developed a GPGPU implementation of a Hartmann-Shack processing system targeted at measuring the aberrations of living human eyes, performing on the GPU all the necessary tasks from H-S pupil tracking and spot centroid search, to extrapolating the Zernike polynomials and computing the wavefront map.

Mocci *et al.* [17] achieved a very short processing time by simply using the CPU. However, their implementation was intended for calibrating lasers, so they could use lower resolution images and a simpler (faster) spot search algorithm, than those needed for dealing with images from the human eye. Furthermore, since the laser beam position was fixed, no pupil tracking was needed. Yu and Zhang [32, 31] proposed another implementation in a CPU, achieving 110 frames per second (although just the processing could reach up to ˜166 FPS) on 512x512 pixels images with 193 microlenses (spots). Again, they did not perform any pupil tracking and there was a limited precision due to the low resolution H-S images.

An early GPGPU implementation for searching the Hartmann-Shack centroids and calculating the Zernike polynomials was developed by Marichal-Hernández *et al.* [13]. However, this implementation was developed for telescope optical systems, achieving a 10x speedup for the centroid search stage and 2x speedup for the wavefront map reconstruction stage.

One recent high-speed approach was proposed by Pichler *et al.* [25] using an FPGA (Field Programmable Gate Array) and an ASIC (Application Specific Integrated Circuit) implementation. They achieved a high throughput (operating at 830 Hz) although the resulting accuracy was limited because of the simple centroid detection algorithm which was used, in addition to the limited number of patterns available during the training of the neural network. They compute the brightest spot inside the cell of each lens (of the microlense array, see Figure 1-(b)) to find its centroid and calculate its displacement. Then, to reconstruct the wavefront aberration they proposed to use a neural network trained to relate the vector of spots displacements with previously generated displacement vectors whose wavefront aberration had been previously calculated. They developed an autonomous system capable of estimating the wavefront aberration at a high speed. Again, this approach was aimed at processing H-S images for artificial lenses (telescopes) and, therefore, the achieved accuracy (due to the simple spot search algorithm) is not appropriate for measuring the aberrations of living optical systems, such as the human eye, which is the target of our proposal.

Reichenbach *et al.* developed another approach using both FPGAs and GPUs to process Hartmann-Shack spot images [28]. This implementation was also aimed at wavefront sensing for artificial optical systems (telescopes in particular). Therefore, it does not include a pupil tracking step. Nevertheless, an interesting approach was used for spot centroiding. They proposed a two-stage algorithm: in the first phase the potential spots are found by searching for the brightest pixels within a given area; and in the second phase, the rest of spots in the H-S image are correlated with those in the list of potential spots, starting from a few selected ones (further details of this algorithm are given in Section 3.2.2). They use an FPGA for spot centroid search, while the CPU is used for correlating the spots, and finally, the GPU is used for reconstructing the wavefront aberration. They achieved a processing speed of 294 frames per second for an array lens producing 100x100 spots in a one megapixel (1024x1024) image. This is the fastest implementation we have found in the literature, although it was intended for artificial optical systems, we will compare our proposed PaPyCS algorithm for processing H-S images against Reichenbach's approach in terms of both performance and measurement accuracy.

On the other hand, when measuring the aberrations of the human eye it is crucial to start by detecting the center and size of the pupil of the patient, which, unlike that of artificial systems, typically moves and changes. While it would be possible to use a secondary camera for performing pupil tracking, it would increase the complexity of both the optical setup and the processing software. A more convenient approach is to track the pupil on the Hartmann-Shack images themselves. Some specialized pupil tracking methods have been developed to this end. The method developed by Meimon *et al.* [16] consists on integrating the pixels over each lenslet, thresholding them and fitting an ellipse to the border detected. Although this method has been characterized for its high accuracy, a high-performance version has not been developed. Another method was developed by Arines *et al.* consisting on thresholding the H-S image and calculating the centroid of the resulting image [1]. Although this method works for detecting the center of the pupil when it is completely inside the image, it does not work correctly with partially occluded pupils. Furthermore, it does not measure the pupil radius. de Castro *et al* [7] showed a pupil tracking method in the H-S images using the detected spots and a metric depending on their brightness to find the pupil. They were able to build a close loop system showing the advantages of performing pupil tracking in H-S images.

Additional developments have been carried out by Mauch and Reger [14, 15] using FPGAs. Their approach uses a tightly integrated FPGA with a camera to achieve an impressive rate of 905 frames per second (although the resolution of the camera is 224x224 pixels). A dynamic algorithm is used to be able to detect big aberrations. First, the spots are detected using a Connected Components Labelling algorithm. Later they are reordered to find their positions relative to the reference system.

In this paper, in order to reduce the complexity of the whole optical system, we detect and track the pupil directly on the H-S images. Using a method based on a previous work of our own [19] where a highly parallel pupil tracker was proposed to accurately estimate the pupil size and center, but in that case dealing with images of the eye illuminated with diffuse infrared light, producing dark pupils surrounded by the lighter iris. Instead, in this paper we modify that approach to live track the two pupils directly from the binocular H-S spot images (such as those shown in Figure 2), using the diffuse component of retinal reflection that back-illuminates the pupils, outlining them against the virtually black iris.

3. Parallelizing H-S image processing

This Section describes our full approach for accurately measuring ocular aberrations in both eyes at a high speed from high-resolution H-S spot images. All the stages involved in the processing of the H-S images have been parallelized: pupil tracking, centroid detection, wavefront calculation, and the aberration map calculation.

Regarding the core algorithm –spot detection and centroid calculation– the proposed PaPyCS (*Parallel Pyramidal Centroid Search*) algorithm evolve from our own previous work in [27] and [11], back in 2000, where it was first introduced the pyramidal algorithm for spot centroid search, as a tool for (off-line) measurement of monocular aberrations. This preliminary pyramidal algorithm is now highly parallelized for enabling real-time processing of both eyes simultaneously. Furthermore, our current approach has been adapted to the characteristics of current H-S wavefront sensors, with much higher microlens density and shorter focal length, resulting in H-S images comprised of a large number of very sharp and compact spots.

Section 3.1 details the pupil tracking phase. In Section 3.2 our PaPyCS algorithm for searching the centroid of the spots is described, together with Reichenbach's algorithm [28] used for comparison purposes. In Section 3.3 the methods used for calculating Zernike polynomial coefficients are discussed. Finally, Section 3.4 is devoted to the parallelization of the final algorithm responsible for calculating the wavefront map.

*3.1. **Step 1:** H-S Pupil Tracking*

When the Hartmann-Shack sensor is used to characterize the human eye's aberrations the system's pupil must be tracked so as to analyze the correct area of the H-S image. While in other artificial setups (e.g., in telescopes) the H-S spots in the CCD sensor area are static, when dealing with a human eye, the area of the sensor covered by the spots is constantly moving. Therefore, knowing the position of the eye's pupil is important to analyze the correct area of the image. Figure 2 shows an example of spot image taken with a binocular H-S sensor.

H-S images from living eyes show a high variability in brightness. The pupils of the subjects can also have different sizes and, in binocular systems, both pupils may partially overlap each other, or they may fall outside of the bounds of the camera's CCD. To correctly track the pupil, a robust algorithm needs to be developed. Our approach for H-S pupil tracking is able to operate at high-speed while overcoming those issues.

Our setup has been designed to work with binocular H-S images. First, the global H-S image is divided in two sub-images (Figure 3a shows one of them). The image is

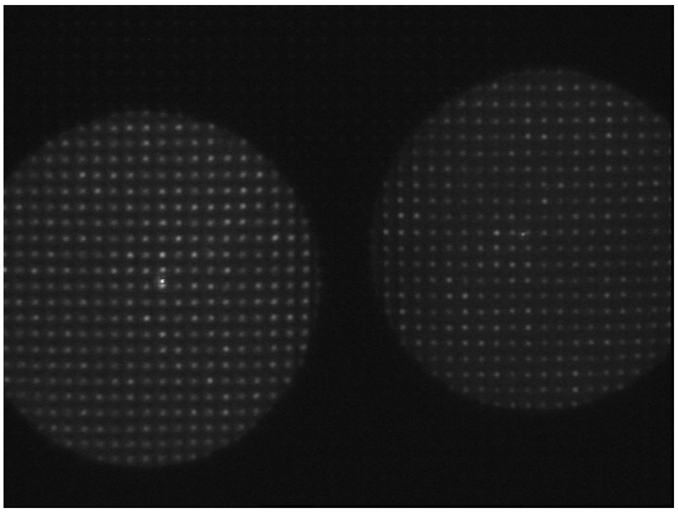


Figure 2: Binocular H-S sensor image.

then smoothed with a Gaussian filter (radius 4, sigma 4) as shown in Figure 3b. This is an important step since, given the nature of the H-S images, most of the light is concentrated in the spots. However, in this first stage we want to find the border of the pupil and for that we need to have a smooth rounded area brighter than the background outside. A Gaussian filter is appropriate since it reduces the difference in brightness between the spots and the surrounding area while maintaining the brightness of the background.

We have used a *thresholding* approach to select the pixels that are inside each pupil (Figure 3c). However, given the high variability expected in the brightness of the images, a fixed threshold value is not convenient. To overcome this issue, we have used the Otsu algorithm [24] to calculate an appropriate threshold value in each frame. Furthermore, large differences in brightness can be found between both pupils, since they might not be equally illuminated. Recall that two eyes are indeed two different optical systems. To avoid the problem of the illumination difference, we process each pupil independently, calculating a particular threshold for each one.

After *thresholding* each pupil, its borders are still too rough, and not as rounded as expected, to perform the ellipse fitting step properly. Furthermore, it is common to have holes inside the *thresholded* area, as shown in Figure 3c, that must be removed. To alleviate this problem, a *morphological close* operation is applied to smooth the borders around the *thresholded* area, as shown in Figure 3d. The resulting image is more suitable for processing by the ellipse fitting algorithm. The border is designated with a simple method: an above-threshold pixel is classified as a *border pixel* if any of its 8 neighbours is below the threshold, as shown in Figure 3e. Finally, several ellipses are fitted using a RANSAC (Random Sample Consensus) approach [9] to select the ellipse that best fits the border pixels (Figure 3f). The RANSAC technique is very appropriate here since most but not necessarily all the points are correct border pixels. Therefore, using the complete set of border points for the ellipse fitting could easily lead to an incorrect result. The RANSAC method[1] not only overcomes this issue but is also robust to missing parts (e.g., if part of the pupil falls outside of the CCD, or if it overlaps with the other pupil). Figure 3f shows the final best ellipse for this example.

GPU implementation of Pupil Tracking. The described pupil tracking approach has been fully parallelized by using CUDA. Since most of the operations are image processing functions, they can be parallelized by assigning one thread per pixel. However, in order to achieve a higher performance we have optimized the implementation to take advantage of all the resources available on the GPU. Several operations performed to find the pupil share the same pattern: they load data from a squared area in the input image and generate an output value for the output pixel. These operations are the Gaussian filter and the morphological close (a dilation filter followed by an erosion filter). Since they share the same memory access pattern, we have applied the following CUDA optimizations:

1. **Separable kernels**: Since the Gaussian, dilation and erosion filters are applied as 2-dimensional square filters, they can be separated into two 1-dimensional filters reducing the amount of operations and also the memory pressure. Instead of having one thread processing n^2 pixels, each thread processes $2 * n$ pixels. There are two phases: first, a 1-d filter is applied horizontally and its output is stored into a temporal image; second, the 1-d filter is applied vertically producing the final image. In the first phase all the pixels are processed, therefore, the data from each row loaded in the second phase is the result of applying the filter to n pixels horizontally.

2. **Shared memory**: Although the memory pressure is greatly reduced with the previous optimization, there are still many threads loading the same data, saturating and wasting the memory bandwidth. This can be alleviated by using the GPU's shared memory, which has a much higher bandwidth and lower latency than the global memory, resulting in higher overall performance.

3. **Template unrolling**: Loops are a convenient tool for processing data. However, the ending condition must be tested in each iteration and the accessed addresses have to be re-calculated for each iteration. By unrolling the loop inside the 1-d filter, the ending condition does not need to be tested anymore and the memory addresses are known at compile time. However, in order to apply the loop unrolling optimization, the size of the

[1]For fitting one ellipse we use 5 points, and a total of 1024 ellipses are fitted for each pupil.

(a) (b) (c)

(d) (e) (f)

Figure 3: (a) Original H-S image. (b) Smoothed image after the Gaussian filter. (c) Thresholded image. (d) Application of a morphological close to the thresholded image. (e) Borders found on the *closed* image. (f) Final elliptical pupil found on the image.

kernel input set must to be known at compile time. Fortunately, we only use a reduced range of radii and these can be provided at compile time.

4. **Reducing memory bandwidth usage**: Our H-S images are grayscale with values between 0 and 255. Consequently the bandwidth consumption can be reduced by changing the data type from 32-bit integers to 8-bit unsigned chars.

After pre-processing the H-S image, the ellipse fitting is performed. First we select the points that will be used for each fitting. We are using 5 points per fitting and we are doing a total of 1024 ellipse fittings. The 5 points for an ellipse are randomly selected from the overall set of border points. Afterwards, the best fitting is selected as the one with the most votes from the border points (a vote is accounted if the ellipse is less than 2 pixels away from the border point). Some of these operations are computationally intensive, and to reduce the execution time in the GPU we have applied the following optimizations:

1. **Fast math**: Some mathematical operations performed with high precision such as *pow* or *sqrt* can be performed with a lower precision but faster. We have used this lower precision in the ellipse fitting calculations without any significant difference in the results.

2. **Pre-calculated random numbers**: When the points for each fitting are randomly selected, the random in-

dices have to be generated within a range (0 to number of border points-1). Generating random numbers is an expensive operation that impacts the overall performance. As a solution we have decided to pre-calculate the random indices just once at the beginning, in the range of [0..1]. When the number of border points is known, these random indices are scaled to the final range (0 to number of border points-1). This is much faster than generating the random points every time, and the sets of points are still randomly selected and changed from frame to frame.

3. **Reduction with *ballot* and *popc***: After fitting the ellipses the votes for each one of them are counted. This task has been optimized using the *ballot* and *popc* instructions. *Ballot* sets the i^{th} bit of a given integer to one when the i^{th} thread of a warp is active and provides a number bigger than zero to the function. So after calling that function each warp has an integer with as many ones as votes there are in that warp. Finally, the *popc* instruction counts the number of bits set to one in an integer, i.e., the number of votes in each warp. This is faster than using either atomic operations or the shuffle instruction.

4. **Reducing device-to-host communications**: The last task is selecting the best ellipse fitting, i.e., the most voted one. In the *naive* implementation the index of the best ellipse was copied to the host and then the best ellipse was copied to the host. However, it is possible to avoid one memory copy (plus synchronization) if the best ellipse is copied to a known address in the device memory where it can be accessed directly from the host.

3.2. ***Step 2:*** *Centroid Search*

Once the pupil is found, the second step is to detect all the spots in the Hartmann-Shack image. This is the most critical phase in the entire wavefront calculation since each spot's centroid (or center of mass) must be computed with the best possible accuracy in order to achieve the most accurate wavefront reconstruction.

In this paper we propose PaPyCS, a *Parallel Pyramidal Centroid Search* algorithm, explained in the following Section 3.2.1. We also evaluate two other centroid search algorithms for comparison purposes, explained in Sections 3.2.3 and 3.2.2. Note that the three centroid search algorithms evaluated in this paper make use of a sub-pixel precision to provide the best possible input for the Zernike polynomials fitting step.

Finally, once the centroids for the H-S image are calculated we can determine their displacement with respect to a reference position. A reference H-S image, with no eye in place, is used for calibration purposes. The spots on this image are searched and their positions saved at the beginning of the process to be used as reference centroids throughout. Using a reference image also removes any pre-existing aberration that could be present in the optical system.

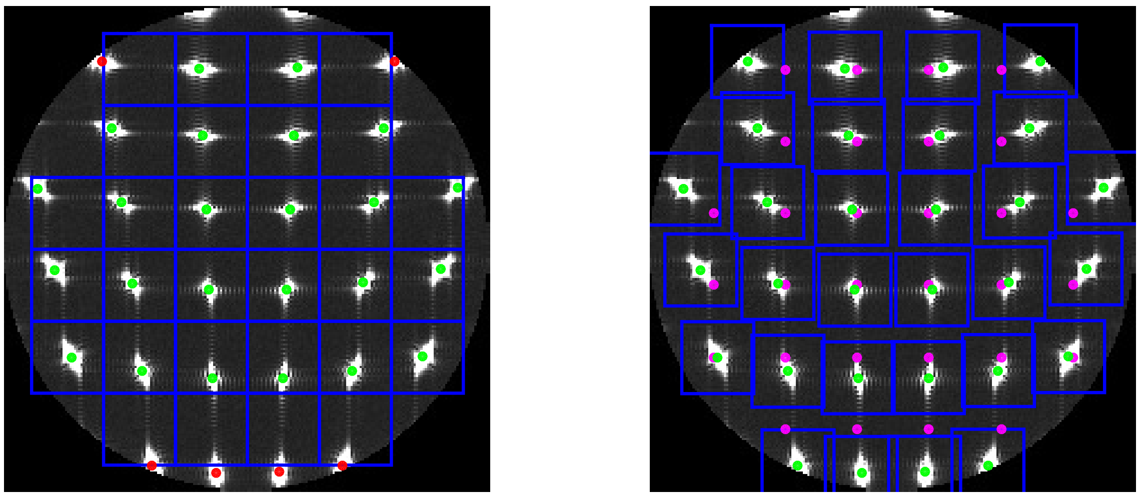
3.2.1. PaPyCS*: Parallel Pyramidal Centroid Search*

Our pyramidal algorithm divides the H-S image in sub-apertures. Each image has as many sub-apertures as microlenses in the lenslet array. As the size of the microlenses is known (it is a design parameter given by the manufacturer), the image can be divided in to small sub-windows each one theoretically containing one spot. To determine the position for each sub-window, the reference image is initially processed to find its centroids, as mentioned before. These reference centroids are then used as the center for each one of the search sub-windows. This way we create a *static search mesh* where each sub-window of the *static mesh* serves as the starting search area for each centroid in the actual H-S image we want to evaluate (see Figure 4a).

The pyramidal algorithm then starts an exhaustive search to find the centroid of each spot. While other simpler algorithms just search for the brightest pixel, the pyramidal approach iteratively calculates the center of mass of the current sub-window, and for the next iteration it re-centers a smaller sub-window in the current centroid, repeating the search. In each iteration, the minimum value of the pixels within the scanned area is calculated. And subtracted from each pixel while the center of mass is calculated, which enables a faster convergence towards the centroid of the spot. When the center of mass of the i^{th} iteration is calculated, the window side is reduced by 1 pixel, and the next iteration is performed until a minimum scan window of 3x3 is reached. This method has sub-pixel accuracy since the center of mass is calculated using float values, and weighted values are considered for the pixel borders to exclude the sub-pixel region that falls outside the scanned area, as proposed in [27].

While this iterative approach is very robust against image noise and helps to reduce the impact of corneal reflections, the use of a *static* search mesh limits the area where each spot is expected to be found. This might lead to wrong aberration measurements for strong aberrations, if the spot falls outside its corresponding microlens cell (see red points in Figure 4a). To overcome this issue we propose the use of a *dynamic search mesh* as follows.

Use of a Dynamic Search Mesh. In order to increase the the range of measurable aberrations, a *dynamic search mesh* is proposed in PaPyCS. In a first stage, a small static mesh is used to search the centroids just for a reduced central part of the pupil, and a preliminary aberration is calculated for this central area. In a second stage, this preliminary aberration for the current H-S image is extrapolated to the overall pupil size, its derivatives are used to predict the likely spot positions, and a new distorted mesh is created form these positions. A second centroid search is performed for this *dynamically* expanded mesh. This optimization allows us to process H-S images with



(a) (b)

Figure 4: (a) Static mesh for searching the spots. Blue rectangles show the search windows. Red points show spots outside of their search window while green points show spots inside. (b) Dynamic mesh for searching the spots. Blue rectangles show the search windows. Green points show spots inside their search window; magenta points show the center of the static (initial) search windows.

strong aberrations while still using a mesh for finding the spots, as it can be seen in Figure 4b.

Discarding Stage. As previously hinted, ocular Hartmann-Shack images can suffer from important amounts of *noise* depending on the optical setup and the subject. This *noise* can be just a background, which is not a problem for the pyramidal approach, it can be a corneal reflection, or it can be the result of dry eyes where the tear film is broken. Noise and other factors can lead to incorrect spot centroiding which can greatly distort the calculated aberration. To mitigate the impact of these issues, three criteria have been used to discard incorrect spots.

1. If a cell is empty or saturated, it should be discarded. To test if this is the case, the value of the pixel in the centroid point is compared with the values of the pixels around it. In particular, it is compared with the mean value of the eight neighbours around it.

2. Comparing the centroid of a spot with it neighbouring spots. While strong high-order aberrations can produce odd displacements of spots, it is usually safe to assume that spots are not either too close or too far from each other.

3. Finally, any isolated spot is discarded because it is complicated to test its centroid against its neighbours. Furthermore, an isolated spot typically means that the spots around it are incorrect as well.

GPU implementation of PaPyCS. In order to parallelize the pyramidal centroid search at a high level, the work is divided in spots. Since the work to be performed within a spot is independent from the rest, PaPyCS can be parallelized efficiently, contrarily to *Reichenbach* algorithm, which is inherently sequential (as it is described in Section 3.2.2). Therefore, each spot is searched by a group of CUDA threads. The amount of threads working on each spot is fixed to 32 because the *shuffle* instructions are used to share data between the threads working on the same spot, and also because the scope of the *shuffle* instructions is limited to threads within the same warp. The pseudo-code for this high-level parallelization is shown in Algorithm 1.

Algorithm 1: PaPyCS high-level pseudo-code.

```
1 Image: Hartmann-Shack image
2 Spots: Array with the coordinates of each spot
3 NumSpots: Number of spots
4 NumSteps: Number of steps for the search
5
6 block(32 * SPOTS_PER_BLOCK, 1, 1)
7 grid(NumSpots / SPOTS_PER_BLOCK, 1, 1)
8 SearchSpots<<<grid,block>>>(Image, Spots,
    NumSpots, NumSteps)
```

At a lower level of parallelization, a group (warp) of 32 threads calculates the centroid of one spot. Each thread processes one part of the scanned window and calculates the centroid of that area (lines 12-17 in Algorithm 2). To improve the effective memory bandwidth, vectorized accesses are used (line 15 in Algorithm 2). However, alignment of the accessed memory is not guaranteed, therefore, the first and last part of the pixels loaded are treated differently if they are not aligned (lines 14 and 16 in Algorithm 2). Finally, the threads of each group share their results with each other, adding them using CUDA *shuffle* instructions (line 19 in Algorithm 2). With the obtained result,

Algorithm 2: Kernel of the PaPyCS algorithm.

```
Image: Hartmann-Shack image
Spots: Array with the coordinates of each spot
NumSpots: Number of spots
NumSteps: Number of steps for the search

currCenter ← GetCenter(Spots, threadId)
for $i \leftarrow 0$ to NumSteps-1 do
    pyramidSize ← $LensSize - i$
    Each thread computes its partial center of mass
    partCenter ← (0, 0)
    minimum ← inf
    for $y \leftarrow$ currCenter.$y - \frac{\text{pyramidSize}}{2}$ + threadId to currCenter.$y + \frac{\text{pyramidSize}}{2}$ by 32 do

        [partCenter, minimum] += PartialMassCenterMissalignedBeginning(Image, currCenter, y, pyramidSize)
        [partCenter, minimum] += PartialMassCenterVectorized(Image, currCenter, y, pyramidSize)
        [partCenter, minimum] += PartialMassCenterMissalignedEnd(Image, currCenter, y, pyramidSize)
    end
    Threads computing the same spot share their partial results
    currCenter ← ShuffleMassCenter(partCenter, minimum)
end
StoreCenter(Spots, currCenter)
```

each thread calculates the new centroid and the process is repeated until the final scanning area is reached (loop in line 7 in Algorithm 2).

A couple of optimizations that have been applied in this parallelization process are further described next:

1. **Shuffle instructions**: After each group of threads has processed the pixels belonging to its spot, they have to share their results with each other. By using *shuffle* instructions data can be shared efficiently between the 32 threads of the group. In particular, the *shuffle* instructions are used to accumulate the results of each thread and, finally, thread 0 scatters the centroid position to the other 31 threads within the warp. The centroid data is calculated by each thread using 4 parameters: the sum of the x coordinates weighted by each pixel intensity; the sum of the y coordinates weighted by each pixel intensity; the total sum of pixel intensities; and the minimum intensity value within the area. These four numbers are reduced using the *shuffle* instructions. As a side note, the performance has been further improved by interleaving the four *shuffle* instructions.

2. **Vectorized memory accesses**: In order to calculate the center of mass of each spot, the corresponding pixels are loaded from global memory one by one. It is possible to improve the loading of this data by vectorizing the memory accesses. The data type of a pixel is *uint8_t*, i.e., a size of 8 bits per pixel. So, four pixels can be loaded simultaneously (32 bits) and later separated into four independent pixels. Address alignment is ensured and non-aligned pixels are treated separately. As a result, memory operations are more efficient and the total execution time is reduced.

To calculate the *dynamic search mesh*, a matrix with the derivatives of the Zernike polynomials is calculated for the overall pupil size. This matrix is multiplied by the value of the Zernike coefficients to obtain the theoretical displacements of the centroids for the given aberration. These displacements will be used to set the starting search windows (the so-called dynamic mesh) for the centroids. The multiplication of the two matrices is performed using the *CUBLAS* library [23]. The calculation of the starting search windows is parallelized by using as many threads as spots there are in the pupil.

The *discarding stage* is implemented in several steps. First, one thread per spot is used to calculate the difference between the central pixel and the mean value of its eight closest neighbours. If this difference is bigger than 15 the spot is copied to the list of valid ones. Otherwise, the spot is discarded. To test the second criterion, the spots are copied from the 1D vector where they are stored to a 2D structure of the same size as the microlens array. Then each spot is compared with its four neighbouring spots and, if any of them is either too close or too far, they are both discarded (again using one thread per spot). Finally, all the spots are checked and any one of them found isolated is discarded as well.

3.2.2. Reichenbach's Algorithm

The algorithm developed by Reichenbach *et al.* [28] follows a very different approach from PaPyCS. It starts by processing the whole image, pixel by pixel, searching for a local maximum, over a threshold, within areas of 13x13 pixels. This first step is intended to detect all the potential spots in the image (i.e., all the distinct small areas with a maximum point). This step fits quite well the SIMD architecture of a GPU, with different threads working on different data but running the same instructions.

In a second phase, the rest of spots are correlated with those in the list of potential spots, starting from a few selected ones. Given an H-S image of a human eye, it is known that spot displacements are bigger as we go away from the center of the pupil. Therefore, central spots are typically close to their original reference position. Unfortunately, corneal reflections are also typically located around the center of the pupil and they can be easily misdetected as spots. In any case a group of 5x5 spots in the center of the pupil are initially selected and assigned to their closest positions in the reference H-S image. From them, other spots in the pupil are *correlated* with those in the list of potential spots. In order to detect a new spot, two previously calculated ones are used, and from their respective distance and direction an approximated position is extrapolated for a new spot. Then the closest spot, from the list of potential spots found in the first step, to this extrapolated position is selected. This process is repeated, iteratively, in all directions until the border of the pupil is reached and no more spots can be found.

One advantage of this algorithm, when compared with our pyramidal PaPyCS, is its capability to detect spots that are outside their reference sub-window, which only happens for very strong aberrations. However, its major disadvantage is the lack of robustness in the presence of corneal reflections or when the signal-to-noise ratio is low. Both situations are very common when dealing with H-S images from human eyes, although it is not the case for artificial optical systems (such as telescopes) for which this approach was designed. Finally, it is worth noting that Reichenbach's algorithm is inherently sequential, since the position of each spot is derived from some previously calculated ones, which makes it hard to parallelize. Also note that because of its incremental spot searching approach, any early error (e.g., a corneal reflection being mis-detected as a legit spot) can be propagated to the rest of searched spots, leading to higher errors in the calculated aberration, as it will be seen in the results section.

GPU implementation of Reichenbach's. In the first stage of the algorithm, all the potential spots are searched. To do so the image is divided in tiles and each CUDA block loads one tile to shared memory. One thread per pixel is used to test if there is a potential spot in that location, and if so, it is added to a 1D vector which will contain the list of potential spots. When this list is finished, an initial mesh of 5x5 spots is created and the centroid of each one is determined to be the closest in the list of potential spots. A parallel search is done to find the closest spot in the list, with 32 threads working in parallel for each spot in the mesh. After completing the initial 5x5 mesh, other spots are *iteratively* searched for in two steps. First, one thread per potential spot tests whether its position can be extrapolated from the previously found spots. If that is the case, it is added to a list of temporary spots. In a second step, the closest spot to each one of the temporary spots from the potential spots is searched, and if any is found it is stored as a new found spot. This procedure is repeated for as many steps as spots fit in the chosen radius of the pupil. One specific CUDA optimization has been applied:

1. **Shared memory**: Since many threads in the same block are accessing the same memory addresses, the use of shared memory reduces the amount of loads issued to global memory and greatly reduces the latency of the load operations.

3.2.3. Center of Mass Algorithm

The third and final centroid search algorithm we evaluate in this paper is called the *Center of Mass* (CoM) algorithm. This is a very straightforward and naive implementation, which is not computationally intensive at the cost of a much lower accuracy. Actually, it can be seen as a simplified version of the pyramidal PaPyCS algorithm described in Section 3.2.1. The difference between them is the way the centroid of each spot is calculated. While the *pyramidal* approach performs an iterative converging search, the *center of mass* performs only one iteration to calculate the centroid, starting from the reference H-S image and using the static search mesh. This reduces the execution time but also degrades its accuracy.

3.3. ***Step 3:*** *Zernike Polynomials Fitting*

Once the centroids of all the spots have been calculated, the next step is fitting the Zernike polynomial derivatives to the displacements of the spots to obtain the so-called *Zernike coefficients* [30, 21, 22, 20] by solving an overdetermined equation system.

In order to parallelize this step, we have tested two different methods. The first one consists of using the *gels* function from the *CUBLAS* library, in order to perform the least squares fitting. The second method consists of calculating the inverse of the matrix containing the derivatives of the Zernike polynomials and multiplying the result by the displacements of the spots. Although both methods pursue the same purpose they achieve very different accuracy and performance. The *gels* function is slower but more accurate than calculating the inversion matrix. To speedup the processing the inverse has been used in all the performance and accuracy tests.

3.4. ***Step 4:*** *Wavefront Map Calculation*

While the values of the Zernike coefficients are the most important data for characterizing a wavefront, it is often convenient to present the aberrations as an image, the so-called *wavefront map*, in order to intuitively convey the magnitude and distribution of the distortions. An example of a wavefront map is shown in Figure 5. Furthermore, if a closed-loop Adaptive Optics system is running, the wavefront map might be used to update a spatial light modulator to dynamically correct the measured aberration.

Calculating the wavefront map of an aberration is a computationally-intensive operation, specially when many Zernike polynomials are used and/or the size of the image is big. Therefore, a high performance implementation is required for real-time (or higher) performance. In this paper we have developed a parallel GPU version of the wavefront map reconstruction step that will be evaluated at the end of the results section, in 4.4.

To reconstruct the wavefront map of a given aberration, the wavefront map of each Zernike polynomial is first calculated. These calculations are done only once since

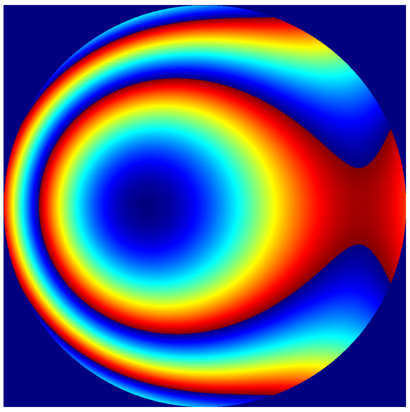


Figure 5: Example of wavefront map.

the result, can be reused. Afterwards, the map of each polynomial is added, weighted by the value of its corresponding Zernike coefficient. The modulo 1 of the value of each pixel is calculated and stored in the output image. Finally, the values of the image are scaled to be between 0 and 255.

GPU implementation. Each step has been parallelized in our CUDA implementation, using one thread per pixel processed with two remarkable optimizations:

1. **Shared memory**: The values of the Zernike coefficients are used by all the threads. Since these values are the same for all of them, the use of shared memory results in a significant improvement. The Zernike coefficients are loaded once from global memory and subsequently are accessed from the much faster shared memory. Moreover, the careful use of CUDA *pitched memory* and *pinned memory* results in additional performance improvements.

2. **Half data type**: Although calculations are performed with floating point arithmetic, a high precision is not required for visualization purposes. Instead of using *float* or *double*, the *half* data type is used (with a size of 2 bytes), reducing the bandwidth required to load data. Note that the main bottleneck of the kernels is the huge bandwidth required to load the wavefront map of each Zernike polynomial. Operations within the kernel are still performed using FP (*float*) arithmetic.

4. Experimental Results

4.1. Evaluation Methodology

In order to measure the performance and accuracy of the implemented algorithms, several configurations have been evaluated, varying two main parameters. First, the number of spots inside the pupil: we have evaluated four cases (250, 500, 1100 and 2000 spots) with a constant image size of 2560x2560 pixels. Since the image size was fixed, the area for each spot became smaller as their number increased. The second varied parameter was image resolution: 640x640, 1280x1280 and 2560x2560 pixels. In this case the number of spots was fixed at 250, hence the area corresponding to a spot increased with image size. Real systems may have a wide range of configurations regarding the number of microlenses used and the number of pixels in the camera sensor. Therefore, the tested configurations should be very useful to understand the performance of the implementations in real systems used in very different situations.

To properly evaluate the accuracy of the centroid search algorithms, our first set of H-S images was synthetically created using MATLAB from a set of given aberrations, generating a video with 100 H-S images. Although the aberrations were randomly selected, the weight of each Zernike order was set to comply with the statistics by Castejón-Mochón *et al.* [5] for a population of normal subjects. To make this first set of synthetic H-S images more realistic, reflections and brightness maps were added. Reflections are quite common in a H-S system. Sometimes they can be mitigated but this is not always possible. These brightness maps were extracted from real H-S images and emulate local changes in brightness as found in real images. Finally, random (white) noise was also added to the whole H-S image.

Actual H-S images from a real H-S system (Figure 6) have also been used to evaluate both performance and accuracy of the centroid search algorithms. Although, accuracy in this case is more difficult to check since the original aberration is unknown and not readily available for comparison as is the case for synthetic images. Still, we took images from an artificial eye placed in front of the actual H-S system. As this was a static (non-living) experiment, only the image noise was expected to change from frame to frame. We used the variability of the measured aberration as an indicator of the robustness of our PaPyCS algorithm against noise and other external artifacts that pollute actual H-S images (reflections, brightness variations, system instability, etc.).

The graphic card for testing was a state-of-the art NVIDIA 980 GTX implementing 2048 cores. CUDA version 8.0 was used. The sequential implementation was tested in an Intel i5-4690 (up to 3.9Ghz) processor which integrates four physical cores compiling with *gcc* V7. The OpenCV library (v3.3.1) [2] was also used in the sequential implementation for the computer vision functions (Gaussian filter, threshold, and morphological close) that involved in the pupil tracking step. OpenCV implements highly optimized functions, making use of the AVX vector instructions whenever possible.

4.2. Performance

It is important to note that all the performance results showed in this Section include the copy time between CPU and GPU, or vice versa, whenever a copy is performed (i.e., copying H-S images from CPU to GPU, copying Zernike

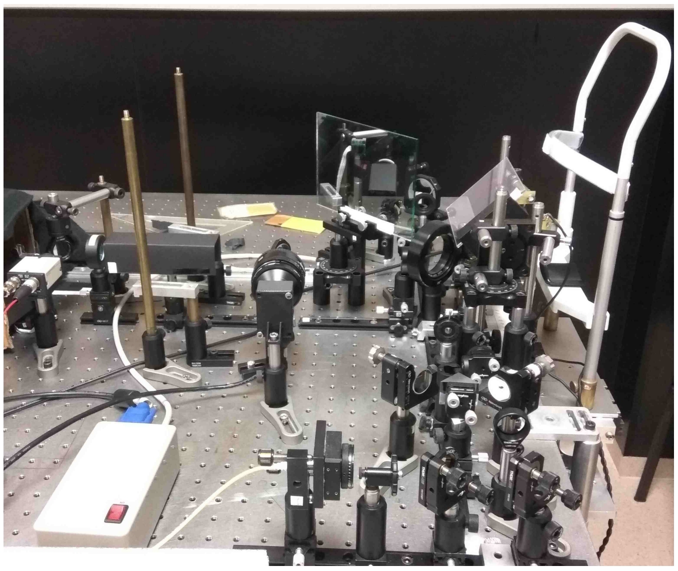


Figure 6: Hartmann-Shack open-view setup for human eyes.

coefficients to CPU, or copying wavefront map to CPU, in addition to any other intermediate copy that the algorithms require).

4.2.1. Speedup

When analyzing the achieved speedup, the results have been separated in two different categories: varying the number of spots (Figure 7) and varying image size (Figure 8). Both figures show, for each of the three evaluated search algorithms, the speedup achieved (over its own sequential implementation) for just the centroid search phase (Step 2), and also the overall speedup for the whole process (i.e., including all the 4 processing steps explained in the previous Section).

When varying the number of spots, the speedup of the centroid search step (first three bars of each group) scales well with an increasing number of spots, as it can be seen in Figure 7. In particular, the average speedup for the centroid search phase using the 2000-spots configuration is 192x, 78x and 66x for PaPyCS, Reichenbach's and the Center of Mass (CoM) algorithms respectively.

While it is interesting to analyze the speedup for just the centroid search step, it is even more relevant to examine the speedup obtained for the whole process, represented by the last three bars (of each group) in Figure 7. Again, the scalability for an increasing number of spots is very good for the three algorithms, obtaining average *overall* speedups of 103x, 65x and 58x (2000 spots configuration) for PaPyCS, Reichenbach's and the Center of Mass (CoM) algorithms respectively.

When analyzing the scalability depending on the image size, the results are more complicated, as shown in Figure 8. Reichenbach's algorithm scales well, because the number of working threads is increased as the image size is increased. However, in PaPyCS and CoM algorithms each thread is doing more work because the size of the search window is increased linearly the image size, going from 32x32 pixels up to 128x128 pixels, while the number of threads working on each window is the same, 32 threads, to be able to use the *shuffle* instructions. Therefore, each thread is processing a bigger area and the speedup does not increase with image size. The average speedup obtained for the Centroid Search phase (2560x2560 configuration) is 115x, 15x and 19x for PaPyCS, Reichenbach's and CoM algorithms respectively.

Finally, the last three bars (of each group) in Figure 8 show the speedup for the overall process. In this case, the weight of the centroid search step is small in comparison with the rest of operations for the biggest image size (we will discuss this point in Section 4.2.2 by showing a breakdown of the execution time). Summarizing, the average *overall* speedup for the 2560x2560 image size was 47x, 13x and 14x for PaPyCS, Reichenbach's and CoM algorithms respectively.

4.2.2. GPU Time

The speedup is an interesting measurement to show how well each method fits the GPU architecture, but to better understand these results it is worth analyzing how much time is spent on each step of the whole process for each configuration.

Figure 9 shows the time in milliseconds that each algorithm spends on each task for the GPU implementation, when considering an increasing number of spots. Note that as they all share the same implementation for copying, preprocessing, pupil searching and coefficient fitting, the same time is reported for those tasks. Only the *centroid search* phase changes. One interesting property exhibited by PaPyCS is that its execution time is mostly independent of the number of spots, unlike Reichenbach's, whose latency increases as the spot count progresses, making it less efficient for dense H-S wavefront sensors. In general, most configurations achieve a throughput of more than 100 frames per second (i.e., less than 10ms per frame) for high-resolution images (2560x2560 pixels) with up to 2000 spots.

Similarly, Figure 10 shows the time in milliseconds that each algorithm spends on each task for the GPU implementation as a function of image size. Focusing on the centroid search step, the time spent by PaPyCS increases with image size, especially for the biggest (2560x2560) image, for the same reasons explained above. Reichenbach's algorithm also increases its search time but less markedly.

4.3. Accuracy

In Optics the aberration of a wavefront is commonly *summarized* by its RMS (Root Mean Square) since Zernike polynomials are orthonormal to each other, wavefront RMS equals the square root of the sum of squared Zernike coefficients. However, RMS is also used for calculating the error of a particular measurement (w.r.t. a reference point). The later case is also known as RMSE (Root-Mean-Square

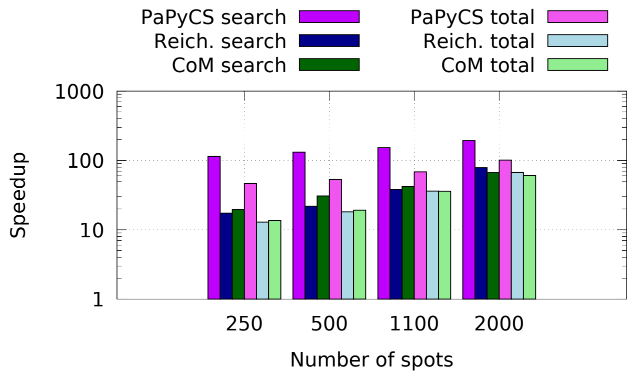


Figure 7: Speedup over the sequential implementation of both the centroid search step and the overall process, depending on the number of spots (image size fixed at 2560x2560 pixels).

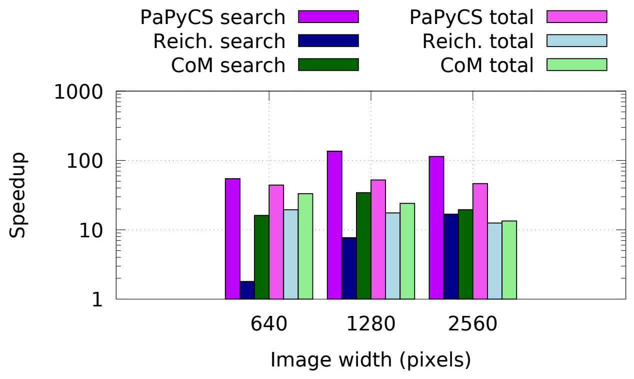


Figure 8: Speedup over the sequential implementation of both the centroid search step and the overall process, depending image size (the number of spots was fixed at 250).

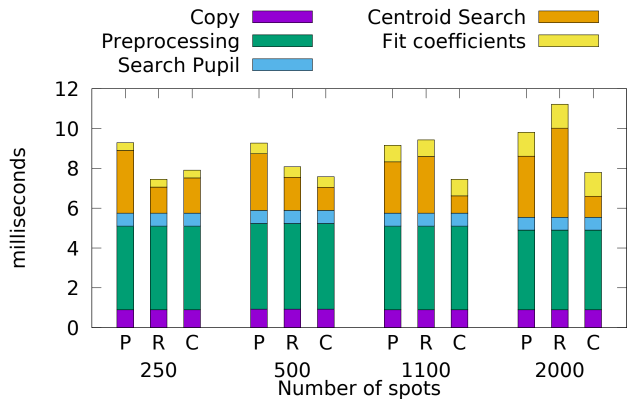


Figure 9: GPU time spent in each task for a varying number of spots (image size was fixed at 2560x2560 pixels). P stands for PaPyCS, R for Reichenbach's and C for Center of Mass.

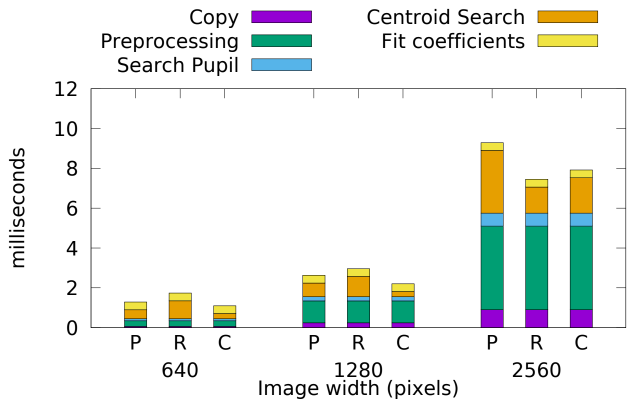


Figure 10: GPU time spent in each task for a varying image size (the number of spots was fixed at 250). P stands for PaPyCS, R for Reichenbach's and C for Center of Mass.

Error). When used for calculating the error of a measured aberration, the RMSE accumulates the squared difference between the measured Zernike coefficients and their reference values.

As mentioned previously, in order to properly evaluate the accuracy of the centroid search algorithms, a series of H-S images containing 250 spots were synthetically generated with MATLAB. We evaluated aberrations whose magnitude, or RMS, ranged from 0.5 to 6 microns, in 0.5 micron steps. The pupil diameter used for the generated images was 5.12mm. For each aberration we randomly generated 100 H-S images with the features previously described (added noise, brightness map and reflections). As previously stated, the distribution of the aberrations followed the trend of a normal population as reported in [5].

As a first accuracy analysis, after processing all the generated images with both the PaPyCS and the Reichenbach's algorithms (we have skipped the Center of Mass algorithm due to its high inaccuracy – it only works acceptably for low-aberrated and artifact-free H-S images) we calculated the aberration RMS from each H-S image. Figure 11 shows the measured RMS values against the actual (theoretical) ones for the PaPyCS (crosses) and the Reichenbach's (circles) algorithms. The aberrations measured by PaPyCS fall close to the actual values, especially below 5-micron RMS (which would correspond to a very strongly aberrated eye). Conversely, Reichenbach's algorithm exhibits a much bigger dispersion for the whole range of aberrations considered.

In a more detailed error analysis, Figure 12 shows the average RMSE (blue and orange lines) considering the 3rd, 4th and 5th degrees of Zernike coefficients for those images whose RMSE is smaller than 1 microns. This *quality* requirement avoids the few outlier cases that highly distort the average. The graph also shows (with bars) the percentage of images that fall within the required [0..1] RMSE range. PaPyCS achieves a significantly higher amount of images (96% on average) detected within the [0..1] RMSE range (vs. 71% for Reichenbach's). Considering the RMSE values (blue and orange lines), PaPyCS achieves an average RMSE smaller than 0.05 microns for aberrations ranging from 0.5 to 4. Reichenbach approach, however, discards 29% of the H-S images, and those not discarded are measured with an average RMSE of 0.20 microns. Overall,

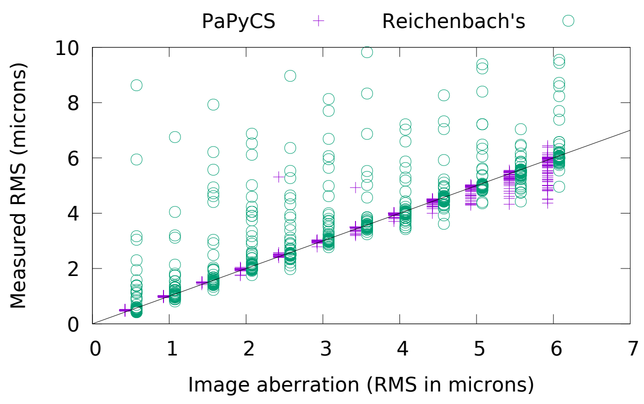


Figure 11: Measured vs theoretical RMS for PAPYCS (crosses) and Reichenbach's (circles) algorithms. The gray line represents perfect 1:1 correlation.

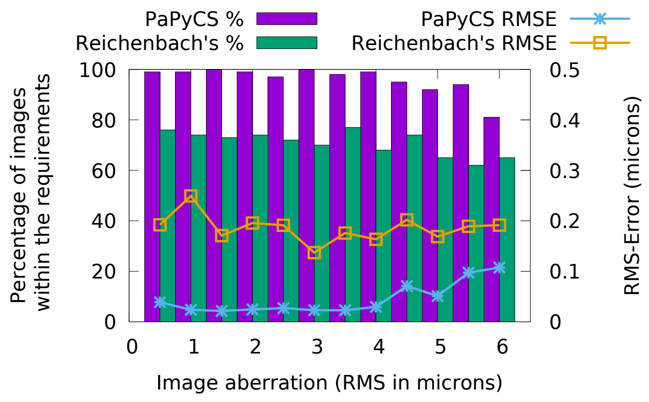


Figure 12: Bars (left-side Y-axis) show the percentage of H-S images with an RMSE smaller than 1. Lines (right-side Y-axis) show the average RMSE for the same images.

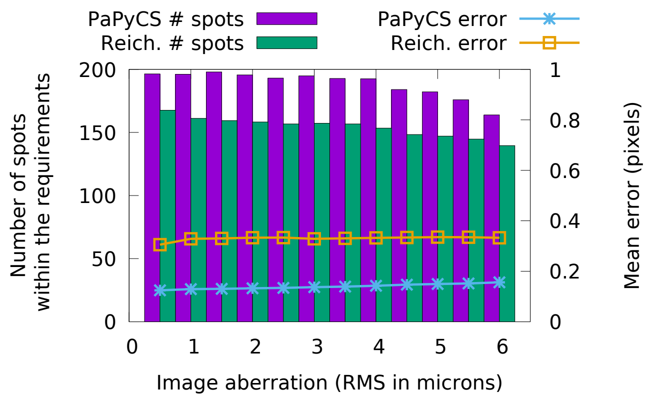


Figure 13: Bars show the number of centroids detected with an error in pixels smaller than 1 (left-side Y-axis). Lines show the mean error in pixels (right-side Y-axis).

Reichenbach's incurs 4x more computational error than PAPYCS. To give an idea of the magnitude of these errors, in [3] it was reported an RMSE of 0.15 microns as satisfactory for an Adaptive-Optics-corrected eye over a 4.8 mm pupil, similar to that used in our simulations.

Alternatively, Figure 13 shows the error spots detection expressed in pixels. Only those centroids localized with an error smaller than 1 pixel were considered. The bars show the amount of centroids detected within 1 pixel of their theoretical position. Besides, the average error (in pixels) of the centroid search is also plotted, being very stable through the set of images. PAPYCS shows an average error around 0.18 pixels, whereas Reichenbach's doubles this error, which significantly affects the accuracy of the measured aberration (as it was shown in Figures 11 and 12). As expected, the amount of *correctly* detected centroids (within 1 pixel of error) decreased as the aberration increased: for PAPYCS it went from 200 detected centroids, for small aberrations, to 165 centroids for a very strong aberration of 6 microns.

Finally, we evaluated both algorithms with actual H-S images obtained with a real Hartmann-Shack system (shown in Figure 6). For this experiment, as there was no reference image to compare against, an artificial eye was used and a series of actual H-S images were captured. Figure 16 shows the temporal variation of the 4th Zernike coefficient (which represents the *defocus* aberration) as measured by both PAPYCS and Reichenbach's algorithms. It can be observed that PAPYCS aberration measurements are noticeably more stable than those measured by Reichenbach's. The mean *Z4* value obtained by PAPYCS is -0.0478 and by Reichenbach's is -0.0493, while their standard deviation are 0.0011 and 0.0037 respectively. This experiment illustrates the robustness of our proposed PAPYCS algorithm against external artifacts and experimental variability typical in real systems.

4.4. Wavefront Map Calculation Performance

Although the wavefront map calculation is a final step which can be useful or even mandatory for certain applications, such as those involving a closed-loop adaptive optics system as explained in Section 3.4, it is typically an optional step, not needed for measuring an aberration which is perfectly characterized by a set of Zernike coefficients. For this reason we have separated this step results from those of the previous phases.

Wavefront map calculation is a very computing intensive operation, but it is also a trade-off between performance and resolution. In order to understand how this trade-off works, a wide variety of configurations have been tested. The images generated have two variable parameters: image size and number of Zernike coefficients used. A combination of both has been evaluated in Figure 14. The logarithmic scale was used to show the big difference in performance obtained by the CPU version and the GPU implementation. It can also been observed that performance rapidly decreases as either image size or the number of coefficients are increased.

Finally the speedup obtained by the GPU depending on image size and number of Zernike coefficients used, is shown in Figure 15. As it could be expected, the bigger

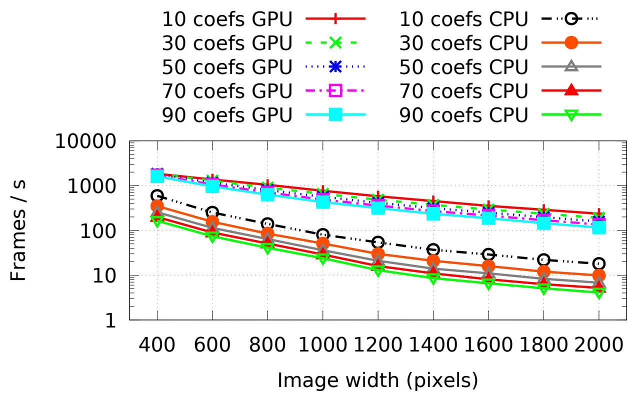


Figure 14: Wavefront map calculation scalability. X-axis corresponds image size. Each line plots a different number of coefficients for either the GPU or CPU.

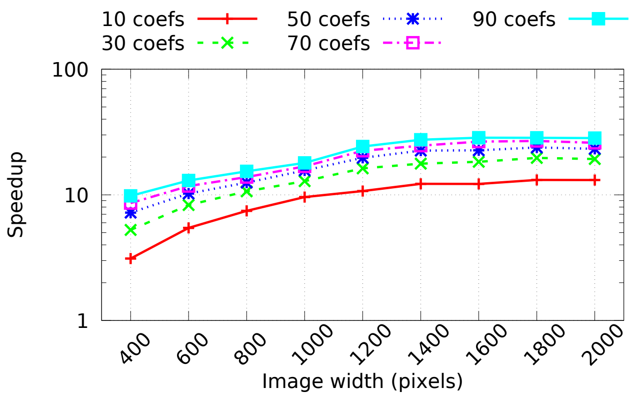


Figure 15: Speedup of the wavefront map calculation step, as a function of image size for vaying number of coefficients.

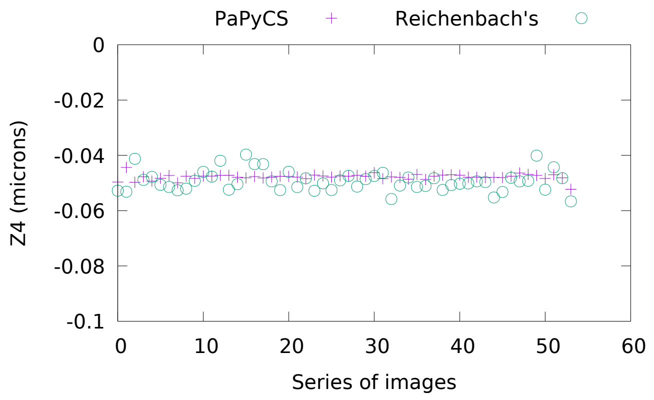


Figure 16: Measured defocus (in microns) for a series of images from a real H-S wavefront sensor system.

the image or the more complex the wavefront description, the higher the speedup.

5. Conclusions and Future Work

An accurate and high-speed GPGPU implementation has been developed for processing Hartmann-Shack images from human eyes in order to dynamically measure their wavefront aberration. All the necessary steps to automatically process H-S images have been parallelized: pupil determination, centroid search and Zernike coefficient fitting. Even the wavefront map calculation step has been parallelized to be able to integrate our approach in a closed-loop adaptive optics setup. While a sequential implementation is not capable of reaching a speed of 25 frames per second for real-time processing, unless resolution is severely limited, our setup and approach delivers 380 frames per second when processing H-S images of 1280x1280 pixels containing 250 spots. A comparison with two other state-of-the-art algorithms showed that PaPyCS is better suited to process H-S images from human eyes due to its higher resilience to system and image artifacts (corneal reflections, broken tear film, white noise) since more than 98% of the H-S images with an aberration smaller or equal than 4 microns were measured with an individual RMSE below 1 micron producing a mean RMSE lower than 0.05 microns.

Improvements are still possible, new synchronization methods between threads are now available in CUDA (Cooperative Groups), which could be used to improve the parallelism of some operations (e.g., thread group synchronization in the iterative expansion of the border of the Reichenbach's algorithm). In addition, some current GPUs include dedicated hardware to perform calculations on *halfs* which should speedup the wavefront map calculation. Since the data is already stored and transferred using halfs to save both memory and bandwidth, accuracy should not be a problem. Also, regarding our pupil tracking algorithm, some adjustments might be necessary with different optical configurations if the background light from the retina is very dim. Finally, our implementation is being integrated on a second system in the Laboratory of Optics of the University of Murcia and new challenges will appear, as they always do, when dealing with real-time systems used to measure human eyes.

6. Acknowledgments

This research has been supported by the European Research Council Advanced Grant ERC-2013-AdG-339228 (SEECAT), "Fundación Séneca", Murcia, Spain (grant 19897 /GERM/15), and the Spanish SEIDI under grants FIS2013-41237-R, and TIN2015-66972-C5-3-R, as well as European Commission FEDER funds. We would like to thank Dimitrios Christaras for proofreading the article.

References

[1] Justo Arines, Paula Prado, and Salvador Bará. Pupil tracking with a hartmann-shack wavefront sensor. *Journal of biomedical optics*, 15(3):036022–036022, 2010.

[2] G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000.

[3] Carmen Cánovas, Pedro M Prieto, Silvestre Manzanera, Alejandro Mira, and Pablo Artal. Hybrid adaptive-optics visual simulator. *Optics letters*, 35(2):196–198, 2010.

[4] M Carbillet, A Ferrari, C Aime, HI Campbell, and AH Greenaway. Wavefront sensing: from historical roots to the state-of-the-art. *European Astronomical Society Publications Series*, 22:165–185, 2006.

[5] José Francisco Castejón-Mochón, Norberto López-Gil, Antonio Benito, and Pablo Artal. Ocular wave-front aberration statistics in a normal young population. *Vision research*, 42(13):1611–1617, 2002.

[6] Emmanuel Chirre, Pedro M Prieto, and Pablo Artal. Binocular open-view instrument to measure aberrations and pupillary dynamics. *Optics letters*, 39(16):4773–4775, 2014.

[7] Alberto de Castro, Lucie Sawides, Xiaofeng Qi, and Stephen A. Burns. Adaptive optics retinal imaging with automatic detection of the pupil and its boundary in real time using shack-hartmann images. *Appl. Opt.*, 56(24):6748–6754, Aug 2017.

[8] Robert Duffner. *The adaptive optics revolution: a history*. University of New Mexico Press, 2009.

[9] Martin A Fischler and Robert C Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–395, 1981.

[10] J Hartmann. Bemerkungen über den bau und die justierung von spectrographen. *Zf Intrumentenk*, 20(4), 1900.

[11] Heidi Hofer, Pablo Artal, Ben Singer, Juan Luis Aragón, and David R Williams. Dynamics of the eye's wave aberration. *JOSA A*, 18(3):497–506, 2001.

[12] Rastislav Lukac and Konstantinos N Plataniotis. *Color image processing: methods and applications*. CRC press, 2006.

[13] José G Marichal-Hernández, Jose M Rodríguez-Ramos, and Fernando Rosa. Recuperación de fase de frente de onda atmosférico usando hardware gráfico. In *XI Congreso Nacional de Teledetección*, pages 569–572, Tenerife, Spain, September 2005.

[14] Steffen Mauch and Johann Reger. Real-time spot detection and ordering for a shack–hartmann wavefront sensor with a low-cost fpga. *IEEE Transactions on Instrumentation and Measurement*, 63(10):2379–2386, 2014.

[15] Steffen Mauch and Johann Reger. Real-time implementation of the spiral algorithm for shack-hartmann wavefront sensor pattern sorting on an fpga. *Measurement*, 92:63–69, 2016.

[16] Serge Meimon, Jessica Jarosz, Cyril Petit, Elena Gofas Salas, Kate Grieve, Jean-Marc Conan, Bruno Emica, Michel Paques, and Kristina Irsch. Pupil motion analysis and tracking in ophthalmic systems equipped with wavefront sensing technology. *Applied Optics*, 56(9):D66–D71, 2017.

[17] Jacopo Mocci, Martino Quintavalla, Cosmo Trestino, S Bonora, and Riccardo Muradore. A multi-platform cpu-based architecture for cost-effective adaptive optics systems. 01 2018.

[18] V Molebny. Wavefront sensors. In Pablo Artal, editor, *Handbook of Visual Optics, Vol. II: Instrumentation and Vision Correction*, pages 17–36. CRC Press, 2017.

[19] Juan Mompeán, Juan L. Aragón, Pedro M. Prieto, and Pablo Artal. Design of an accurate and high-speed binocular pupil tracking system based on gpgpus. *The Journal of Supercomputing*, 74(5):1836—1862, May 2018.

[20] K Nienhuis and BRA Nijboer. The diffraction theory of optical aberrations: Part iii: General formulae for small aberrations; experimental verification of the theoretical results. *Physica*, 14(9):590–608, 1949.

[21] BRA Nijboer. The diffraction theory of optical aberrations: Part i: General discussion of the geometrical aberrations. *Physica*, 10(8):679–692, 1943.

[22] BRA Nijboer. The diffraction theory of optical aberrations: part ii: diffraction pattern in the presence of small aberrations. *Physica*, 13(10):605–620, 1947.

[23] CUDA Nvidia. Cublas library. Accessed: 2017-06-05.

[24] Nobuyuki Otsu. A threshold selection method from gray-level histograms. *IEEE Transactions on Systems, Man, and Cybernetics*, 9(1):62–66, 1979.

[25] Alexander Pichler, Pierre Raymond, and Marc Eichhorn. Fast wavefront sensing using a hardware parallel classifier chip. *Applied Physics B*, 115(3):325–334, 2014.

[26] Jason Porter, Hope Queener, Julianna Lin, Karen Thorn, and Abdul AS Awwal. *Adaptive optics for vision science: principles, practices, design and applications*, volume 171. 2006.

[27] Pedro M Prieto, Fernando Vargas-Martın, Stefan Goelz, and Pablo Artal. Analysis of the performance of the hartmann–shack sensor in the human eye. *JOSA A*, 17(8):1388–1398, 2000.

[28] Marc Reichenbach, Ralf Seidler, Benjamin Pfundt, and Dietmar Fey. Fast image processing for optical metrology utilizing heterogeneous computer architectures. *Computers & Electrical Engineering*, 40(4):1158–1170, 2014.

[29] Roland V Shack and Ben C Platt. Production and use of a lenticular hartmann screen. *Journal of the Optical Society of America*, 61(5):656–660, 1971.

[30] Zernike von F. Beugungstheorie des schneidenver-fahrens und seiner verbesserten form, der phasenkontrastmethode. *Physica*, 1(7-12):689–704, 1934.

[31] Yongxin Yu, Tianjiao Zhang, Alexander Meadway, Xiaolin Wang, and Yuhua Zhang. High-speed adaptive optics for imaging of the living human eye. *Optics express*, 23(18):23035–23052, 2015.

[32] Yongxin Yu and Yuhua Zhang. Dual-thread parallel control strategy for ophthalmic adaptive optics. *Chinese Optics Letters*, 12(12):121202–121202, 2014.