The Journal of Supercomputing
https://doi.org/10.1007/s11227-022-04332-7

Energy-efficient design of a presbyopia correction wearable powered by mobile GPUs and FPGAs

Juan Mompeán[1,2] · Juan L. Aragón[2] · Pablo Artal[1]

Accepted: 20 January 2022

Abstract

This paper presents an energy-efficient design and evaluation of a novel portable device for the automatic correction of presbyopia in human eyes driven by the use of opto-electronic lenses and based on the dynamic pupil response of the subject. Due to the *wearable* nature of the proposed Dynamic Auto–Accommodation Glasses, in addition to the real-time requirement, an energy-efficient implementation is critical for the success of the device. In this work, the binocular pupil tracking of a subject, followed by the calculation of the eyes' vergence, and the control of a pair of opto-electronic lenses are implemented and evaluated on several hardware platforms, including two mobile GPU/SoCs, a high-end FPGA, a low-cost FPGA, and a desktop GPU (as a reference point). The pupil tracking algorithm has been parallelized, applying different platform-specific optimizations for each case, to design a fast yet energy-efficient wearable. The hardware platforms have been evaluated to determine which one is the most appropriate for the presbyopia correction task. The experimental results show that the most energy-efficient platform is a mobile GPU (Samsung Exynos 8890) capable of processing frames at 0.016 Joules/frame, still allowing real-time processing (24 frames/sec).

Keywords GPU · FPGA · Image processing · OpenCL · Presbyopia · Real time

✉ Juan Mompeán
juan.mompean@um.es

Juan L. Aragón
jlaragon@um.es

Pablo Artal
pablo@um.es

[1] Laboratorio de Óptica, IUiOyN, Universidad de Murcia, Murcia, Spain

[2] Departamento de Ingeniería y Tecnología de Computadores, Universidad de Murcia, Murcia, Spain

Published online: 16 February 2022

1 Introduction

Nowadays, there are many devices fitting in the category of wearables that are designed to target many different purposes such as monitoring/analyzing body signals or sensing ambient data in real time to provide immediate feedback to the user (e.g., heart rate monitoring, diabetes control, etc). Wearables have exploded in the recent years and a plethora of devices can be found in the market. They have become so popular that major semiconductor companies have developed specific low-cost and power-efficient processors for this particular market, such as the Cortex-M series from ARM [26], aimed at developing more complex and powerful wearable solutions.

This paper proposes the energy-efficient design and evaluation of a novel wearable device for the automatic correction of *presbyopia*, which is the reduction of the accommodation range of the human eye that disables us to focus near objects. This is accomplished by using a pair of opto-electronic lenses which are driven by the dynamic pupil response of the subject's eyes (the subject is the person wearing the glasses). The wearable nature of the proposed Dynamic Auto–Accommodation Glasses, in addition to the fluent response that is needed to be optically comfortable, make the device's energy efficiency and performance two critical implementation challenges.

In this work, we have implemented the three key tasks of the system (namely, the subject's binocular pupil tracking, the eyes' vergence determination, and the opto-electronic lenses' control) on several hardware computing platforms aiming at designing a fast yet energy-efficient device. It is important to note that our working prototype for the automatic presbyopia correction is a device that demands more computing power than most of the currently available wearables. This is due to the hardware involved (two cameras and two opto-electronic lenses) and the real-time image processing that has to be performed. Therefore, it is expected a higher power consumption than that of simple monitoring wearables. As wearables are powered by batteries, a high power consumption severely limits the autonomy of the device before recharging its battery. In our case, due to the cordless nature of the presbyopia correction glasses, having a long-lasting battery is critical for the success of the device.

The most compute-intensive part of the proposed presbyopia correction device is the binocular pupil tracking, which involves heavy image processing and some other complex operations. Therefore, our implementation is primarily focused on improving the energy efficiency of this task. In particular, we have evaluated two mobile GPUs/SoCs (a Samsung Exynos 8890 and a Qualcomm Snapdragon 650), two FPGAs (a low-cost Xilinx ZedBoard and a high-end Xilinx Alveo u250), and a high-end desktop GPU (NVIDIA 980 GTX) in order to determine the most energy-efficient platform. Running compute-intensive operations on battery-powered devices is challenging, and choosing the right computing platform is important to achieve the highest autonomy for the device. Furthermore, efficiently programming FPGAs to obtain their maximum performance and energy efficiency is still a challenge. Although, the usage of OpenCL to program FPGAs may ease the implementation effort, applying platform-specific and energy-oriented optimizations is still critical to achieve the maximum battery autonomy and performance [15].

The device performance is another key factor to achieve a truly smooth optical feeling for the subject. The eyes of young people accommodate very fast when switching their focus from far to near distance, and vice versa. Therefore, a similar response has to be provided by our Dynamic Auto–Accommodation Glasses (as analyzed in Sect. 4.3). To achieve such a fast response, the performance has been improved by parallelizing the pupil tracking algorithm and enabling platform-specific optimizations (refer to Sect. 5 for further implementation details). Mobile SOCs are becoming more powerful and they are capable of executing compute-intensive applications, provided they are properly optimized [1, 27].

On the other hand, presbyopia is an eye condition that affects everybody. As such, different solutions have been developed to reduce the lack of accommodation of the eye. The most common solution is the utilization of multifocal or progressive spectacles. Fixed-distance glasses for near work are also popular, but also the utilization of contact lenses or intra-ocular lenses (although the latter are most commonly used to replace lenses with cataracts, there are intra-ocular lenses specifically designed for presbyopia). All those are good solutions in general, however, all of them present drawbacks and limitations. Some of the aforementioned treatments are invasive such as intra-ocular lenses or refractive surgery. Non-invasive solutions also present trade-offs. E.g., progressive spectacles enhance either far or near distance at the expense of reducing image contrast. In addition, the available focus points are limited, so the user experience is limited as well. Differently, the proposed Dynamic Auto–Accommodation Glasses are non-invasive and offer a dynamic continuous focus, providing a smooth optical experience similar to the human lens.

The rest of the paper is organized as follows. Section 2 presents some background and reviews some relevant literature. Section 3 describes some of the technologies and works related to this paper. Section 4 explains the system's design and behavior. Section 5 explains the implementation details and the platform-specific optimizations. Section 6 presents the performance and energy-efficiency results for the different computing platforms, as well as the discussion of those results. Finally, Sect. 7 summarizes the major conclusions of the work.

2 Background on presbyopia

Presbyopia is the reduction of the accommodation range of the human eye as a result of aging that disallows the capacity to focus near objects. The accommodation range of the human eye defines how close it can focus an object, and it ranges from more than 10 diopters for a child, to about 0 diopters for a 60 year old person. Or expressed in distance, a child can clearly focus an object at 10 cm, while a 60 year old person can only focus properly at the infinity, unless he or she is myopic. A large body of research can be found in the Optometry and Ophthalmology literature. Measuring the human lens and the muscles involved in accommodation is extremely complicated and such complexity is the reason why some aspects of presbyopia are not completely known yet. Fisher, and Glasser and Campbell [10, 11] observed that the loss of elasticity of the eye's lens is the main factor of the decrease in the accommodation range. The eye's lens loses its elasticity in a natural way due to aging. Another study by Heys et al. [12] described a massive increase in the stiffness of the nucleus of the eye's lens, which might be an important factor in presbyopia.

The accommodation range of the human eye continuously decreases through our lives. Children can usually accommodate more than 10 diopters, which means that they can focus targets as near as 10 cm [25]. However, the accommodation range of the human eye keeps decreasing until the late fifties [7] or sixties [12]. Although presbyopia starts developing at early ages and keeps developing throughout a person's life, it remains unnoticed until the forties, when near targets start to appear blurred. At the fifties, presbyopia is already limiting our capabilities to focus near targets making it difficult to read small texts. I.e., when the text is far enough to be focused, it is too small to be readable. And when the text is near enough to have an appropriate size, it cannot be focused by a presbyopic eye. However, myopic people are still able to focus near targets thanks to their myopia condition.

3 Related work

3.1 Tunable lenses

Our Dynamic Auto-Accommodation Glasses rely on the use of a pair of tunable opto-electronic lenses which can change their focal length dynamically. This change might be accomplished with different methods. The capability of dynamically changing the focal length is a very useful property in different fields such as machine vision, microscopy or refraction correction. For our glasses, we have selected the Optotune tunable lenses [6] that are polymer-based lenses with a high focal length range, going from −100 to +100 mm (or analogously, from −10 to +10 diopters).

3.2 Eye trackers

Eye tracking is widely used in many fields, from research to marketing. Many different approaches have been developed for such purpose, including thresholding techniques, labelling algorithms, usage of the Hough transform and template matching. A survey in [2] compares the most common approaches to perform eye tracking. Because of the high interest in eye tracking, there are different commercial devices from several companies such as Tobii, SMI, Eyelink, ISCAN, or Pupil Labs.

Even though we have used an image processing-based eye tracking approach in this work, other ways of performing pupil tracking might be useful in wearable devices. A novel method for eye tracking has been developed by Mastrangelo et al. comprising a small number of LEDs and light sensors to reduce both the size of the tracking device and the power required to run it [17].

3.3 Integrated wearable solutions

In [18] we presented an early and rather bulky proof-of-concept prototype for presbyopia correction. It was a large optical set-up sitting on a 2 m^2 benchtop and driven by a high-end desktop PC and a high-performance GPU. This early setup used a high-speed camera to image both eyes at 150 frames per second and a resolution of 1280 × 1024 pixels. To process such a big amount of data, a PCI-based GPU was needed. The major drawback of this early proof-of-concept system was the fact that it was attached to an optical benchtop, therefore, not being portable nor autonomous. Contrarily, the device evaluated in this paper is fully portable thanks to the smartphone where all the processing is performed and the control is executed. A previous work focused on the evaluation of the optical and visual performance [20], however, the current paper focuses on the hardware implementation and the energy efficiency of the system evaluating different computing platforms with the goal of creating the best possible prototype with enough performance and battery life.

Two other works have presented prototypes for presbyopia correction with opto-electronic lenses devices [13, 22]. The work in [13] focuses on the design of an opto-electronic lens with a big aperture of 20 mm and a range of 3 diopters. They have integrated their lens with a distance sensor to change their focal length according to the distance where the subject is looking at. However, this design has a main drawback: if the user eyes are not looking straight, the optical power applied by the lenses could be wrong. Our presbyopia correction glasses, instead, calculate the vergence of the eyes to avoid this problem. The device presented in [22] uses a commercial pupil tracker, a depth sensing system and a couple of opto-electronic lenses. However, to the best of our knowledge, their system is not portable, which is the most important goal of our presbyopia correction glasses.

4 Dynamic auto–accommodation glasses

These glasses are a new invention which disrupt with existing solutions for presbyopia. This section describes how they are built, the control algorithm and shows some tests of their functioning. Understanding the device implementation is important to understand the challenges associated to developing a fully functional solution. Looking at the device is easy to understand the need for portability and comfortability. In the same way, the control algorithm shows how important is to have a fast and reliable pupil tracking system to update the correction.

4.1 Glasses frame

A great effort has been made to build our presbyopia correction device. It integrates many parts within a compact custom frame to enhance portability. Achieving such integration is challenging due to the reduced space available, but also because of the distance between the eyes and the cameras. Tiny cameras and

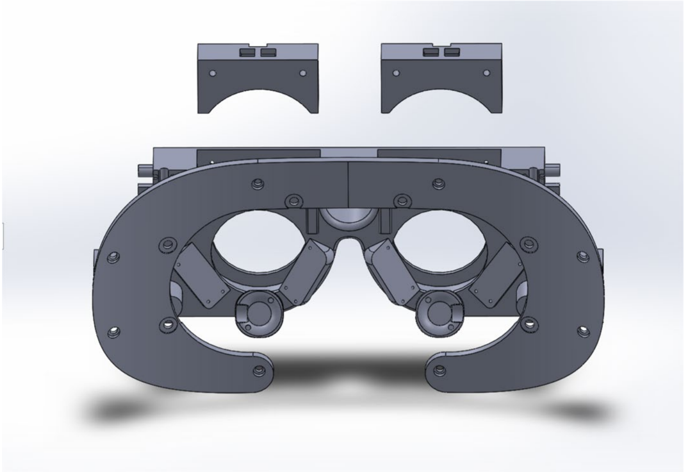


Fig. 1 A 3D model of the frame used for the Dynamic Auto-Accommodation Glasses where the opto-electronic lenses, the drivers of the lenses and the cameras will be integrated

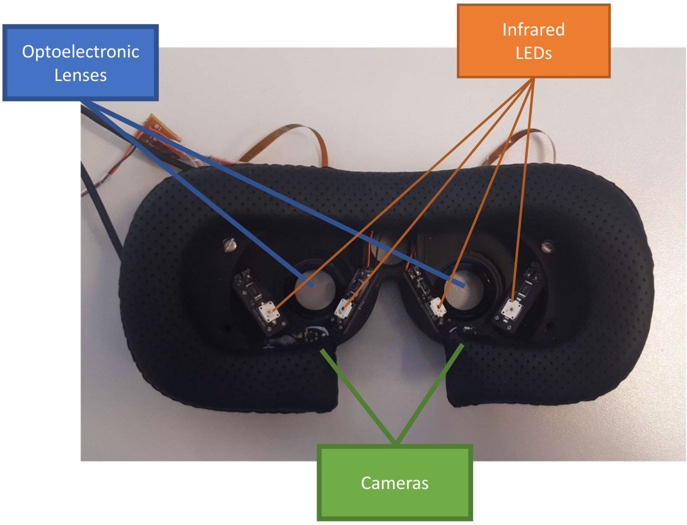


Fig. 2 Actual 3D-printed frame covered with a soft pad for better comfort and stability. This prototype includes the opto-electronic lenses and the wiring

infrared LEDs have been used to achieve the desired integration. Furthermore, a micro-USB hub has been installed to connect the smartphone with the two drivers of the opto-electronic lenses and the two USB cameras. As a result, the device is small and relatively light. Figure 1 shows a 3D model of the frame whereas Fig. 2 shows the actual prototype with the soft pad, the opto-electronic lenses, the cameras and all the connections.

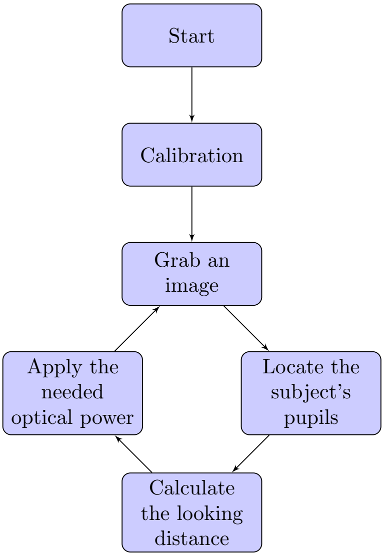
The cameras are compatible with the UVC standard (USB Video Class) which is not natively compatible with Android, so a library has been used to drive them: *UVCCamera* [24]. The Optotune lenses are controlled through the COM port. In order to connect and communicate with the COM port in Android, the *UsbSerial* library [8] has been used.

4.2 Glasses control

A high-level scheme of the algorithm used to control the presbyopia-correction glasses is shown in Fig. 3. Initially, a one-time calibration step is performed: the subject is asked to look straight at a far target and his/her eyes' position is recorded. As soon as the calibration is finished (which takes <0.2 sec) the correction loop begins. Images are grabbed continuously by the cameras, which are then processed to locate the pupils on them. When the pupils' positions are known, the eyes' vergence is calculated and the distance where the subject is looking at is obtained. Finally, depending on this distance, the required optical power is applied by the opto-electronic lenses. This loop is repeated until the correction phase is stopped by the user.

An Android application has been developed to operate the glasses and execute the control algorithm. This application includes different configuration parameters to allow the user to run the calibration protocol, control the camera brightness or check the pupil tracking performance. Figure 4 shows a screenshot of the application, where it can be seen a preview of both cameras while running the pupil tracking algorithm which highlights the located pupils of the subject (in green). This screen also shows a plot of some pupil tracking and correction information over the last few seconds to check the system behavior.

Fig. 3 Diagram of the algorithm used to control the device


Start
Calibration
Grab an image
Locate the subject's pupils
Calculate the looking distance
Apply the needed optical power

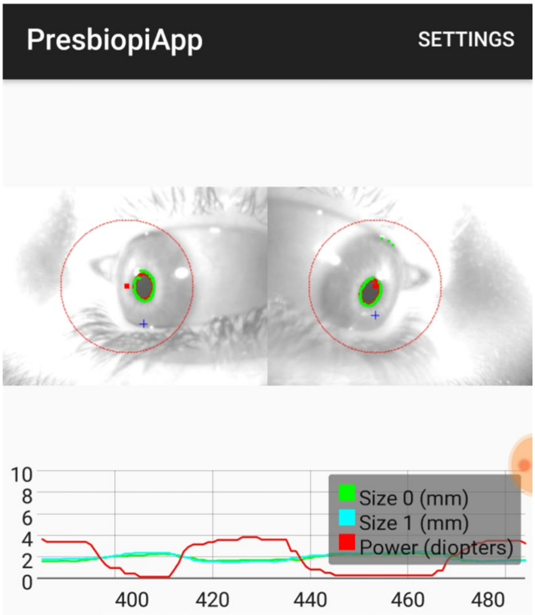


Fig. 4 Screenshot of the developed Android application to control the Dynamic Auto-Accommodation Glasses. The located pupils of the subject and a plot of some measured parameters over the last few seconds are shown

4.3 System behaviour and validation

A deep integration and a correct synchronization of all the steps is important to achieve a smooth response to the eyes' movements in order to guarantee a smooth user experience. To illustrate the response time and behavior of the Dynamic Auto-Accommodation Glasses, a use case example is shown in Fig. 5. A subject was asked to alternatively look at a far and at a near target. The Figure shows the change in the eyes vergence (i.e., the distance between the two pupils – green line) and the corresponding diopters applied by the opto-electronic lenses (purple line). It can be observed that when the eyes are closer (lower values for the

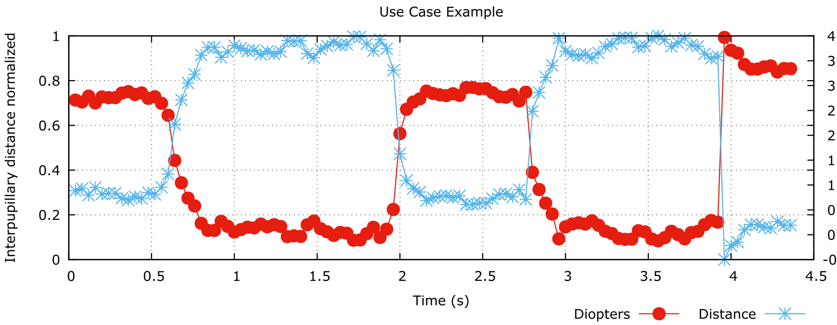


Fig. 5 Use case example for a subject looking at far and near alternatively. The green line shows the distance between the two pupils while the purple line represents the diopters applied by the opto-electronic lenses as a response to the movement of the pupils

inter-pupillary distance) the glasses react by applying the diopters needed by the subject. Similarly, when the two pupils are farther away from each other (higher values for the inter-pupillary distance) the applied diopters are reduced accordingly.

In order to validate that the algorithm used to calculate the position where the subject is looking at is working correctly, a validation process has been performed. Two artificial eyes were used to simulate the movement of the eyes of a subject. The artificial eyes were placed on top of a rotation mount with a 2-degree precision. Several positions have been tested for each eye, including positions with both eyes at the same angle, and positions with each eye at different angle. The calculations of the current rotation of the eye are performed as described in the patent in [5]. Figure 6 plots both the *expected* and the *measured* angle in degrees. It can be observed how well they both correlate, and therefore, the high accuracy of the calculations.

5 Energy-efficient implementation of pupil tracking

Several computing platforms have been tested and evaluated in order to determine which one performs better for the presbyopia-correction application, paying special attention to their respective energy consumption in order to select the most efficient one for a wearable design. The evaluated platforms include two mobile GPU/SoCs (integrated in commercial smartphones), a low-cost FPGA, a high-end FPGA and, finally, a high-end desktop GPU (to be used as a reference point).

As mentioned before, in terms of computing, the most demanding task is tracking both pupils. Therefore, we describe first the utilized pupil tracking algorithm to later focus on the implementation details to provide a highly parallel implementation of the pupil tracking algorithm for each computing platform.

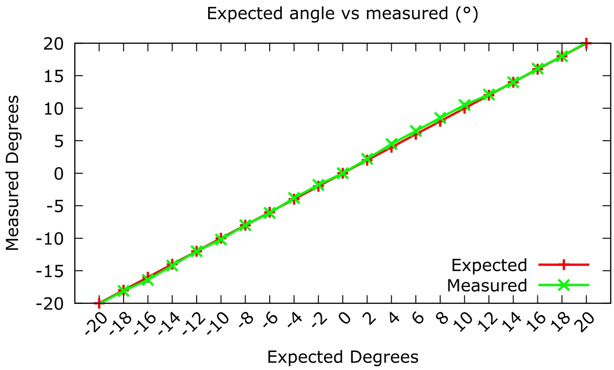


Fig. 6 Accuracy of the eye angle calculation performed by the presbyopia-correction glasses using an artificial eye. A comparison between the real angle and the measured one is shown

5.1 Pupil tracking algorithm

The Starburst algorithm [16] has been used for our pupil tracking phase. This algorithm was parallelized for high-end NVIDIA desktop GPUs using CUDA in [19]. However, in this paper, we present a novel implementation using OpenCL, due to the incompatibility of CUDA with the evaluated mobile platforms, and focusing on optimizing the energy efficiency of the device rather than just focus on the performance, as it was the case of the work in [19].

Algorithm 1 Pupil Tracking Algorithm Pseudocode

```
 1: procedure PUPILSEARCH($Image, PrevCenter$)
 2:     $Image \leftarrow Preprocessing(Image)$
 3:     $Distance \leftarrow \infty$
 4:     while $Distance > ConvergenceLimit$ do
 5:         $Points \leftarrow SearchPoins(Image, PrevCenter)$
 6:         $Pupil \leftarrow RANSAC(Points)$
 7:         $NewCenter \leftarrow Center(Pupil)$
 8:         $Distance \leftarrow Distance(PrevCenter, NewCenter)$
 9:         $PrevCenter \leftarrow NewCenter$
10:     end while
11:     return $Pupil$
12: end procedure
```

The pseudocode of the algorithm used to perform the pupil tracking is shown in Algorithm 1. The main processing tasks to locate the pupils are: the preprocessing of the input image, the search of points that correspond to the border of the pupils, and the fitting of those points to ellipses. A detailed explanation of those tasks follows.

1. A preprocessing phase (line 2 in algorithm 1) is performed to identify the corneal reflections introduced by the LEDs.

    (a) A *top-hat* transform is applied to the image. This operation intends to exacerbate the corneal reflections to more precisely identify them. In particular, it returns an image containing the elements that are smaller than the used kernel size.

    (b) The result of the top-hat transform is *thresholded* to select only the corneal reflections. However, after applying the thresholding operation, the area covered by the detected corneal reflections may be smaller than their actual size.

    (c) Therefore, a *dilation* operation is applied to ensure that the whole corneal reflection is selected.

2. The border points of the pupil are searched (line 5 in algorithm 1) as follows:

(a) First, an initial point is provided. This initial point is usually the center of the pupil detected in the previous frame. Otherwise, the geometric center of the image is used.

(b) Then, an initial set of points is searched in the image. For that purpose, a set of "rays" going from the initial center to the image borders are processed. These "rays" are evenly distributed around 360°. A pixel along one of these rays is identified as a potential pupil border if the gradient at that point is bigger than a threshold. Pixels in the border of the pupil usually have high gradient values due to the change in brightness between the pupil (blackish) and the iris (grayish).

(c) A second wave of "rays" is generated from the points found in the previous step. However, this new wave is narrower than before (100°). Also, the search direction goes from the points found in (b) towards the center provided in (a). The same procedure as in (b) is used to identify new potential pupil border points. This step intends to reinforce the results from (b). I.e., if a good point was found in (b), the "rays" generated in this step will be directed towards the other side of the pupil. As a result, more pupil border points can be found.

(d) The center of mass of the pupil border points located in (c) is calculated and its position is compared with the previous center. If the difference is smaller than a given threshold, the algorithm has converged and it continues to the third stage. Otherwise, the search is repeated from step (b) but using the calculated center of mass as the new search center.

3. Finally, a RANSAC (Random Sample Consensus) ellipse fitting (line 6 in algorithm 1) is applied to the complete set of border points and the best fitting ellipse is selected as the final pupil. The RANSAC algorithm is a powerful approach when performing a fitting that includes noisy data points since the latter can be ignored in order to fit only the correct data points [9].

As it can be noticed, pupil tracking is a very computing intensive task. In particular, the Starburst algorithm shows high computing demands in stages (2) and (3). The RANSAC also performs many calculations of distance for fitting to an ellipse which is an expensive operation. Likewise, the preprocessing phase requires applying some computer vision functions which process the complete image pixel by pixel. The preprocessing and the Starburst algorithm are memory-bound tasks, they have a low ratio of calculations per memory access. While the RANSAC has a higher ratio of calculations per memory access mainly due to the ellipse fittings. Of course, depending on the resolution of the images the computing required to process the images will vary.

Next, we describe the different hardware computing platforms we have evaluated and the specific optimizations we have used on each case.

5.2 Case 1: Mobile GPU implementation

A specialized *novel* implementation using OpenCL [21] has been programmed for mobile GPUs (also referred to as the smartphone implementation in the paper). Smartphones are heterogeneous computing devices integrating general purpose cores (CPUs), graphics processors (GPUs) and many other specialized hardware units, such as DSPs or AI specific accelerators. Our smartphone implementation has focused on using the GPU to run the binocular pupil tracking algorithm, whereas the control of the cameras and the opto-electronic lenses is performed on the CPU cores. The GUI of the Android application is also controlled by the CPU. Since the highest compute-intensive task is the execution of the pupil tracking algorithm, only this part has been taken into account for the performance and energy-efficiency measurements reported in Sect. 6.

Two smartphones have been evaluated, one of them is the Samsung Galaxy S7 (running Android 7.0) which integrates a Samsung Exynos 8890 SoC and supports OpenCL 1.2. The Exynos 8890 incorporates a Mali-T880 GPU which has a peak rate of 265 GFlops [4]. The other is a Xiaomi Redmi Note 3 Pro (running Android 6.0) which integrates a Qualcomm Snapdragon 650 SoC and supports OpenCL 2.0. The Snapdragon 650 incorporates an Adreno 510 GPU which has a peak rate of 154 GFlops In order to run C++ and OpenCL codes in Android, the NDK kit (Native Development Kit 18b) has been used.

Several optimizations have been applied to the OpenCL implementation developed for GPUs. Those optimizations are: vectorized memory accesses, local memory usage, native math operations (sine, cosine, etc.), kernel fusion, and reduced-precision floating point data types where possible. Those optimizations are described next.

Vectorized memory accesses

They improve the effective bandwidth of the memory by means of fusing several small load operations into a bigger one. Increasing the effective memory bandwidth results in an increase in performance due to having more data available to process. This kind of optimization works best with algorithms that load continuous data in a loop and process each element independently. In those cases, several iterations of the loop can be unrolled into a single one and the data can be loaded in a single wider operation. This is a common optimization in image processing tasks. We have used it for the top-hat transform, the thresholding and the dilation operations. In addition, the coordinates of the located pupil border points are stored using packed data types, such as `float2`.

Local memory

It is a small fast on-chip memory with a much higher bandwidth than the main device memory. Because of its smaller size, it is commonly used for very specific purposes. E.g., it can be very useful on algorithms that load several times the same data for different threads (or the same ones). But, *local memory* is only shared by threads within the same work group. In our case, this optimization has been used for the image processing operations, the RANSAC and in the search of the pupil border points. An important point about the usage of *local memory* on the Mali GPU evaluated on this paper should be made. The architecture of the Mali Midgard maps local memory to global memory, therefore, it is supported but there is not performance benefit from using it [3]. However, to have a homogeneous and future-proof code we have decided to keep this optimization in all the implementations.

Native operations

They are specialized math functions optimized for performance at the expense of losing some arithmetic precision. We have employed them for calculating the angles during the search of the pupil border points and to perform the distance calculations. In particular, we have used: `native_sin`, `native_cos` and `native_sqrt`.

The half data type

This is a floating point data type that uses 16 bits to store its information. As such, its range and precision is lower than `floats` and `doubles`. However, it allows for more calculations to be performed simultaneously and more `half` elements can be accessed consuming the same memory bandwidth. We have used this optimization to improve the search of the pupil border points and in the RANSAC algorithm.

Kernel fusion

It consists of joining two kernels which are executed one after another. However, to obtain a benefit from the fusion there must be some data which is stored by the first kernel and later accessed by the second kernel. By fusing kernels, the load/store operations to the main device memory can be avoided improving performance. Furthermore, kernel fusion has a second advantage when the kernel overhead has a relatively big weight, removing the launch of a second kernel completely removes its overhead. That is specially important for mobile SoCs [14]. This optimization is applied to the top-hat transform and the thresholding operation.

Separable filters

The final optimization, we have used in our GPU implementation is known as *separable filters*. It has been used to optimize the image processing functions leading to a further reduction of the execution time.

To measure the power consumption, the app *Trepn Power Profiler* [23] by Qualcomm has been used, which can monitor several parameters of a mobile phone. In order to only account for the time of the processing (pupil tracking), our App notifies *Trepn profiler* when the pupil tracking has started and when it has finished. To exclude the power of the screen and other potential power drain sources in the smartphone, a *base* power measurement is done with the device in an idle state as follows: during 30 seconds the phone's power dissipation is recorded and the average is used as the base (or idle) power to be later removed.

5.3 Case 2: FPGA implementation

Our second case, an FPGA implementation has been tested on both a high-end Xilinx Alveo u250 and a low-cost Xilinx ZedBoard. The Alveo u250 has 1728K LUTs (Look-up Tables) and 3456K registers, while the ZedBoard used (Z-7020) has 53K LUTs and 106K registers. Xilinx offers a wide variety of programming alternatives for their devices, including: C/C++, OpenCL and HLS. We have used OpenCL 1.0 and synthesized using the Xilinx SDX 2018.3 development environment. Since the same programming language (OpenCL) has been used for all our implementations, a lot of the code is shared among all of them. Although specific optimizations have been applied to each computing platform where it was possible. In fact, some of the optimizations are used in both computing platforms but employed in a different way due to the big differences between the platforms. For the FPGA implementation, local memory, loop unrolling, pipelining, and array partitioning have been used by including specific `pragmas` developed by Xilinx for their FPGA products. Other optimizations are used in a similar way as for the Smartphone/GPU implementation, like *kernel fusion* to avoid OpenCL API calls overhead and unnecessary memory operations; or the usage of *vectorized data types* for storing the pupil border locations; or *native math operations* which can potentially increase the performance and are also available for Xilinx's FPGAs. Specific details about the FPGA optimizations, we have implemented follow.

Local memory

Similarly to its equivalent in GPUs, local memory is a close, low latency, high-bandwidth and small memory which might be used for caching data that is reused through the computation. Local memory has been used in the image processing functions (erosion, dilation, and gauss filters) and in the RANSAC. Contrarily to the GPU implementation, the local memory is statically allocated, so its size its known at compile time and the compiler can optimize the generated hardware to take full advantage of it. Only the filter size provided at compile time can be used by the synthesized hardware, but that is a common trade-off on FPGAs. Just like in GPUs, the local memory is a scarce resource, and an FPGA can easily run out of it.

Loop unrolling

This optimization is crucial in FPGAs for an improved performance since the unrolled loop iterations can run in parallel if there are enough resources (logic cells) in the FPGA. However, it is needed to know the total number of iterations at compile time. We have implemented loop unrolling in the image processing functions, which already had a fixed size at compile time after applying the previous optimization. Loop unrolling is enabled by adding a pragma before the loop to unroll: `#pragma` `__attribute__((opencl_unroll_hint))`. It can be applied to the loops that load the data into shared memory and to the loops that process the data. However, loop unrolling has to be used cautiously since it dramatically increases the usage of hardware resources.

Pipelining

This optimization allows for a much higher concurrency. It is a widely used feature (in many fields) because it is a very effective technique to increase resource utilization. On FPGAs it is commonly used in loops: as soon as the first iteration has completed its first instruction, the second iteration starts and so does the third iteration, when the previous one moves a step forward. In our implementation, pipelining has been used for the search of the pupil border points with a significant performance gain. This optimization is applied in a similar way as *loop unrolling*, a pragma is added before the pipelined loop: `#pragma  __attribute__((xcl_pipeline_loop))`. Note that enabling this optimization does not consume more hardware resources, as opposed to *loop unrolling*.

Array partitioning

This optimization increases the bandwidth that might be used to load data from local memory. This is especially useful in the image processing functions where a heavy usage of the local memory has been implemented. Array partitioning splits the local memory in several smaller arrays and increases the number of read and write ports. As a result, the bandwidth is significantly increased for the partitioned array. An example of this optimization is shown in the following code example where a cyclic partitioning is employed.

Listing 1 Array partitioning example.

```
local uchar lmem[WIDTH + 2 * EROSION_RADIUS + 1] __attribute__((
    xcl_array_partition(cyclic, 8, 1)));
local uchar out_lmem[WIDTH] __attribute__((xcl_array_partition(
    cyclic, 8, 1)));
```

Max memory ports

This option has been selected to allow using different ports for different global memory buffers, thus enabling parallel memory operations. Without this option only one port is available to load/store data to global memory, serializing the memory operations and reducing the overall performance.

The image processing tasks of the preprocessing step have achieved a significant performance boost through the combination of *Local memory* + *Loop unrolling* + *Array partitioning* + *Max Memory Ports* optimizations. As we will report in the

Results Section, when comparing the non-optimized initial version with the optimized one, the speedup is 17.0x for the ZedBoard and 17.2x for the Alveo u250.

The pupil border search step has also improved its performance with the combination of *Pipelining* + *Max Memory Ports* optimizations. Comparing the non-optimized initial implementation with the optimized one, the speedup is 10.7x for the ZedBoard and 9.2x for the Alveo u250.

Finally, the power consumption of the several synthesized implementations for each FPGA is the one reported by the Xilinx Vivado environment.

5.4 Case 3: Desktop GPU implementation

As a final case, a high-end desktop GPU has been included in the evaluation of computing platforms, mainly as a reference point, even though its size and power consumption make this platform unfeasible as a wearable solution. In particular, we have evaluated a NVIDIA 980 GTX mounted in a desktop PC. This implementation uses again OpenCL and has been compiled with CUDA 10.1. The OpenCL code here is very similar to the one used in the smartphone implementation (Sect. 5.2). The main difference is the replacement of the `half` data type by the `float` data type because the former is not supported on this GPU (desktop GPUs support for `halfs` has not been very wide until recently, when mainstream graphics cards from NVIDIA started to include half-precision compute capabilities). Overall, desktop GPUs are much more powerful than mobile GPUs. Their performance is much greater but also their power consumption (that we have measured using the command line tool `nvidia-smi`).

5.5 Performance and efficiency analysis methodology

In order to fairly evaluate the three hardware platforms, the same input images have been used across all the platforms. A video of 250 frames recorded with the Dynamic Auto-Accommodation Glasses at a resolution of 1280 × 960 has been used to evaluate the performance and energy efficiency. This video includes one blink and eye movements to different positions, so it is representative of the normal behavior of an eye. To include the variability from the different images in the results the whole video is processed and the average time for processing a frame is calculated. Even though the video was recorded at the highest (native) resolution of the cameras, we are also interested in evaluating lower resolutions. Therefore, the video has been downscaled to 640 × 480 and 320 × 240 pixels. The lower-resolution images are needed to enable real-time processing on some of the tested devices (e.g., the low-cost FPGA). The videos at different resolutions have been used to evaluate the different platforms: a Samsung Exynos 8890 (mobile GPU), a Qualcomm Snapdragon 650 (mobile GPU), a Xilinx ZedBoard (low-cost FPGA), a Xilinx Alveo u250 (high-end FPGA), and a NVIDIA 980 GTX (desktop GPU).

Finally, even though the *control* of the Dynamic Auto-Accommodation Glasses includes the management of two cameras and two opto-electronic lenses, as this time is independent of the computing platform, our evaluation only includes the processing time of the images, i.e., the binocular pupil tracking algorithm (which determines the center of both pupils) plus the computation of the optical power to be applied by the lenses.

6 Results and discussion

6.1 Experimental results

In order to evaluate and compare the developed implementations, several experiments have been carried out. Figure 7 reports the overall performance of each computing platform, showing the time (in milliseconds) required to process a single frame (i.e., the reported performance metric is milliseconds/frame or frames per second) with a red line showing the minimum performance to achieve real-time processing (25 frames/sec). On the other hand, Fig. 8 shows the time breakdown for the main phases involved in processing an image. Those phases are the same than the ones described in the pseudocode of the pupil tracking algorithm shown in Fig. 1. Finally, Fig. 9 reports the energy efficiency of each platform showing how much energy (in Joules) it takes to process a single frame (i.e., the reported energy-efficiency metric is Joules/frame). A detailed discussion of the experimental results follows.

6.2 Performance

Figure 7 shows the processing time per frame in milliseconds for the different platforms and resolutions used. The desktop GPU platform (NVIDIA 980 GTX)

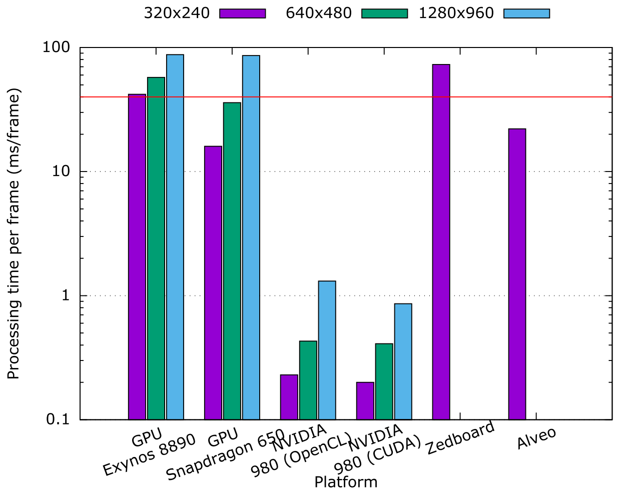


Fig. 7 Performance comparison of the different computing platforms. Three different image resolutions are compared: 320 × 240, 640 × 480 and 1280 × 960 pixels. The red line highlights the real-time limit (40 ms/frame, or 25 frames/sec)

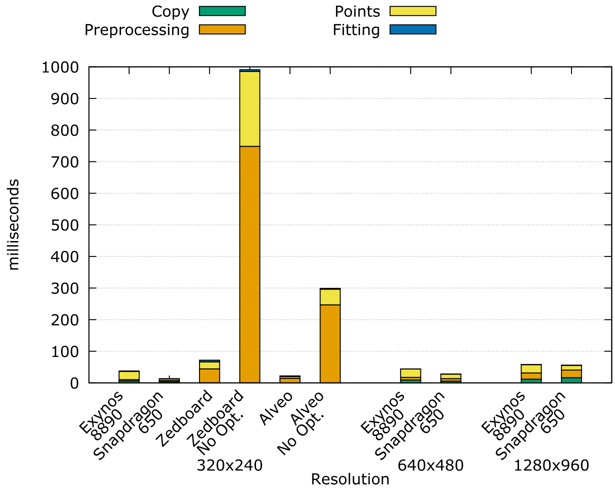


Fig. 8 Time spent by each platform in each processing step. *Copy*: copy the input data, the results and any intermediate value between the host and the device. *Preprocessing*: image processing operations before starting with the search of the pupils. *Points*: find the points in the border of the pupils. And *fitting*: fit the ellipses to the points and select the best ones

achieves a very high performance, as expected, beating both FPGAs and mobile GPUS by a big margin. Next are the mobile GPUs and the Alveo FPGA with similar results. Finally, the Zedboard FPGA obtains the slowest processing time of all the tested platforms.

More detailed results are shown in Fig. 8, this figure shows the time per frame broken down in four main phases of the processing: copying the data, image preprocessing operations, searching the points and fitting the ellipses (RANSAC). In this figure two implementations are plotted for the FPGAs, an optimized implementation and a not-optimized one. The slowest implementations are the not-optimized FPGAs versions, in those implementations most of the time was spent in the preprocessing and in the points search phases. After applying the optimizations it remains true, but their overall contribution is much smaller. For higher resolutions the preprocessing phase increases the execution time due to the more work required.

Finally, looking at the performance results achieved by the mobile GPUs, both the Samsung Exynos 8890 and the Qualcomm Snapdragon 650 are similar in performance, being the Exynos slightly slower than the Snapdragon. In particular, the Qualcomm Snapdragon 650 is capable of processing 640 × 480 images in less than 36 ms, i.e., fast enough to allow for real-time processing (or at least 40 ms/frame). The Samsung Exynos 8890, on the other hand, is only capable of processing 320 × 240 images under the real-time constraint. Still this lower resolution is enough for accurately detecting the pupils and properly driving the Dynamic Auto-Accommodation Glasses.

The final takeaway of Fig. 7 is that the processing time of these mobile GPUs slowly increases with the increase of the image size. This is partially due to the

Table 1 Maximum power (in watts) for each platform under three different resolutions

Resolutions	320 × 240	640 × 480	1280 × 960
Mali T-880	1.58W	2.06W	2.39W
Adreno 510	4.18W	4.78W	4.92W
NVIDIA 980 GTX	80.35W	92.12W	112.09W
Zedboard	3.48W		
Alveo	19.78W		

overhead of the OpenCL API calls being constant and also due to the better usage of the hardware resources with bigger images.

6.3 Energy efficiency

Regarding the energy efficiency of our OpenCL implementations for each of the evaluated computing platforms, Figure 9 reports the Joules required by each platform to process a single frame (J/frame). A common energy efficiency metric used in computer architecture is performance/watt (or throughput/watt). As the performance of our device, at the application level, is measured as processed frames per second (fps), it leads to:

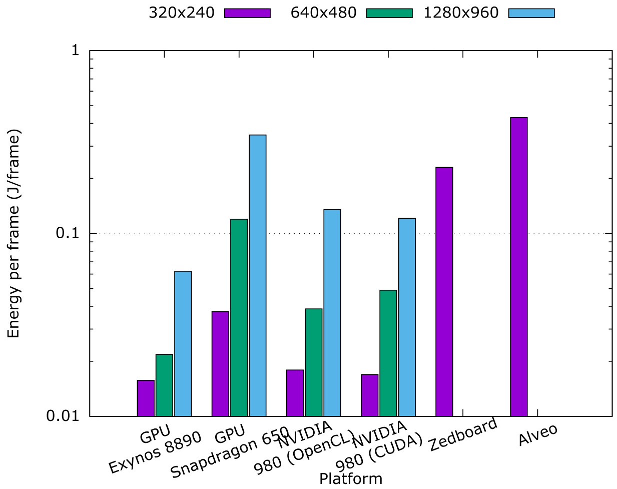


Fig. 9 Energy efficiency of the different computing platforms. Three different image resolutions are compared

$$\begin{aligned}\text{Energy Efficiency} =& \frac{\text{performance}}{\text{watts}}\\ &= \frac{\text{frames per second}}{\text{watts}}\\ &= \frac{\text{frames}}{\text{seconds} * \text{watts}} = \frac{\text{frames}}{\text{Joules}}\end{aligned} \tag{1}$$

As a more intuitive way to report the energy efficiency of the evaluated platforms, in Fig. 9 we report the inverse of Eq. (1), i.e., Joules/frame (or the energy consumed in processing each frame).

On the other hand, Table 1 shows the peak power drawn by each platform under different scenarios. This metric is useful to understand which platforms could be powered by a battery and which not.

It can be observed that both FPGA implementations show a low energy efficiency. In the case of the low-cost ZedBoard (which achieves 0.23 J/frame), it is because of the lack of logic cells that results in a not completely optimized implementation. The ZedBoard has a relatively low peak power consumption of 3.48 Watts. On the other hand, the Alveo u250 platform suffers from the high power dissipation of this high-end FPGA (we have measured an average of 19.2 Watts and a peak power of 19.8 Watts). This high power dissipation hurts its energy efficiency (0.43 J/frame) despite the reasonable achieved performance (22.1 ms/frame when considering 320 × 240 images).

The desktop GPU, as expected, has a very high power consumption (we have measured an average of 100 Watts depending on the image size) but this time it is compensated by a very high performance. As a result, its energy efficiency (0.018 J/frame) is similar to the efficiency obtained by the mobile GPUs.

Finally, mobile GPUs show a good efficiency which is similar for both of them, being the Exynos 8890 more energy efficient than the Snapdragon (0.016 J/frame vs. 0.037 J/frame). This was an expected result since mobile GPUs are designed to be highly energy efficient.

6.4 Discussion and future work

Overall, low-power consumption is a key requirement for our Dynamic Auto-Accommodation Glasses which are powered by a small battery. Table 2 shows a summary of the results presented in Sect. 6. That table shows that using a mobile

Table 2 Summary of the performance and energy efficiency of the different devices

Device	Time per frame (ms)	Energy per frame (J)
Exynos 8890	42	0.016
SnapDragon 650	16	0.037
Zedboard	73	0.23
Alveo u250	22.1	0.43

GPU for processing the images from the camera results in the best power-performance tradeoff, making the mobile GPU the best computing platform among the evaluated ones for our particular purpose.

The performance achieved by the low-cost FPGA platform (ZedBoard) might look poor in Figs. 7 and 8 (73 ms per frame) but it is not surprising due to the small number of programmable elements it includes (85K logic cells) preventing an efficient utilization of the FPGA optimizations such as loop unrolling, the use of local memory regions or array partitioning. Due to limited resources only the lowest resolution (320x240) could be processed on the FPGAs. Furthermore, even though some previous works have pointed that almost-direct ports of OpenCL kernels to a FPGA might result in poor performance [28], in Fig. 8 it can be seen a comparison of both FPGAs implementations (ZedBoard and Alveo) along with their non-optimized version. This test shows that the non-optimized version is much slower than the optimized one, which shows that even the basic optimizations applied are able to significantly boost performance (13.8x for the ZedBoard and 13.5x for the Alveo FPGA).

Regarding the desktop GPU, it is much more powerful than mobile GPUs since it has much higher memory bandwidth, computing power and resources (cores, internal memories, caches, clock frequency). Therefore, it was expected that it would get the best performance of all the platforms. Its energy-efficiency is also good because the high performance compensate for its high power dissipation. However, a desktop GPU is not feasible as a wearable solution due to its size and high power dissipation. Still, this platform has been added as a reference point to enrich the comparison.

From these findings, it is clear that future work using mobile SoCs is the best approach for a wearable solution. However, it is true that a lower-level FPGA implementation would probably yield better results in both metrics, performance and energy-efficiency. Of course, developing a lower-level FPGA implementation takes longer than writing the same code for a mobile SoC. Therefore, it is critical to evaluate if the potential improvement is worth the development time.

7 Conclusions

A novel Dynamic Auto-Accommodation device controlled by either a mobile GPU or a low-cost FPGA has been proposed. The device is capable of tracking in real time the pupil of the subject, calculating its gaze distance in order to apply the right amount of diopters. A novel OpenCL implementation of a pupil tracking algorithm has been implemented and optimized for both GPUs and FPGAs. A performance and energy-efficiency comparison of several hardware platforms have been performed to find the best one for the presbyopia correction application. Two mobile GPUs, two FPGAs and one desktop GPU have been evaluated, measuring their performance and energy efficiency, and showing that the most energy-efficient platform while meeting the real-time constrain is a mobile GPU (Samsung Exynos 8890) capable of processing frames at 0.016 Joules/frame.

Acknowledgements This work was supported by the Spanish MCIU and AEI under grant RTI2018-098156-B-C53, grant PID2019-105684RB-I00 and Fundación Séneca-Agencia de Ciencia y Tecnología de la Región de Murcia (19897/GERM/15).

Funding Open Access funding provided thanks to the CRUE-CSIC agreement with Springer Nature.

References

1. Afonso S, Acosta A, Almeida F (2019) High-performance code optimizations for mobile devices. J Supercomput 75(3):1382–1395
2. Al-Rahayfeh A, Faezipour M (2013) Eye tracking and head movement detection: a state-of-art survey. IEEE J Trans Eng Health Med 1:2100212
3. Arm Ltd (2021) Arm mali midgard opencl developer guide 3.14. https://developer.arm.com/documentation/100614/0314/OpenCL-concepts/Mali-GPU-OpenCL-memory-model. Accessed 17 Nov 2021
4. Arm Ltd (2021) Mali-t880. https://developer.arm.com/ip-products/graphics-and-multimedia/mali-gpus/mali-t860-and-mali-t880-gpus. Accessed 17 Nov 2021
5. Artal P, Mompeán J, Aragón JL. Optoelectronic binocular instrument for the correction of presbyopia and method for the binocular correction of presbyopia. US Patent No. 10,898,073. 26 Jan 2021
6. Blum M, Büeler M, Grätzel C et al (2011) Compact optical design solutions using focus tunable lenses. Optical Design and Engineering. International Society for Optics and Photonics, Bellingham
7. Duane A (1922) Studies in monocular and binocular accommodation with their clinical applications. Am J Ophthalmol 5(11):865–877
8. Felipe Herranz (2016) Usbserial. https://github.com/felHR85/UsbSerial. Accessed 16 Nov 2016
9. Fischler MA, Bolles RC (1981) Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. Commun ACM 24(6):381–395
10. Fisher R (1971) The elastic constants of the human lens. J Physiol 212(1):147–180
11. Glasser A, Campbell MC (1998) Presbyopia and the optical changes in the human crystalline lens with age. Vision Res 38(2):209–229
12. Heys KR, Cram SL, Truscott RJ (2004) Massive increase in the stiffness of the human lens nucleus with age: the basis for presbyopia? Faculty of Engineering and Information Sciences - Papers: Part A. 2667
13. Jarosz J, Lavigne Q, Molliex N et al (2018) Experimental optical analysis of an original presbyopia-correcting variable focus lens. Invest Ophthalmol Vis Sci 59(9):255
14. Kim S, Oh S, Yi Y (2021) Minimizing GPU kernel launch overhead in deep learning inference on mobile GPUS. Association for Computing Machinery, New York, NY, USA, pp 57–63
15. Kono F, Nakasato N, Hayashi K et al (2018) Evaluations of OpenCL-written tsunami simulation on FPGA and comparison with GPU implementation. J Supercomput 74(6):2747–2775
16. Li D, Winfield D, Parkhurst DJ (2005) Starburst: a hybrid algorithm for video-based eye tracking combining feature-based and model-based approaches. In: 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)-Workshops, IEEE, pp 79
17. Mastrangelo AS, Karkhanis M, Likhite R et al (2018) A low-profile digital eye-tracking oculometer for smart eyeglasses. In: 2018 11th International conference on human system interaction (HSI), IEEE, pp 506–512

18. Mompeán J, Manzanera S, Aragón JL et al (2016) Presbyopia correction with optoelectronic lenses driven by pupil size. Invest Ophthalmol Vis Sci 57(12)
19. Mompeán J, Aragón JL, Prieto PM et al (2018) Design of an accurate and high-speed binocular pupil tracking system based on GPGPUS. J Supercomput 74(5):1836–1862
20. Mompeán J, Aragón JL, Artal P (2020) Portable device for presbyopia correction with optoelectronic lenses driven by pupil response. Sci Rep 10(1):1–9
21. Munshi A (2009) The opencl specification. In: 2009 IEEE Hot chips 21 symposium (HCS), IEEE, pp 1–314
22. Padmanaban N, Konrad R, Wetzstein G (2018) Autofocals: gaze-contingent eyeglasses for presbyopes. ACM SIGGRAPH 2018 Emerging Technologies. ACM, New York, p 3
23. Qualcomm Technologies, Inc (2018) Trepn power profiler. https://developer.qualcomm.com/software/trepn-power-profiler. Accessed 10 Jun 2018
24. Saki (2018) Uvccamera. https://github.com/saki4510t/UVCCamera. Accessed 5 Mar 2021
25. Sterner B, Gellerstedt M, Sjöström A (2004) The amplitude of accommodation in 6–10-year-old children-not as good as expected! Ophthalmic Physiol Opt 24(3):246–251
26. Yiu J (2017) Arm cortex-m for beginners. An overview of the Arm Cortex-M processor family and comparison. [White paper]. ARM
27. Zhao D (2015) Fast filter bank convolution for three-dimensional wavelet transform by shared memory on mobile GPU computing. J Supercomput 71(9):3440–3455
28. Zohouri HR, Maruyama N, Smith A et al (2016) Evaluating and optimizing opencl kernels for high performance computing with fpgas. In: SC'16: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, IEEE, pp 409–420