

Performance of a simple remote video-based eye tracker with GPU acceleration

Jean-Pierre du Plessis
University of the Free State
South Africa

Pieter Blignaut
University of the Free State
South Africa

Eye tracking is a well-established tool that is often utilised in research. There are currently many different types of eye trackers available, but they are either expensive, or provide a relatively low sampling frequency. The focus of this paper was to validate the performance and data quality of a simple remote video-based eye tracker that is capable of attaining higher framerates than is normally possible with low-cost eye trackers. It utilises the Graphical Processing Unit (GPU) in an attempt to parallelise aspects of the process to localize feature points in eye images. Moreover, the proposed implementation allows for the system to be used on a variety of different GPUs. The developed solution is capable of sampling at frequencies of 200 Hz and higher, while allowing for head movements within an area of $10 \times 6 \times 10$ cm and an average accuracy of one degree of visual angle.

Keywords: GPU, accuracy, precision, sampling frequency

Introduction

Eye trackers are useful for a number of research applications, such as research in reading and dyslexia, language processing and mental health disorders (Duchowski, 2002; SensoMotoric Instruments, n.d.). With current commercial high speed trackers costing more than 30 000 euros, there is certainly reason to conduct research into more affordable alternatives.

There are several existing non-commercial implementations that have demonstrated the feasibility of performing eye tracking at sampling frequencies of over 200 Hz. The eye tracker of Hennessey, Noureddin and Lawrence (2008) are capable of sampling at 407 Hz. Mulligan (2012) presents a GPU assisted solution that utilises NVIDIA's CUDA programming language and is capable of achieving theoretical sampling rates of up to 250 Hz. Mompean, Aragon, Prieto and Artal (2015) used CUDA and OpenMP and proved that pupil tracking can be efficiently performed at high speeds with high-resolution images (up to 530 Hz


with images of 1280×1024 pixels) using a state-of-the-art GP-GPU.

In this paper, we present a GPU assisted eye tracking solution capable of attaining sampling frequencies in excess of 200 Hz on a mid-range laptop. The system is capable of attaining an average accuracy of one degree of visual angle or better along with a precision of approximately 0.3 degrees.

The following section will focus on the GPU and its potential for use in eye tracking. Thereafter, the use of the GPU in the solution is discussed, along with details regarding the process followed to identify feature points and perform gaze estimation. Finally, details regarding the experiment performed to evaluate the system are provided along with a discussion on the results.

The GPU and Eye Tracking

Traditionally, in an attempt to limit the overall cost, eye tracking systems utilised commercial off-the-shelf components. To this end, the fact that modern day computers commonly come equipped with multi-core CPUs and integrated GPUs presents researchers with the prospect of additional computing power through the use of parallelism.

Received April 6, 2016; Published August 15, 2016.
Citation: Du Plessis, J-P & Blignaut, P.J. (2016). Performance of a simple remote video-based eye tracker with GPU acceleration. *Journal of Eye Movement Research*, 9(4):6, 1-11.
Digital Object Identifier: 10.16910/jemr.9.4.6
ISSN: 1995-8692
This article is licensed under a [Creative Commons Attribution 4.0 International license](https://creativecommons.org/licenses/by/4.0/). 

Parallel potential in eye tracking

The idea of writing programs that perform tasks in parallel has become more popular in recent years due to the ever increasing amount of data that must be processed. Parallelisation can be divided into two categories, namely task parallelism and data parallelism (Pacheco, 2011), where the former refers to executing multiple tasks concurrently and the latter to dividing the data into smaller portions and executing the same series of tasks on each portion in parallel. Parallelism works best when the tasks can be executed independently (Sottile, Mattson & Rasmussen, 2010) as this limits the amount of communication that must occur between tasks.

When one considers the eye tracking process, there are several potential areas where parallelism can be applied. The most obvious place to start would be in the processing of the eye video – arguably the most time consuming task in eye tracking. Typically, the processing of the eye video involves numerous iterations over millions of pixels for each image, a task that is trivial to implement in parallel if one considers image processing functions that do not rely on the results of processing on neighbouring pixels.

A popular example of image processing within eye tracking is simple thresholding, where the colour intensity of a pixel is compared against a certain threshold. A further example is an algorithm, such as Canny edge detection (Canny, 1986), that is used to find the edges of pupils. The Gaussian blur that often precedes an edge detection operation can also be executed in parallel (vertical and horizontal blur) before combining the results. Each of these functions has the additional advantage of being performed on a per-pixel basis, and can therefore be data parallelised.

When one considers that an eye video consists of many thousands (even millions) of pixels, it stands to reason that a dedicated device, such as the GPU, could be highly advantageous in parallelising the image processing tasks in eye tracking. This ultimately points to the SIMD architecture of the modern GPU, which allows operations to be performed on many pixels simultaneously.

Advances in GPU design in the last decade or so have increasingly opened up the power of the GPU to researchers wishing to utilise the massively parallel architecture of

the GPU to accelerate certain tasks. The resulting programs are referred to as GPGPU (General Purpose Calculations on the Graphics Processing Unit). An early example of such an implementation is that of matrix multiplication involving large vectors (Thompson, Hahn & Oskin, 2002).

As a result, there are several application programming interfaces (APIs) that can be used to perform computations on the GPU. One example of such an API is NVIDIA's Compute Unified Device Architecture (CUDA) (NVIDIA, 2016). CUDA was developed to simplify the process of writing code for general computations on the GPU (Castaño-Díez, Moser, Schoenegger, Pruggnaller & Frangakis, 2008) and has already been used in eye tracking (Mulligan, 2012; Duchowski, Price, Meyer & Orero, 2012; Mompean et al., 2015).

Microsoft's DirectX provides access to programmable shaders¹ through the High Level Shader Language (HLSL). HLSL allows the creation of C like programmable shaders for the Direct3D pipeline. It was first introduced in DirectX 9 (Peeper & Mitchell, 2004) and allows developers to take exact control of what happens to graphics data on a per-vertex and per-pixel basis. As of Microsoft DirectX 11, researchers have access to compute shaders through which high speed general purpose computing can be performed (Microsoft, 2016a).

Existing work

To date, the GPU has only been utilised in single instances in eye tracking. As mentioned previously, Mulligan (2012) proposed a solution making use of CUDA that is capable of achieving 250 Hz on a 640×480 image, with average accuracies close to 0.5 degrees of visual angle. In another example, Duchowski et al. (2012) made use of the GPU to build real time heat maps.

Mompean et al. (2015) used CUDA and OpenMP to implement and compare three different tracking algorithms, *inter alia* the well known Starburst algorithm (Li et al., 2005), accelerated by GPUs. They found that pupil tracking can be efficiently performed at framerates up to 530 Hz using a state-of-the-art GP-GPU.

¹ A small C type program that executes on the GPU

Pitfalls in GPU accelerated eye tracking

It should be noted that the enormous processing power of the GPU comes with a specific drawback, namely that data must be transferred back and forth between the GPU and system memory. As illustrated by Mulligan (2012) and depending on the size of the data to be copied, as well as the physical performance and system specifications of the host machine, the transfer can take anything from a millisecond or longer. This delay can be considered time consuming in the light of the real-time nature of eye tracking.

A delay of only a couple of milliseconds can result in a drastically lower sampling rate. This delay becomes increasingly problematic as the required sampling frequency increases, as even a single millisecond delay can lead to a significantly lower sampling frequency. Thus, to prove beneficial to the eye tracking process, the GPU has to perform work at a significantly faster rate than CPU equivalents to justify the added cost of data transfer to and from the GPU.

An additional issue with GPU programming is that the use of the GPU may limit the kind of image processing that can be performed. Not all algorithms are parallelisable and the GPU's memory architecture is not as flexible as that of the CPU, making programming for GPUs challenging (Castaño-Díez et al, 2008).

These two issues are perhaps the reason why currently there do not exist many real-time GPU accelerated eye tracking solutions.

Motivation

While the studies of Mulligan (2012) and Mompean et al. (2015) were, in our eyes, ground-breaking, we believe that the current study will contribute to the extent that it confirms that GPU-enabled eye tracking is also feasible with off-the-shelf components and mid-range computers.

Furthermore, Mompean et al. (2015) evaluated the tracking algorithms in terms of the accuracy of pupil detection but did not go as far as mapping the detected eye features to gaze coordinates and evaluate the gaze data quality in terms of accuracy and precision of reported gaze coordinates. Although this is a next step in the eye tracking process and the data quality can be adversely affected by any of the preceding phases (i.e. image quality and feature detection), we believe it is important since it gives credibility to the process and can easily be interpreted in terms of known standards.

Implementation

Video-based eye tracking

In a video-based eye tracker, the relative position of the pupil to the corneal reflection(s) (the so-called pupil-glint vector) changes as the eyes move. Using an appropriate model, the x and y dimensions of the pupil-glint vector can be mapped to screen coordinates (X,Y). As the magnitude of the pupil-glint vector changes with backward and forward head movements, it has to be normalised in terms the distance between the glints (if there are more than one IR source) or between the pupils.

The implementation discussed below makes use of two infrared light sources, positioned equidistantly on either side of a high speed camera to generate corneal reflections.

Image processing

The solution presented in this paper makes use of Microsoft DirectX 9 and shaders written in HLSL. By using DirectX 9, we ensure that the implementation is compatible on a wide range of graphics cards, rather than relying on specific GPUs such as those manufactured by NVIDIA and AMD. This brings down the overall cost of the system, as potential users can buy a high-speed camera off-the-shelf and use whatever PC they have at their disposal, provided the display adapter meets the minimum requirements for DirectX 9 and supports Shader Model 2. The host application was developed using C# with .Net 4.5.

The proposed eye tracking solution consists of a three step process to analyse eye videos and locate the feature points. Firstly, the raw frames from the camera are wrapped into the structure required by DirectX for rendering. Next, the camera image is transferred to the GPU's memory (VRAM) for manipulation using two shaders written in HLSL. The goal of these shaders is to simplify the job of extracting candidate feature points. The results of both of these shaders are stored in off-screen render targets. The first render target was used as input for the second shader. The final render target contained the final processed image – the combined result of the first and second shader. This image is then copied back to system memory and processed by the CPU to compute the locations of the feature points. Figure 1 below gives an outline of the program architecture.

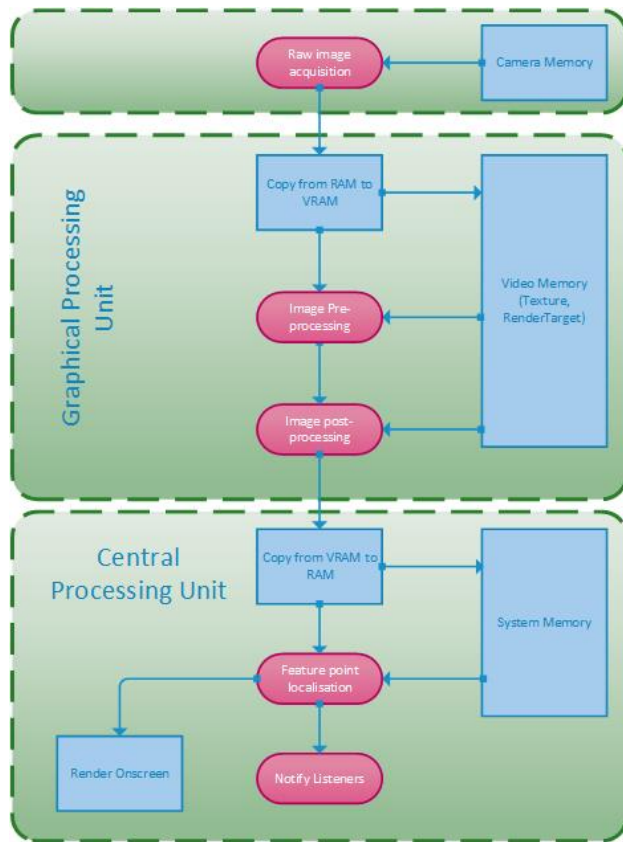


Figure 1: Overview of system architecture

The use of two separate shaders was necessary due to the low instruction count imposed by Shader Model 2.0 (256 instructions) (Microsoft, 2016b), as well as the dependencies between certain of the image processing functions.

As mentioned earlier, there are several commonly used image processing functions utilised in eye tracking that are embarrassingly² parallel and are therefore simple to implement on the GPU. The use of simple functions was motivated by the need to evaluate the worst case for the GPU in terms of delay versus processing gain. In other words, should the GPU prove beneficial even when using simple functions, it can be argued that its worth will increase as the complexity of the image processing functions increase.

Shader implementations

The first of the image processing shaders (pre-process) performs a number of threshold functions to isolate the corneal reflections (a.k.a. glints) and pupils from the rest

of the image in order to simplify the image for later processing. All thresholds were set by hand and adjusted per-participant as needed. As the ultimate goal of the shaders is to simplify the job of locating the feature points, the pupil and glint candidate pixels are represented with two different colours – green and red respectively. To isolate the glints, a simple intensity gradient is constructed using the brightness of the surrounding pixels. The resulting gradient, along with a suitable threshold is then used to select corneal reflection candidates.

A second threshold test is used to isolate pupil candidates. If the previous positions of the corneal reflections are known, the distance to the closest corneal reflection is used to eliminate false positives that result from eye lashes or eyebrows. The resulting image is then sent to a second shader for edge detection.

The second shader (post-process) performs Canny (Canny, 1986) edge detection (instead of the usual ellipse fitting) on the image from the previous shader to further decrease the number of pixels that must be processed by the CPU. The centroids of the pupils and glints were calculated as the arithmetic mean of the x and y coordinates of the pixels in the pupil/glint.

An intersection between the glint and pupil causes the pupil to lose its circular shape which will in turn offset the computed pupil centre. To counter this, an iterative process was followed to include glint pixels inside the previously known pupil radius until the pupil centre stabilised within pre-set parameters.

The entire GPU related process is summarised in Figure 2.

CPU processing

After the image has been processed and candidate pixels separated into the green and red colour channels, the CPU calculates the centres of the pupils and glints. The pupil and glint candidate points are aggregated, and an image processing library (<http://www.aforgenet.com/>) is used to calculate the centres of each of these collections.

In this particular implementation of the HLSL tracker, no ellipse fitting was performed. However, it is certainly possible, if necessary, to add this step at a later stage – given that all the pixels belonging to the identified features are already grouped into separate collections.

² Parallelism that can be implemented without any dependencies.

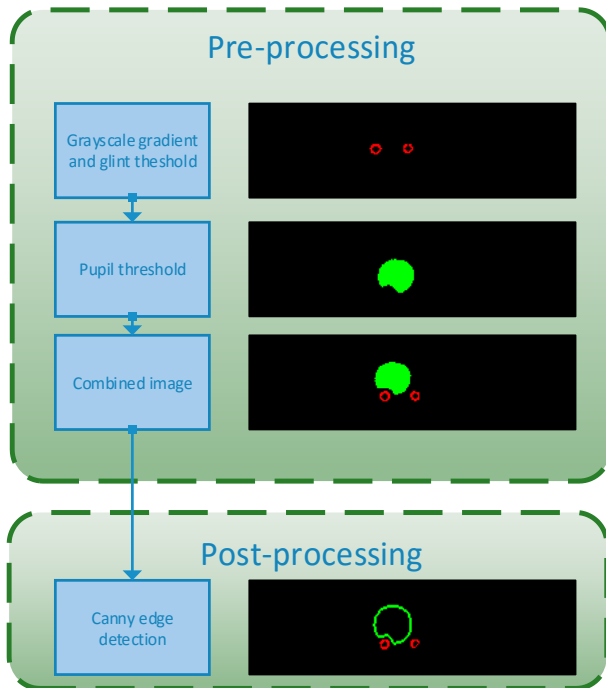


Figure 2: Overview of the pre- and post-process shaders

Evaluation of the HLSL eye tracker

Participants

To evaluate the performance of the eye tracking solution, an experiment involving 30 participants was conducted in a laboratory. Participants were drawn from staff and students on campus. To ensure the best possible results, none of the participants wore spectacles.

Physical setup and equipment

Using adjustable lighting, the brightness of the room was set at 300 lux. The stimulus consisted of a 20" monitor with a resolution of 1360×768 and was positioned 70 cm from the participant, resulting in a horizontal gaze angle of 11.7° from side to side. The eye tracking software was installed on a HP Pavilion G7 laptop running Windows 8 (x64). The laptop is equipped with an Intel Core i5-3230M processor running at 2.60 GHz with 4 GB of DDR3 1333 Mhz RAM. The laptop is further equipped with switchable graphics containing an Intel HD4000 and a Radeon HD 7670M graphic card. For the purpose of this study, the Intel card was used.

Eye images were captured using a single USB 3.0 CMOS camera from IDS (Model UI-3360CP-NIR-GL)

(<https://en.ids-imaging.com/store/ui-3360cp.html>). The camera has a pixel size of 5.5 μm and a sensor size of 2/3". At its native resolution of 2048×1088 (2M) the camera is capable of 70 fps, but if only a portion of the sensor area is used, the framerate can be increased substantially. Although not tested, we believe that the results obtained in this study could be achieved also with a cheaper camera as long as it supports a framerate of at least 300 Hz for a 90K pixel region of interest at the available levels of illumination. Such cameras, for example the IDS UI-1550LE-C, can be acquired for more or less 400-500 USD.

The camera allows for the adjustment of the size of the video, as well as the frame rate, gain and gamma. This allowed us to sample at a variety of different speeds in addition to making the necessary adjustments to compensate for changing light conditions resulting from different exposure times. The camera was fitted with a 16 mm lens with a filter to allow only infrared light to pass.

Two infrared light sources, positioned equidistantly on either side of the camera, were used to generate corneal reflections. The eye tracker was mounted on a table with an adjustable surface that can be accurately positioned along three axes (*cf.* Figure 3).

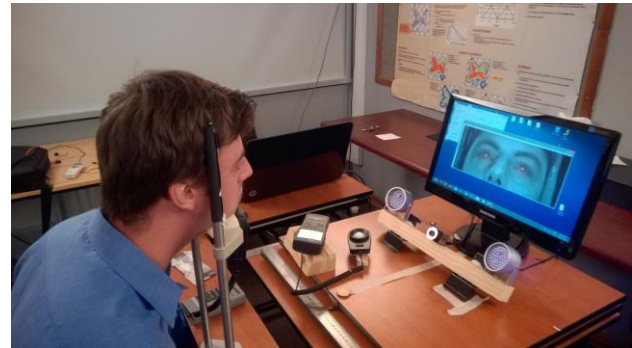


Figure 3: Laboratory setup with the participant on a fixed chin rest and the eye tracker and monitor on an XYZ adjustable table

Experimental procedure for human participants

The experiment required participants to look at a grid of forty dots evenly spaced on the monitor. The dots were displayed in random order for a total of two seconds. A time frame of 250 ms during which the eye data was adjudged to be most stable, was selected for analysis.

The experiment consisted of two phases. In the first phase, the effect of the sampling frequency on the precision and accuracy was investigated while keeping the head

position in the centre of the head box. In the second phase, the effect of head position on the precision and accuracy was investigated at a sampling frequency of 100 Hz while the position of the eye tracker was changed in relation to the participant to simulate head movements (*cf* Figure 3).

During both phases of the experiment, participants were seated 70 cm from the monitor and 65 cm from the eye tracker. Participants' head positions were fixed using a chin rest (*cf* Figure 3). The entire experimental setup involving human participants is summarised in Table 1.

Table 1. Summary of the various combinations of settings that were used during the experiment

Sampling frequency (Hz)	Region of interest (Pixels)	X Pos (cm)	Y Pos (cm)	Z Pos (cm)
Phase 1				
50	600×150	0	0	65
100	600×150	0	0	65
150	600×150	0	0	65
200	600×150	0	0	65
250	600×150	0	0	65
300	600×150	0	0	65
Phase 2				
100	1200×400	+5	0	65
100	1200×400	-5	0	65
100	1200×400	0	+3	65
100	1200×400	0	-3	65
100	1200×400	0	0	70
100	1200×400	0	0	60

Experimental procedure for artificial eyes

In addition to human participants, the theoretical performance of the system was examined using artificial eyes. In theory, artificial eyes should yield precision values of 0°, as they remain completely still. However, eye tracking systems contain noise and the purpose of the test with artificial eyes was to determine the precision that is obtainable if the continuous head movements and eye tremors that are present in human participants were completely eliminated, thereby determining the amount of noise produced by the eye tracking system.

The artificial eyes were mounted on a cardboard and clamped in a fixed position to the table – aiming more or less at the centre of the display. As recommended by Holmqvist et al (2011), the calibration data of a similarly positioned human participant was used to map the feature points to gaze coordinates.

Since the eyes could not be pointed to specific targets on the display, accuracy could not be measured. For the effect of frequency on precision, an ROI of 600×150 was

used and the same procedure as for human participants was followed. This means that the gaze target was displayed at 40 distinct positions but gaze data was captured in the centre of the display only, resulting in 240 samples (40 repetitions × 6 frequencies).

Analysis

All data recorded during the experiment was collected in a database and analysed post-hoc. The data included the location coordinates of all detected feature points in the eye video as well as time stamps for each set of coordinates.

Fixation data from fourteen of the forty dots was used as calibration data to construct the gaze estimation polynomials for every participant (*cf* Figure 4). All dots were used to validate the accuracy and precision of the system. The time stamps were used to verify the actual sampling rate of the system.

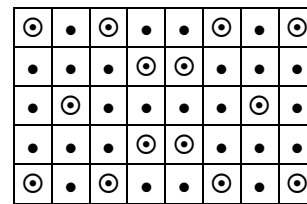


Figure 4: 8×5 grid of dots. All dots were displayed as • to participants. The ○ around the dots only serve to indicate the dots that were used for the regression.

Since the system geometry was similar to that of a system used for an earlier study (Blignaut, 2013), the following two polynomials were used to map feature point coordinates in the eye video to gaze coordinates on the display (x and y refer to the normalised x and y components of the pupil-glint vector of a specific eye at a specific point in time. PoR_x and PoR_y refer to the X and Y coordinates of the point of regard for the specific eye on the two dimensional plane of the screen):

$$PoG_x = a_0 + a_1x + a_2x^2 + a_3x^3 + a_4y + a_5xy + a_6x^2y + a_7x^3y$$

$$PoG_y = b_0 + b_1x + b_2x^2 + b_3y + b_4y^2 + b_5xy + b_6x^2y$$

For samples within a fixation (data captured for a specific gaze target), we assumed that the spread of data around the centroid would be normal and outlier samples were identified as those lying beyond 3σ from the centroid. In other words, maximally 0.54% of the data points were removed and the centroid recalculated.

Precision was then calculated as the pooled variance of mapped sample data within a fixation in the x and y dimensions, as it has been shown previously that the de facto standard sample-to-sample Root Mean Square (RMS) is affected by the sampling rate of the eye tracker (Blignaut & Beelders, 2012):

$$SD(P) = \sqrt{(\sigma_x^2 + \sigma_y^2)/2}$$

$$\text{where } \sigma_x^2 = \frac{1}{N} \sum_{i=1}^N (x_i - \bar{x})^2$$

$$\text{and } \sigma_y^2 = \frac{1}{N} \sum_{i=1}^N (y_i - \bar{y})^2$$

For each gaze target, the centroid of samples was determined. Data sets for which no or not adequate gaze data were recorded (due to blinks or participants being distracted) and data sets further than 3° from the gaze target were regarded as outliers and removed. The accuracy was calculated as the average (over all dots and participants) of the differences between the known locations of the dots and the centroids of the clusters of sample data.

Analyses of variance (ANOVA) were done to determine the significance ($\alpha = .05$) of all results. Tukey's HSD (honestly significant difference) for unequal number of observations was used to determine the significance ($\alpha = .05$) of differences in accuracy and precision between individual combinations of sampling frequencies and head positions.

Results

Precision and accuracy per participant

Figures 5 and 6 show the overage accuracy and precision of gaze data samples in the 250 ms time frame over all target points per participant at a sampling frequency of 100 Hz while the head was positioned in the centre of the head box. The average accuracy and precision of samples over all target points of all participants under these conditions were 1.03° (SD = 0.66°) and 0.33° (SD = 0.15°) respectively. (Note that this is not the same as averaging the per-participant averages.) The average overall accuracy and precision for the other framerates are presented in Table 2 in the next section.

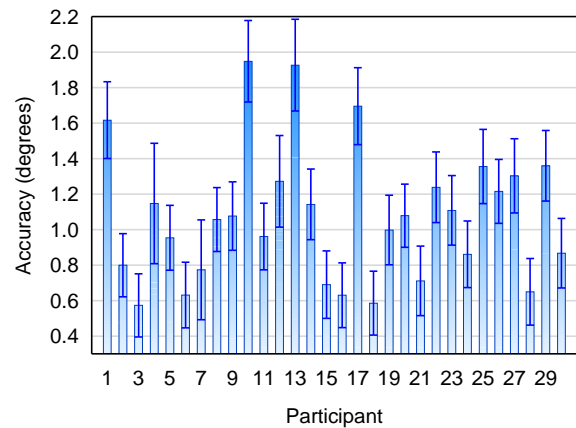


Figure 5: Average accuracy per participant at 100 Hz in the centre of the head box. Vertical bars denote 95% confidence intervals.

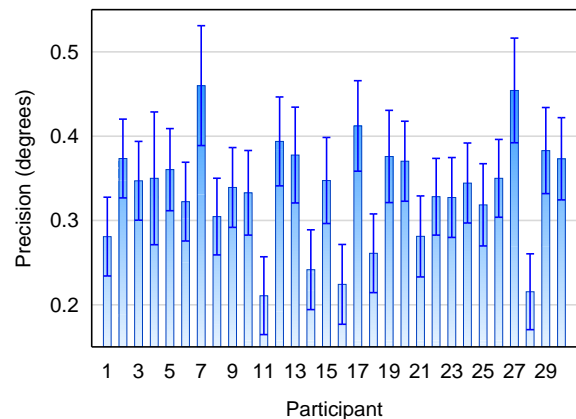


Figure 6: Average precision per participant at 100 Hz in the centre of the head box. Vertical bars denote 95% confidence intervals.

Precision and accuracy against sampling frequency

Table 2 shows the accuracy and precision values that were obtained for a range of sampling frequencies. The data is visualised in Figures 7 and 8. Although the pooled variance was used for precision in this paper, the RMS values are shown for 200 Hz and 250 Hz for purpose of comparison with Mulligan (2012).

Precision was affected significantly ($F(5,5600) = 37.923, p = .000$) by sampling frequency (Figure 7). However, Tukey's unequal honestly significant difference between pairs of frequencies indicated that the effect was not significant for 150 Hz to 200 Hz or 250 Hz to 300 Hz. Likewise, accuracy was significantly affected (Figure 8) by sampling frequency ($F(5,5452) = 20.847, p = .000$). However, the post-hoc test revealed that it was only significantly affected between 50 Hz and 100 Hz.

Table 2. Accuracy and precision for a range of sampling frequencies

Frame-rate (Hz)	Accuracy		Pooled variance		RMS	
	Mean	SD	Mean	SD	Mean	SD
50	0.81	0.46	0.29	0.11		
100	1.03	0.67	0.33	0.15		
150	0.97	0.60	0.35	0.17		
200	1.03	0.64	0.36	0.18	0.38	0.13
250	1.04	0.62	0.38	0.19	0.40	0.12
300	1.07	0.67	0.36	0.16		

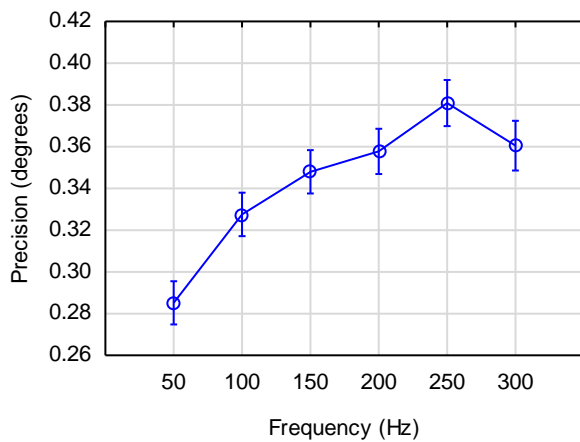


Figure 7: Average precision against sampling frequency. Vertical bars denote 95% confidence intervals.

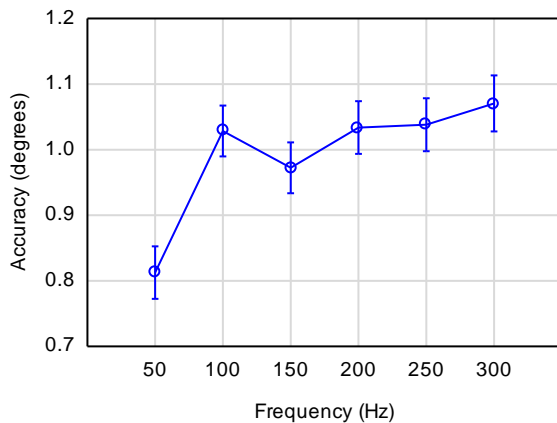


Figure 8: Average accuracy against sampling frequency. Vertical bars denote 95% confidence intervals.

The results show that both accuracy and precision appear to gradually deteriorate as the sampling frequency increases. The change in precision may be attributed to the adjustments made to the gain and gamma of the camera,

which resulted in a slight graininess in the eye video (Figure 9). Although the need for electronic gain could have been reduced by increasing the level of illumination, we decided to keep the illumination constant at a level which was tested to be within accepted safety limits.

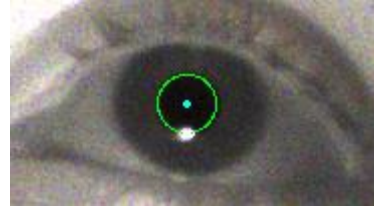


Figure 9. An example of the graininess resulting from adjustments to gain and gamma.

Precision and accuracy vs head position

The mean precision (pooled variance) and accuracy of the system for various head positions are shown in Tables 3 and 4.

Tukey's post-hoc revealed that the effect of head position on precision was not significant for any of the head movements. However, for accuracy (Table 4) the effect was significant on X - 5 and Z - 5. It is expected that horizontal head movements will affect the accuracy of the eye tracker as a simple regression-based gaze estimation model was used.

Table 3. Precision as affected by head position

Axis	Precision (degrees)				Sign.
	Mean	Min	Max	SD	
Centre	0.33	0.05	0.78	0.1531	
X + 5	0.32	0.06	0.74	0.1403	
X - 5	0.31	0.09	0.69	0.1306	
Y + 3	0.33	0.08	0.77	0.1481	
Y - 3	0.33	0.08	0.81	0.1595	
Z + 5	0.34	<0.01	0.8	0.1551	
Z - 5	0.31	0.06	0.83	0.1743	

Table 4. Accuracy as affected by head position

Axis	Accuracy (degrees)				Sign.
	Mean	Min	Max	SD	
Centre	1.03	0.18	3.03	0.6692	
X + 5	1.08	0.15	3.09	0.6908	
X - 5	1.15	0.13	3.39	0.7479	✓
Y + 3	1.12	0.17	3.3	0.7291	
Y - 3	1.04	0.2	3.02	0.6633	
Z + 5	0.94	0.09	2.51	0.5262	
Z - 5	1.25	0.4	3.94	0.9007	✓

Tables 5 and 6 summarise the results of the effect of head movement on precision and accuracy. The key point here is that precision changes whenever the size of the feature points change in response to forward and backward head movements. However, vertical and horizontal head movements did not significantly affect precision.

Table 5. Significance of the effect of head position on precision

Axis	F	p	Significant ($\alpha = .05$)
X	F(2,2775)=2.014	.134	
Y	F(2,2895)=.630	.532	
Z	F(2,2754)=8.519	.000	✓

Table 6. Significance of the effect of head position on accuracy

Axis	F	p	Significant ($\alpha = .05$)
X	F(2,2699)=5.628	.004	✓
Y	F(2,2787)=4.857	.008	
Z	F(2,2678)=43.460	.000	✓

Artificial eyes

As with the human participants, the effect of the sampling frequency on precision was significant ($F(5,234) = 235.3, p = .000$) (Figure 10).

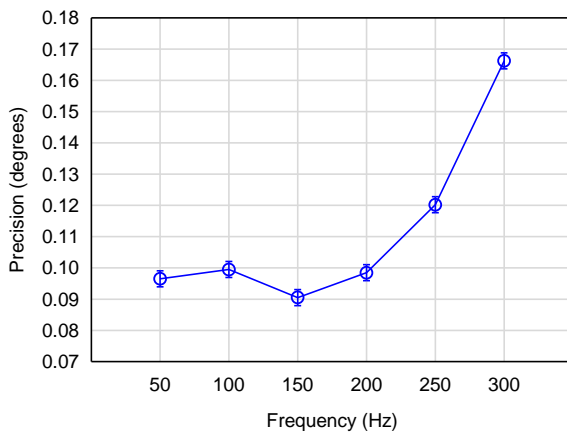


Figure 10. Precision against sampling frequency for artificial eyes

The precision obtained with artificial eyes again shows a decline in precision with increasing frequency, suggesting that the weaker precision is a result of noise in the system. The graininess of the eye video can again be the culprit. Tukey’s post-hoc test did show that this decrease in precision was significant, suggesting that the system’s optimal operating frequency is at lower speed – possibly around the 200 to 250 Hz range.

GPU performance

As mentioned earlier, the use of the GPU does introduce additional overheads into the system in the form of transfer to and from the GPU. Figure 11 shows the inverse of the maximum obtainable sampling frequency (or the shortest possible time interval between successive gaze data samples) against the size of the area on the camera sensor that are analysed. The coefficient of determination, R^2 , of a linear fit of the data was 0.996.

These results imply a trade-off between the amount of head movement that can be tolerated and the maximum framerate that can be achieved. Mostly, participants move their heads in a horizontal direction and therefore the eye video sizes can be optimised to allow for more horizontal movement. With a window of 600 pixels wide and only 200 pixels high (120,000 pixels), a framerate of 357 Hz (0.0028 s between samples) can be achieved, whereas a standard 4:3 window of 640×480 (307,200 pixels) would allow only 167 Hz (0.006 s).

Therefore, it can be concluded that with the correct selection of eye video resolution, the GPU can be used in the eye tracking process and a higher sampling frequency can still be obtained despite the overhead.

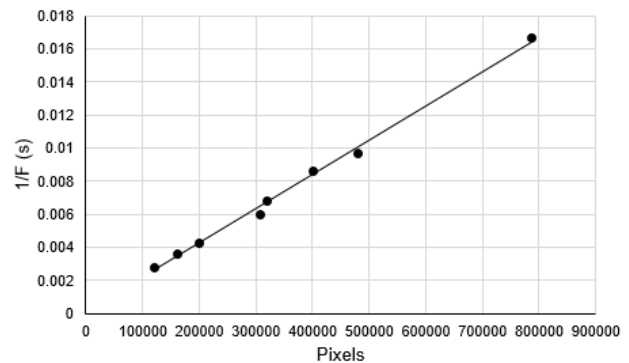


Figure 11. Inverse of maximum obtainable sampling frequency against image size

Summary

The eye tracker presented in this paper was developed to validate the data quality of a simple remote video-based eye tracker that is capable of framerates to 300 Hz. It utilises the Graphical Processing Unit (GPU) in an attempt to parallelise aspects of the process to localize feature points in eye images to attain higher sampling frequencies. The system was evaluated at various sampling frequencies and

with simulated head positions to gauge the effect of these variables on the obtainable precision and accuracy. Artificial eyes were also included to evaluate the theoretical performance of the system at higher frequencies, and the effect of the GPU implementation on the maximum obtainable sampling rate.

The results indicated that it is possible to perform eye tracking at a sampling rate of around 200 Hz with a tolerance towards head movement within an area of $10 \times 6 \times 10$ cm. This is provided that the computer system running the HLSL tracking software has comparable or better system specifications than the laptop used during the experiment, and that it has a USB 3.0 port and a DirectX 9 compatible display adapter. Precision and accuracy figures of around 0.3° and 1° respectively can be expected at this rate. Moreover, as the locations of the feature points are reported by the eye tracker, various gaze estimation methods can be utilised. There is also the potential to add a smoothing algorithm to the gaze estimation data in order to improve precision.

In spite of the fact that sampling frequencies in excess of 200 Hz were achieved, the use of the GPU in the eye tracking process still presents a conundrum. Other researchers (Hennessey, Nouredin & Lawrence, 2008) have already demonstrated that it is possible to achieve high sampling frequencies without the use of the GPU. This suggests that one would be better off implementing the parallelism on the CPU rather than on the GPU, unless a way is found to decrease the overhead of transfers to and from the GPU, something that is possible as shown by Mompean et al (2015), or to make use of image processing techniques that would otherwise be too time consuming for the CPU.

Limitations and future work

While the solution discussed in this paper made use of Microsoft technologies, the shader based implementation does theoretically allow for support in Linux based systems through the use of OpenGL and the corresponding shader language GLSL (OpenGL Shader Language). However, the performance of the solution may differ when using OpenGL. Given that Mompean et al (2015) were capable of achieving double the sampling rate using a CUDA based implementation, the choice of HLSL is a definite weakness of the system.

An additional limitation is the use of a 40 point grid, instead of a 45 point grid. The use of the 40 point grid was ill-advised, as it, without an obvious middle point, made choosing a suitable set of calibration points quite difficult (cf. Figure 4). For this reason, a fourteen point calibration procedure was used.

Precision may be improved at higher frequencies by applying a Gaussian blur before performing the edge detection, as this may eliminate some of the graininess present in the eye video at these frequencies. It will provide a stronger case for the use of the GPU as this can be performed fairly rapidly and is easily implemented.

References

- Blignaut, P., & Beelders, T.R (2012). The precision of eye-trackers : A case for a new measure. In *Proceedings of the Symposium on Eye Tracking Research and Applications*, 289–292, Santa Barbara, California, 2012. ACM: New York.
- Blignaut, P. (2013). A new mapping function to improve the accuracy of a video-based eye tracker. *Proceedings of the South African Institute of Computer Scientists and Information Technologists (SAICSIT)*, 56-59, 7-9 October East London, South Africa. doi: 10.1145/2513456.2513461
- Canny, J. (1986). A Computational Approach to Edge Detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, (6), 679 – 698.
- Castaño-Díez, D., Moser, D., Schoenegger, A., Pruggnaller, S., & Frangakis A.S. (2008). Performance evaluation of image processing algorithms on the GPU, *Journal of Structural Biology*, 164(1), 153 – 160.
- Duchowski, A.T. (2002). A breadth-first survey of eye-tracking applications. *Behavior Research Methods, Instruments & Computers*, 34(4), 455–470.
- Duchowski, A.T., Price, M., Meyer M., & Orero, P. (2012). Aggregate gaze visualization with real-time heatmaps. In *Proceedings of the Symposium on Eye Tracking and Applications*, 13–20, Santa Barbara, California, 2012. New York:ACM.
- Hennessey, C., Nouredin, B., & Lawrence, P. (2008). Fixation precision in high-speed noncontact eye-gaze tracking. *IEEE Transaction on Systems, Man and Cybernetics*, 38(2), 289–298.

- Holmqvist, K., Nyström, M., Andersson, R., Dewhurst, R., Jarodzka, H., & Van de Weijer, J. (2011). *Eye Tracking: A Comprehensive Guide to Methods and Measures* (1st ed), New York: Oxford University Press.
- Li, D., Winfield, D., & Parkhurst, D. 2005. Starburst: A Hybrid Algorithm for Video-based Eye Tracking Combining Feature-based and Model-based approaches. *Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*.
- Microsoft. (2016a). *Compute Shader Overview*. Retrieved 05/04/2016 from [http://msdn.microsoft.com/en-us/library/windows/desktop/ff476331\(v=vs.85\).aspx](http://msdn.microsoft.com/en-us/library/windows/desktop/ff476331(v=vs.85).aspx).
- Microsoft. (2016b). *Shader Model 2*. Retrieved 05/04/2016 from [http://msdn.microsoft.com/en-us/library/windows/desktop/bb509655\(v=vs.85\).aspx](http://msdn.microsoft.com/en-us/library/windows/desktop/bb509655(v=vs.85).aspx).
- Mompean, J., Aragon, J., Prieto, P., & Artal, P. *GPU-accelerated high-speed eye pupil tracking system*. 27th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD), 17-24, 17-21 October 2015, Florianopolis, Brazil. doi: [10.1109/SBAC-PAD.2015.17](https://doi.org/10.1109/SBAC-PAD.2015.17).
- Mulligan, J.B. (2012). A GPU-accelerated software eye tracking system. In *Proceedings of the Symposium on Eye Tracking Research and Applications*, 265–268, Santa Barbara, California, March 28 – 30, 2012. New York, NY: ACM.
- NVIDIA (2016). *CUDA ZONE*. Retrieved 05/04/2016, from <https://developer.nvidia.com/cuda-faq>.
- Pacheco, P. (2011). *An introduction to parallel programming*. Burlington: Morgan Kaufmann Publishers.
- Peeper, C., & Mitchell, J.L. (2004). ShaderX: Introductions & tutorials with DirectX 9. In W. F. Engel (Ed.), *ShaderX2: Introductions & Tutorials with DirectX 9* (pp. 393). Plano: Wordware Publishing.
- SensoMotoric Instruments (n.d.). *SMI Gaze & Eye Tracking Systems*. Retrieved 05/04/2016 from <http://www.smivision.com/en/gaze-and-eye-tracking-systems/home.html>.
- Sottile, M.J., Mattson, T.G., & Rasmussen, C.E (2010). *Introduction to concurrency in programming languages*. CRC Press.
- Thompson, C.J., Hahn, S., & Oskin, M. (2002). Using modern graphics architectures for general-purpose computing: a framework and analysis. In *Proceedings of the 35th annual ACM/IEEE international symposium on Microarchitecture*, 306-317, IEEE Computer Society Press, 2002.