Boosting Speed- and Accuracy of Gradient based Dark Pupil Tracking using Vectorization and Differential Evolution

André Frank Krause
Mediablix IIT GmbH
post@andre-krause.net

Kai Essig
Rhein Waal University
kai.essig@rw-kl.eu

ABSTRACT

Gradient based dark pupil tracking [Timm and Barth 2011] is a simple and robust algorithm for pupil center estimation. The algorithm's time complexity of $O(n^4)$ can be tackled by applying a two-stage process (coarse center estimation followed by a windowed refinement), as well as by optimizing and parallelizing code using cache-friendly data structures, vector-extensions of modern CPU's and GPU acceleration. We could achieve a substantial speed up compared to a non-optimized implementation: 12x using vector extensions and 65x using a GPU. Further, the two-stage process combined with parameter optimization using differential evolution considerably increased the accuracy of the algorithm. We evaluated our implementation using the „Labelled pupils the wild" data set. The percentage of frames with a pixel error below 15px increased from 28% to 72%, surpassing algorithmically more complex algorithms like ExCuse (64%) and catching up with recent algorithms like PuRe (87%).

CCS CONCEPTS

• **Software and its engineering → Software performance**.

KEYWORDS

dark pupil tracking, differential evolution, OpenCL, vector extensions

ACM Reference Format:
André Frank Krause and Kai Essig. 2019. Boosting Speed- and Accuracy of Gradient based Dark Pupil Tracking using Vectorization and Differential Evolution. In *2019 Symposium on Eye Tracking Research and Applications (ETRA '19), June 25–28, 2019, Denver , CO, USA.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3314111.3319849

1 INTRODUCTION

The precise estimation of the pupil center of a near-eye image is an essential component of many eye tracking solutions. Because eye tracking is often employed in real-time application scenarios, the pupil tracking algorithm should not only be precise, but also fast. Another requirement might be low power consumption, e.g. for a portable eye tracking system. Accuracy, speed and power consumption are in many respects conflicting objectives, hence designing a good pupil tracking algorithm is not an easy task. Dark pupil tracking methods have quickly evolved from relatively inaccurate algorithms like Starburst [Li et al. 2005] and Isophote [Valenti and Gevers 2008] to very precise and robust methods like PuRe [Santini et al. 2018] and CBF [Fuhl et al. 2018].

Algorithms are often compared with manually labeled databases of eye-images like the Labeled Pupils in the Wild data set (LPW, [Tonsen et al. 2016]). In the LPW publication, gradient based dark pupil tracking [Timm and Barth 2011] does not compare favorable with other, more recent algorithms. But the core principle of the algorithm and its low algorithmic complexity implies a great potential for parallelization and energy efficiency, e.g. an application-specific integrated circuit (ASIC) might be possible.

In this paper, we improve execution speed (section 2.1) and accuracy (section 2.2) of Timm's algorithm.

2 TIMM'S ALGORITHM

Gradient based dark pupil tracking exploits the radial symmetric nature of the pupil and the iris. The idea is that all relevant gradient vectors usually intersect close to the pupil center. In [Timm and Barth 2011], this relationship is formalized and a robust algorithm is presented. The core of the algorithm samples the whole eye-image and tests all pixel positions (x, y) as potential pupil center candidates $\mathbf{c}$. If a gradient vector $\mathbf{g}$ points in the same direction as the normalized direction vector $\mathbf{d} = \mathit{normalize}(\mathbf{x} - \mathbf{c})$ from a possible pupil center $\mathbf{c}$ to the gradient's position $\mathbf{x}$, then the dot-product $dp = dot(\mathbf{d}, \mathbf{g})$ between the gradient- and direction vector will be maximal. For each pupil center $\mathbf{c}$ the dot-product dp between all relevant gradients and their corresponding direction vectors is calculated and summed up. The position where this sum is maximal is the most likely candidate for the pupil center.

To reduce computational load, only gradients with a length above a certain threshold are considered. Still, a natural eye image can contain a lot of relevant gradients and checking each pixel location $\mathbf{c} = (x, y)$ with each gradient can become computational expensive, see section 2.1.1. Smoothing and down-scaling the image can reduce the computational load, but at the expense of accuracy. Using a two-stage process (see section 2.1.2) recovers accuracy while keeping compute times low.

2.1 Performance Optimization

2.1.1 Computational Complexity. Each pixel location of the down-scaled eye-image is processed with all gradients above a threshold. Because both the number of pixel locations and the number of gradients increase with n^2 (assuming a square image with width=height=n), the resulting computational complexity of the algorithm is $O(n^4)$. Fig. 2 shows that a curve with $f(x) = a \cdot x^4$ fits measured compute times. This polynomial time complexity is usually considered tractable, yet the exponent 4 is unfortunate.

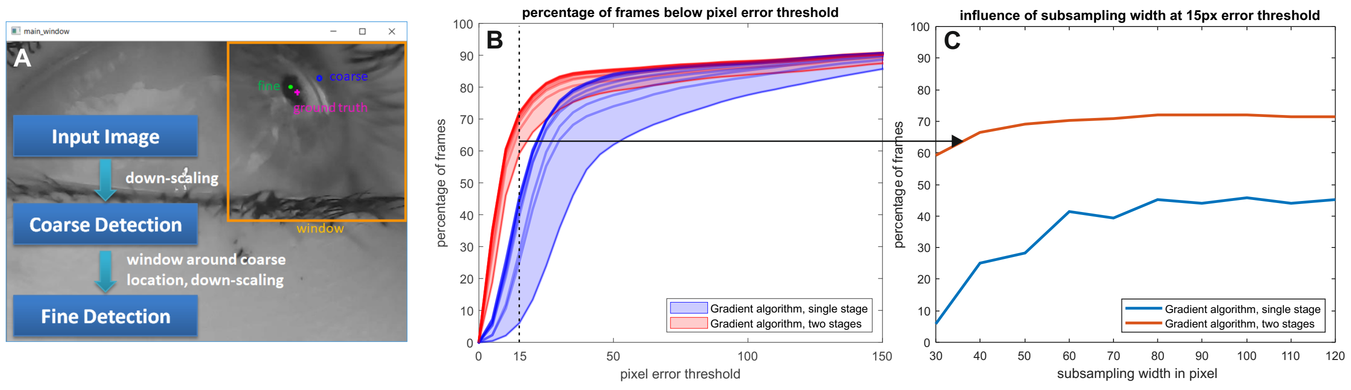


Figure 1: A) Two-stage pupil center estimation. First, the eye image is down-scaled and a coarse pupil center is estimated. Next, a window is defined around the coarse estimation and a second stage computes a finer estimation of the pupil center within this window. B) Accuracy of Timm's algorithm with either one (blue) or two stages (red). Each line represents one down-scaling value, ranging from 30px to 120px in 10 pixel steps. C) shows a vertical slice through fig. B) at a 15 pixel error - threshold.

2.1.2 Two-Stage Architecture. To break-up the rapidly increasing computational demands with image size, we implement a two-stage approach. Cutting the problem size into two halves reduces compute times similar to the divide & conquer algorithm used in quick-sort.

First, the eye image is down-scaled (e.g. from 640 to 85px width) and a coarse pupil center is estimated. Next, a square region-of-interest / window is centered around the coarse estimation. The optimal size of this window depends on the average pupil- and iris size in the eye-image. The windowed image is again down-scaled and a second stage computes a finer estimation of the true pupil center. Fig. 1 shows that this two-stage process clearly improves detection accuracy on the LPW data set even with small down-scaling widths that offer a huge speed benefit (see fig. 2).

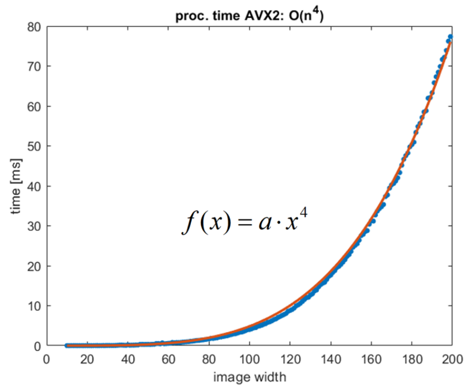


Figure 2: Computational complexity of Timm's gradient based algorithm is $O(n^4)$. The red curve shows a fitted function to measured compute times using AVX2 (blue dots).

2.1.3 CPU Vector Extensions. Many modern CPU's provide vector extensions for data-parallel processing using the single instruction - multiple data (SIMD) principle. With SIMD, a single CPU instruction processes several floating point numbers at once. For example, Intel's AVX-512 (Advanced Vector extensions, [Lomont 2011]) provides instructions that operate on 512 bit registers and can process 16 floating point values in parallel. Other CPU designs provide similar extensions, e.g. ARM's NEON instructions [ARM 2008] and PowerPC's AltiVec [Diefendorff et al. 2000]. The modern RISC-V architecture provides vector extensions [Lee et al. 2014] that are closer to traditional vector-pipelines as found in supercomputers like the famous Cray-1.

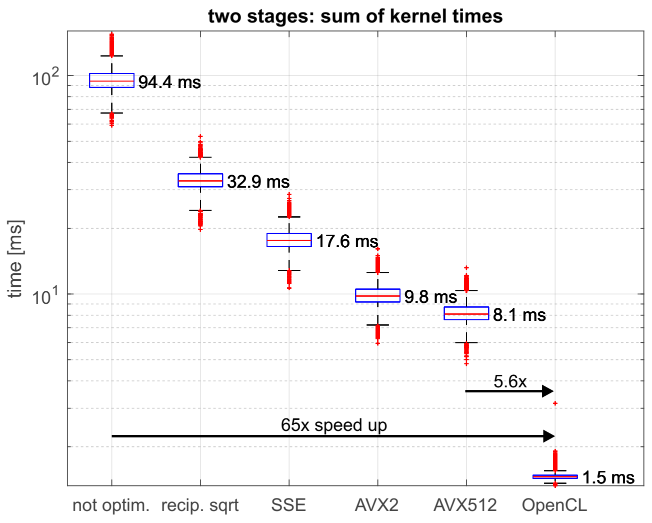


Figure 3: Kernel compute times using fast reciprocal square root, CPU-vectorization and GPU-computation (OpenCL), compared to a non-optimized, purely sequential implementation [Hume 2012]. All computations were performed with two stages, a down-scaling width of 85px and optimized parameters from Differential Evolution. Using OpenCL, a 65x speed-up was achieved.

2.1.4 Speed Optimization. An open source C++ implementation of Timm's algorithm is available on Github [Hume 2012]. This implementation uses OpenCV for most of it’s image processing. But the core computation of the algorithm is implemented in pure sequential code that can be heavily optimized. Removing branching from the inner loop, using a cache-friendly data structure and normalizing the direction vector **d** using fast reciprocal square root (rsqrt) already yields a substantial performance boost (see fig. 3). In a next step, we implemented optimized kernels for SSE, AVX2 and AVX-512 vector extensions. Fig. 4 shows a code snippet of the essential part of the AVX-512 kernel (without the load instructions that store the data into the 512 bit wide registers). For maximum performance, we use a cache-friendly data structure to feed the kernels. We arrange the gradient- and position data in sequential chunks matching the width of the SIMD registers, optimized for sequential DRAM-access. E.g. for AVX-512 we use blocks of 16 floats stored in a contiguous memory container. This way, the cache prefetcher can efficiently stream the gradient data from slow main memory to fast cache memory and finally into the vector registers. Fig. 3 shows the summed kernel processing time for the two-stage algorithm. This includes data-preparation and - in case of OpenCL - the transfer of the data to- and from the GPU device, but excludes pre- and post processing. All tests were performed on the same workstation (see table 1) and with compiler settings selected to maximize speed. If AVX512 is available, a 12x speedup can be achieved. Please note that the code currently is single threaded, so only one of the two AVX512 units of the Xeon CPU is used. With multithreading, performance may double, closing in on OpenCL speed. Independent of kernel compute times, the sum of pre- and post-processing times is 0.45ms ± 0.03ms SD.

Table 1: System Details

CPU	Xeon W-2125 @ 4 GHz
CPU-cache	L1 = 4x32 kB, L2 = 4x1 MB, L3 = 8 MB
RAM	32 GB quad channel DDR4 DRAM @ 1.3 GHz
Graphics Card	GeForce GTX 1080 Ti, 11 GB GDDR5x
GPU Drivers	Nvidia 411.31; CUDA 10.0.140

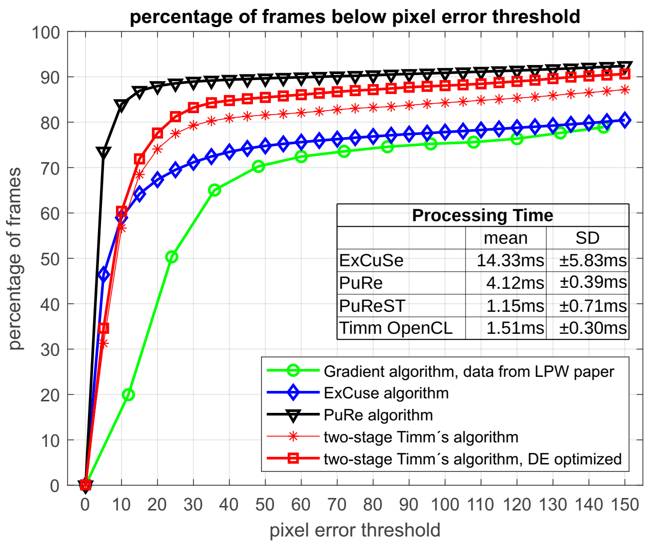
```
// calc difference vector
dx = _mm512_sub_ps(dx, cx);
dy = _mm512_sub_ps(dy, cy);
// calc dot-product
tmp1 = _mm512_mul_ps(dx, dx);
tmp2 = _mm512_mul_ps(dy, dy);
tmp1 = _mm512_add_ps(tmp1, tmp2);
// normalize with reciprocal sqrt
tmp1 = _mm512_rsqrt14_ps(tmp1);
dx = _mm512_mul_ps(dx, tmp1);
dy = _mm512_mul_ps(dy, tmp1);
// dot product with the gradient
tmp1 = _mm512_mul_ps(dx, gx);
tmp2 = _mm512_mul_ps(dy, gy);
tmp1 = _mm512_add_ps(tmp1, tmp2);
// calc max, square and sum
tmp1 = _mm512_max_ps(tmp1, zero);
tmp1 = _mm512_mul_ps(tmp1, tmp1);
return _mm512_reduce_add_ps(tmp1);
```

```
for (int i = 0; i < h; i++)
{
    p.y = i;
    for (int k = 0; k < w; k++)
    {
        g.x = data[idx++];
        g.y = data[idx++];
        p.x = k;
        d = p - c;
        d = d * rsqrt(dot(d, d));
        dp = dot(d, g);
        dp = max(0.0f, dp);
        sum += dp*dp;
    }
}
```

Figure 4: Left: code snippet of the AVX-512 kernel using C++ compiler intrinsics. 16 floating point numbers are processed in parallel with each instruction. Right: OpenCL kernel.

2.1.5 OpenCL GPU Computing. GPU’s perform implicit vectorization by executing serial code on thousands of cores in ten-thousands of lock-stepped threads. For example, a GTX1080TI has 3584 cores (28 streaming multiprocessors with 128 cores each) that can run up to 57344 threads, depending on kernel code complexity. To implement the algorithm on the GPU, we use boost::compute [Szuppe 2016], a thin C++ wrapper around the Open Computing Language (OpenCL, [Stone et al. 2010]). Each pupil center candidate corresponds to one GPU thread that executes the kernel shown in fig. 4. To ensure maximum efficiency, the gradient data must be accessible to all threads simultaneously and as fast as possible. Reading from the GPU’s constant cache can be as fast as reading a register. Therefore, the gradient data should completely fit into the constant cache. With a small size of only 64 kByte on current GPU’s, this results in an upper limit of 85px down-scaled image width.

2.2 Accuracy Evaluation and Optimization



Processing Time		
	mean	SD
ExCuSe	14.33ms	±5.83ms
PuRe	4.12ms	±0.39ms
PuReST	1.15ms	±0.71ms
Timm OpenCL	1.51ms	±0.30ms

Figure 5: Comparison of different dark pupil tracking methods. The insert shows the mean processing time.

2.2.1 Differential Evolution. The algorithm parameters may have complex inter-dependencies within- and between both stages. Further, the number of parameters is rather high, so that an exhaustive grid search is computationally intractable. Therefore, a stochastic optimization method was selected to optimize the parameters. *Differential Evolution* (DE, [Storn and Price 1997]) is a robust, stochastic optimization method with a low number of meta-parameters. The core idea is to interpolate between three randomly selected members a, b, c of the population, calculating a new offspring $y = a + F * (b - c)$ and to perform cross-over with another randomly selected, but distinct member x, such that $y = crossover(x, y)$. Now, if the fitness of this newly created member y is better than that of x, overwrite x with y. DE has only three relevant meta-parameters: 1. the population size P (usually set to 5x - 10x the problem dimensionality), 2. the differential weight F and 3. the crossover probability CR. Here, we used the following values: $P = 50$, $F = 0.5$

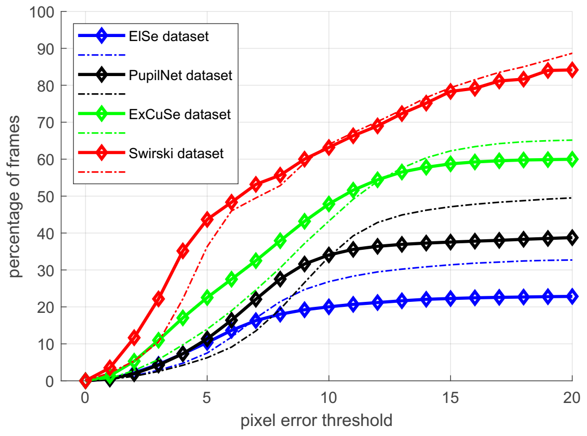


Figure 6: Performance of Timm's algorithm (two-stage) on challenging data sets. Dashed lines: default parameters; bold lines: DE-optimized on LPW data set.

and $CR = 0.5$, similar to suggested values in [Storn and Price 1997]. Evaluating the fitness function over the whole LPW data set (1.3e5 images) is very slow. Therefore, we implemented 2-way tournament selection [Miller et al. 1995], where we tested the fitness of both x and y on the same random subset of LPW with a size of n=3000 images. The fitness value was defined as the percentage of frames within the subset having a pixel error <= 15px. For the two-stage algorithm, there are four parameters per stage and two global parameters to optimize. The first global parameter is the Gaussian kernel size for eye-image pre-smoothing. The second global parameter is the window-size for the second stage. Local parameters are 1. the Sobel filter for gradient calculation, 2. the smoothing kernel width for the weighting map, 3. the gradient threshold and 4. the flood fill post-processing threshold for removing unwanted dark artifacts like eyebrows or glasses frames at the image border.

2.2.2 Differential Evolution Results. The accuracy of the algorithm improves over time and begins to settle around generation 60, see suppl. fig. 1. Hence, the parameters can indeed be fine-tuned using DE. The parameter convergence behavior (fig. 7) can give valuable insights into the sensitivity of the algorithm relative to a certain parameter. For example, the global window size converges to a precise value that optimizes accuracy for the LPW data set. Also, a certain post processing threshold value seems to be optimal for LPW, but only in the first stage. On the other hand, accuracy does not seem to depend much on the local smoothing strength and the local Sobel filter setting. In summary, differential evolution helps to fine-tune parameters, while the two-stage process itself is responsible for the major improvement of accuracy compared to previous publications (see fig. 5, red lines).

Figure 6 shows the accuracy on highly challenging data sets [Fuhl et al. 2015, 2016a,b; Świrski et al. 2012]. Some of the challenges include strong reflections, shadows, eye lid occlusions and fluctuating illumination, see [Fuhl et al. 2016c]. To a limited extend, the DE-optimization on the LPW data set generalizes and carries over: For example, there is an improvement below a pixel error of 10px for the PupilNet, ExCuSe and Świrski data set (fig. 6, dashed lines: default parameters –> solid lines: LPW-DE-optimized parameters).

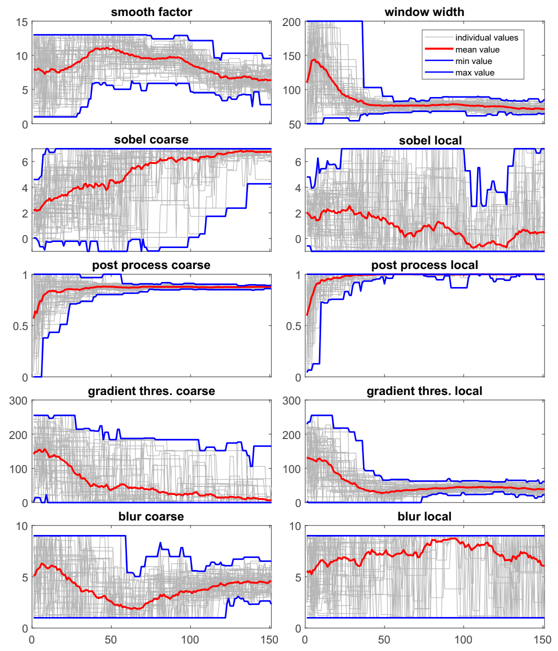


Figure 7: Parameter evolution over time. Some parameters converge early (window size, local threshold and post processing), while others do not seem to have a relevant influence on accuracy (e.g. local sobel and blur settings).

2.2.3 Supplementary Materials. The source code (GPL License), further figures and data can be found at [Krause 2019].

3 CONCLUSIONS AND FUTURE WORK

Using vectorization, a substantial performance boost was achieved. Interestingly, the margin between AVX-512 and the OpenCL implementation is not as large as expected. This is in line with other publications that confound the myth that only GPU's can achieve huge speed-ups, see "Debunking the 100X GPU vs. CPU myth", [Lee et al. 2010]. Yet, the OpenCL implementation shows that Timm's algorithm is well suited for parallel processing using CPU's, GPUS and potentially specialized digital signal processors (DSP). This makes this algorithm interesting for low-power embedded devices with DSP coprocessors, like the OpenCL programmable Texas Instruments C66x DSP [Coombs and Prabhu 2011]. As eyetracking will become an integral part of future mobile AR- and VR devices, it is important to not only investigate the accuracy and speed of algorithms, but also their power consumption. In some scenarios (e.g. foveated rendering) this might be more important than achieving the highest pixel accuracy.

ACKNOWLEDGMENTS

This work was supported by the European Regional Development Fund (ERDF, Grant No.: GE-1-1-047) and the State Government of North Rhine-Westphalia (Grant No.: IT-1-2-001).

REFERENCES

ARM. 2008. ARM NEON support in the ARM compiler. https://www.arm.com/files/pdf/NEON_Support_in_the_ARM_Compiler.pdf.

Joseph Coombs and R Prabhu. 2011. OpenCV on TI’s DSP+ ARM® platforms: Mitigating the challenges of porting OpenCV to embedded platforms. *Texas Instruments* (2011).

Keith Diefendorff, Pradeep K Dubey, Ron Hochsprung, and HASH Scale. 2000. Altivec extension to PowerPC accelerates media processing. *IEEE Micro* 20, 2 (2000), 85–95.

Wolfgang Fuhl, David Geisler, Thiago Santini, Tobias Appel, Wolfgang Rosenstiel, and Enkelejda Kasneci. 2018. CBF:Circular binary features for robust and real-time pupil center detection. In *ACM Symposium on Eye Tracking Research & Applications*.

Wolfgang Fuhl, Thomas Kübler, Katrin Sippel, Wolfgang Rosenstiel, and Enkelejda Kasneci. 2015. Excuse: Robust pupil detection in real-world scenarios. In *International Conference on Computer Analysis of Images and Patterns*. Springer, 39–51.

Wolfgang Fuhl, Thiago Santini, Gjergji Kasneci, and Enkelejda Kasneci. 2016a. Pupilnet: Convolutional neural networks for robust pupil detection. *arXiv preprint arXiv:1601.04902* (2016).

Wolfgang Fuhl, Thiago C Santini, Thomas Kübler, and Enkelejda Kasneci. 2016b. Else: Ellipse selection for robust pupil detection in real-world environments. In *Proceedings of the Ninth Biennial ACM Symposium on Eye Tracking Research & Applications*. ACM, 123–130.

Wolfgang Fuhl, Marc Tonsen, Andreas Bulling, and Enkelejda Kasneci. 2016c. Pupil detection for head-mounted eye tracking in the wild: an evaluation of the state of the art. *Machine Vision and Applications* 27, 8 (2016), 1275–1288.

Tristan Hume. 2012. EyeLike: A webcam based pupil tracking implementation. https://github.com/trishume/eyeLike.

André Frank Krause. 2019. Boosting Speed- and Accuracy of Gradient based Pupil Tracking, Supplementary Materials & Source Code. http://andre-krause.net/publications/etra2019 | mirror: https://mediablix.de/publications/etra2019.

Victor W Lee, Changkyu Kim, Jatin Chhugani, Michael Deisher, Daehyun Kim, Anthony D Nguyen, Nadathur Satish, Mikhail Smelyanskiy, Srinivas Chennupaty, Per Hammarlund, et al. 2010. Debunking the 100X GPU vs. CPU myth: an evaluation of throughput computing on CPU and GPU. *ACM SIGARCH computer architecture news* 38, 3 (2010), 451–460.

Yunsup Lee, Andrew Waterman, Rimas Avizienis, Henry Cook, Chen Sun, Vladimir Stojanović, and Krste Asanović. 2014. A 45nm 1.3 GHz 16.7 double-precision GFLOPS/W RISC-V processor with vector accelerators. In *European Solid State Circuits Conference (ESSCIRC), ESSCIRC 2014-40th*. IEEE, 199–202.

Dongheng Li, David Winfield, and Derrick J Parkhurst. 2005. Starburst: A hybrid algorithm for video-based eye tracking combining feature-based and model-based approaches. In *null*. IEEE, 79.

Chris Lomont. 2011. Introduction to intel advanced vector extensions. *Intel White Paper* (2011), 1–21.

Brad L Miller, David E Goldberg, et al. 1995. Genetic algorithms, tournament selection, and the effects of noise. *Complex systems* 9, 3 (1995), 193–212.

Thiago Santini, Wolfgang Fuhl, and Enkelejda Kasneci. 2018. PuRe: Robust Pupil Detection for Real-Time Pervasive Eye Tracking. *Elsevier Computer Vision and Image Understanding* To Appear (2018).

John E Stone, David Gohara, and Guochun Shi. 2010. OpenCL: A parallel programming standard for heterogeneous computing systems. *Computing in science & engineering* 12, 3 (2010), 66–73.

Rainer Storn and Kenneth Price. 1997. Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces. *Journal of global optimization* 11, 4 (1997), 341–359.

Lech Świrski, Andreas Bulling, and Neil Dodgson. 2012. Robust real-time pupil tracking in highly off-axis images. In *Proceedings of the Symposium on Eye Tracking Research and Applications*. ACM, 173–176.

Jakub Szuppe. 2016. Boost. Compute: A parallel computing library for C++ based on OpenCL. In *Proceedings of the 4th International Workshop on OpenCL*. ACM, 15.

Fabian Timm and Erhardt Barth. 2011. Accurate Eye Centre Localisation by Means of Gradients. *Visapp* 11 (2011), 125–130.

Marc Tonsen, Xucong Zhang, Yusuke Sugano, and Andreas Bulling. 2016. Labelled pupils in the wild: a dataset for studying pupil detection in unconstrained environments. In *Proceedings of the Ninth Biennial ACM Symposium on Eye Tracking Research & Applications*. ACM, 139–142.

Roberto Valenti and Theo Gevers. 2008. Accurate eye center location and tracking using isophote curvature. In *2008 IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, 1–8. https://doi.org/10.1109/CVPR.2008.4587529