Massively Parallelized Iris Recognition with CUDA

Platzer Michael, Weber Thomas, Zaruba Florian

***Abstract*—Iris recognition is one of the most accurate biometric methods in use today. However, the two main approaches for implementing iris recognition algorithms are either on general purpose sequential processing systems, such as generic central processing units (CPUs) or on custom hardware field-programmable gate arrays (FPGAs). Implementing iris recognition algorithms with CUDA represents a novel and low-cost alternative to systems solely based on CPUs or FPGAs. [25]**

***Index Terms*—iris recognition, CUDA, Hough transformation, Log-Gabor filter**

I. Introduction

IRIS RECOGNITION is nowadays one of the most favored biometric identification methods. It provides very high reliably regarding the unique identification of an individual. Quite recently India rolled out a large scale biometric identification program (including iris pattern) where at the time of this writing more than half the population are already enrolled into the program. [9]

Daugman, over the past years, has proposed several algorithms for iris recognition and had large impact on the techniques used for the Indian biometric identity cards as well. In one of his latest works he suggested some preprocessing steps in order to extract the iris from the image and subsequently generate a 2048 bit code from the phase information of several Gabor transformed iris images. Afterwards this IrisCode is matched against a database. [6]

While the algorithm itself seems relatively sophisticated it performs some CPU-intensive operations like localizing the iris, unwrapping it and transforming it into time-frequency space with Gabor wavelets.

In the past two major approaches to that problem haven been proposed. On the one hand, there was an effort to implement the algorithm on general purpose multi-processor platforms as well as some specific digital circuits (FPGA implementation) specifically tailored to the needs of iris recognition. [15]

Our approach on the other hand, tries to exploit the benefits of a general purpose graphics processing unit (GPGPU). In fact iris recognition involves many steps that are perfectly suitable for GPGPU since those algorithms are operating data parallel on images. In addition to the speed up the GPU gains in contrast to the CPU it is far more economically in terms of development and production costs than a custom circuit design on, for example, ASICs or even FPGAs.

In the present work we could demonstrate that an efficient implementation of an iris recognition algorithm on GPUs is indeed possible and brings significant performs gains towards a classical GPU approach. The implementation was tested with images from the CASIA Iris Image Database for Testing Version 1.0 (or IR-TestV1). [29]

II. Related Work

As mentioned before iris recognition plays a crucial role in biometric identification. For this reasons it has been improved tremendously over the past years. There were different approaches taken in the past to achieve maximum accuracy and retaining speed to a reasonable maximum.

As it comes to FPGA implementations in one case [16] the system was implemented in a hardware/software codesign approach by using the Nios II soft-core. Other approaches exploited the additional advantage of re-programmable hardware devices by implementing custom hardware modules to additionally relief the (soft-)core in terms of computational resources. [17], [19], [22].

Current iris recognition algorithms can be grouped into three major types:

1) phase-based methods [8]
2) zero-crossing representation methods [2]
3) texture-analysis-based methods [30]

Regardless of the matching technique used, the system has to reliably extract the iris from a given image. In 1987 the first relevant methodology was presented by Flom and Safir [13]. Some years later Daugman introduced an integrodifferntial operator to finding the inner and outer circle of the iris [8]. This approach still is relevant today and constitutes the basis of many functioning systems [24]. Wildes proposed iris segmentation through a gradient-based binary edge-map construction followed by circular Hough transform. This is the most common approach today and can be found in several minor variants throughout the literature [4], [18], [20].

In the paper at hand the implementation focuses on phase-based methods as proposed by Daugman [8] and preprocessing largely based on the gradient-based edge-map construction according to Borovicka [3]. Our approach differs in the platform used for computation.

Daugman in addition proposed some interesting material about more robust feature extraction techniques such as Fourier-based methods allowing for off-axis gaze to be handled, statistical inferences methods for detecting and excluding eyelashes and more disciplined methods for detecting and faithfully modeling the iris inner and outer boundaries [5].

III. Problem Description

The objective of this application is the reliable and fast recognition of an iris from a given image, which is then compared against a potentially large database. The uniqueness of this approach is that most of the work involved is parallelized and given to a graphics processing unit (GPU) for fast parallel computation. The implementation at hand uses NVIDIA's parallel computation platform Compute Unified Device Architecture (CUDA)[1].

[1] www.nvidia.com/object/cuda_home_new.html

The task mainly consists of three steps: preprocessing and feature extraction, generation of a unique bit-sequence-representation of the iris and matching this sequence against a large dataset. At least the two former tasks are highly parallelizable and, as shown later, can largely profit from the computational power of GPUs.

The GPU implementation will pose an additional advantage on the task of iris recognition: With the emerging market of embedded graphics and general purpose processing unit platforms like the Tegra family from NVIDIA, our approach presents a low cost and low power alternative to traditional CPU or custom hardware implementations [1].

IV. System Overview

This section presents our algorithm and techniques used for implementation of an iris recognition system. A general overview is depicted in Figure 1. The main idea behind our

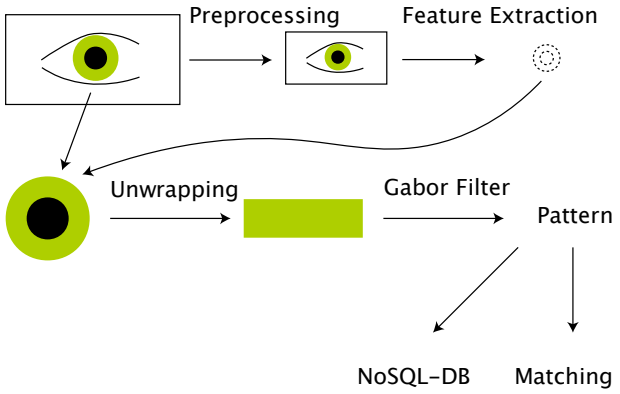


Figure 1: Program flow of iris detection

approach is to perform most of the tasks at hand on a GPU. It turned out that the steps of preprocessing, feature extraction and most subtasks of unwrapping and Gabor filtering can be performed directly on the graphics unit therefore reducing the need on costly memory transfers between the host (CPU) and device (GPU). Furthermore multiple images can be scheduled to be processed simultaneously on the GPU (e.g. using multiple CUDA streams) thus making it even possible to analyze a real-time video-stream.

A. Preprocessing

The task of preprocessing involves various subtasks. The first one being a *conversion from a (possibly) colored image to a grayscale* version. This reduces the amount of data necessary to handle the upcoming tasks. Color conversion is highly parallelizable since there is no dependency of individual pixels whatsoever. In most cases the conversion to greyscale does not influence the detection probability of the iris since most iris scanners in use, capture their images using infrared light in a darkened environment (closed glasses). A typical acquired image can be found in figure 2.

The second step consists of an *image size reduction* of the greyscale input image to a fixed size. This, on the one hand, has the advantage of having an image that has at least its width a multiple of 32 (which turns out to be very efficient for processing on CUDA devices as it exploits the full warp size of a streaming multiprocessor subunit on the graphics

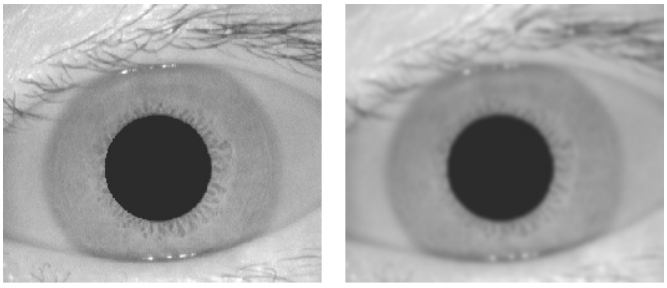


Figure 2: a) Test image from the CASIA Iris Image Database for Testing Version 1.0 [29]. b) Image after Gaussian blur.

chip). On the other hand, as will be discussed in the upcoming section, center detection is a relatively costly tasks in terms of processing power on the size of the image. A larger image does not improve center detection, on the contrary parameters of the tasks can be specifically tailored to the needs of a certain image and iris size (iris size in proportion to the overall image size). Resizing is accomplished by simply picking every other pixel that is needed for the targeted image size. It turned out that this approach is reasonably good suited for this purpose.

The third step involved is a *noise reduction* of the image accomplished by a Gaussian blur. It turned out that resizing the image poses an additional advantage: The Gaussian blur kernel depends largely on the image size. By constraining the image width one can set the filter kernel to a fixed size therefore making it possible to put the kernel into constant memory of the GPU. A second measure to improve computation speed was the fact that the Gaussian blur kernel has the property of being separable. This means that instead of performing one 2D convolution with the filter kernel one can perform two 1D convolutions therefore reducing the amount of arithmetic operations needed (instead of $n * m$ multiplications for each output pixel, where n and m are the width and height of the image only $n + m$ multiplications are needed) [23]. Please see the appendix for a proof on the specified properties.

All three steps are highly profiting from the computational power and the GPU specific device architecture. The convolution in particular performs significantly faster than on traditional CPU systems. In addition to the above mentioned algorithmic details all loops are unrolled at compile time (since the size of the filter kernel is known) therefore reducing unnecessary jumps in the program flow and maximizing warp efficiency. In the present implementation all the memory access is coalescence in order to achieve maximum throughput.

B. Feature Extraction

Feature extraction is concerned with the task of identifying the iris in the image at hand. It basically accomplishes this task in two steps. First the eyes center is detected and afterwards, based on the center two contour lines are identified that form the iris boundaries.

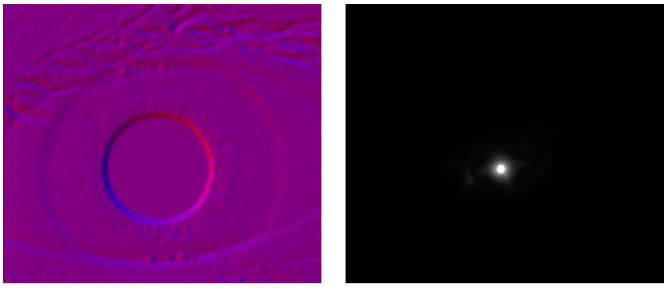
1) Center detection: Regarding center detection the input image is convoluted with a Sobel filter. The Sobel filter consists of an 3x3 matrix for x (horizontal) and y (vertical) direction respectively [28]. The corresponding matrices can

be seen in figure 3. Basically, it is a discrete differentiation

$$G_x = \begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix} \quad G_y = \begin{pmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{pmatrix}$$

Figure 3: Vertical and horizontal Sobel filter kernels.

operator, computing an approximation of the gradient of the image intensity function. It performs poorly for high frequency variations in the image thus founding the reason why it is a good idea to perform a previous noise reduction step. The Sobel operator is separable as well thus making it easily possible to reuse the convolution function implemented for the Gaussian blur. In addition all advantages regarding the implementation of the Gaussian blur apply to the Sobel operation as well. In figure **??** one can see the image after Sobel filtering. As it is clear from the example image the Sobel operator the operator produces an image that emphasizes the edges and transitions of the image on which it is applied. The



a) b)

Figure 4: a) Gradient map after Sobel opration (blue gradient component in x-direction and red gradient component in y-direction) b) heat map of center detection; center of gravity at (123, 133) pixels (at an overall image size of 320 pixels width and 280 pixels height).

next step involved is a *Hough transform*. This transformation is a feature extraction technique mostly concerned with finding imperfect instances of certain parameterized curves [27]. In our cases we are interested in concentric circles and their respective center. The general idea is that perpendiculars to edge points of a circle cross in the center of the circle. Therefore, if we draw perpendicular lines to every edge point of our edge map, we should obtain bright 'hot spots' in the center of the circles.

In fact the implementation iterates through all possible (x, y)-coordinates and looks for radii of size k. In total running the Hough transform accumulates to a runtime of $O(x{\cdot}y{\cdot}k)$ which is, if we assume $x \geq y \geq y$, $O(x^2)$. For each possible center it looks for perpendicular lines around the given radius. If matching lines are found (e.g. the direction of the gradient points to the given center within a certain threshold of $[-\frac{\pi}{12}, \frac{\pi}{12}]$) a local accumulator is incremented. By executing this algorithm we are left with a heat-map of potential centers (see figure 4). In a next step we are looking for the *average mean* of the heat-map which denotes a potential center of the iris. We are assuming that only one eye is present in the image and that it approximately takes up more than two thirds of the image. Those are crucial assumptions as it comes to parameter choices for the Hough transformation: By restricting the potential parameter space (radius size, margin areas, etc.) one can significantly speed up the transformation. Furthermore we cube the values of individual radii in order to favor radii that perform reasonable good for most points on the radius over radii that are only accidentally well suited for some perpendicular lines.

Hough transformation highly benefits from a parallelized optimization. Potential centers are examined individually for each coordinate an can therefore be straight forwardly parallelized since there is no dependency between the dedicated coordinates whatsoever. For each radius one thread is launched and after the computation its result is stored in a shared variable structure and afterwards reduced (by addition) to a single value.

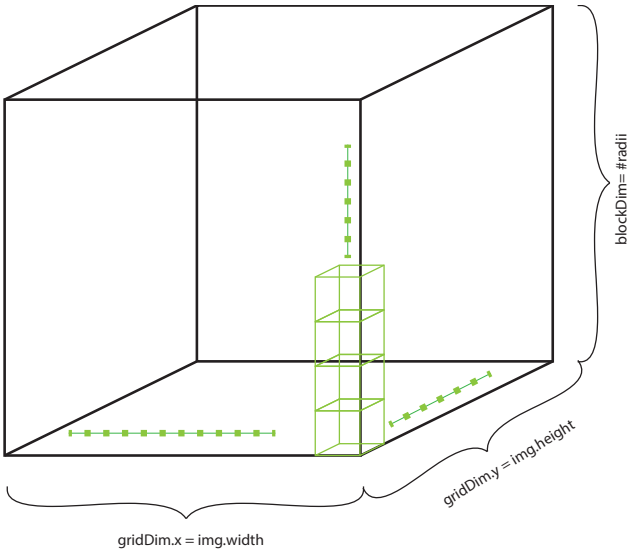


Figure 5: Thread structure of Hough transformation

The thread structure can be examined in figure 5. Each block corresponds to a single thread. Blocks that are stacked denote threads of an individual block. Threads that share the block have a particular useful property namely, that they can access shared variables. Shared memory space performs significantly faster on thread access than global memory and is therefore often used for reductions of all kinds. We set our radius parameter space to a multiple of 32 to gain an additional advantage: Since CUDA Version 3.0 NVIDIA introduced a feature called warp shuffle functions. Essentially those features enable the GPU to pass different sorts of data between threads of a warp (a group of 32 threads that are implicitly synched by lock-step execution). This provided us with a toolset to exchange data (e.g. for a reduction) tremendously faster since there is no synchronization involved and data is directly stored in the cache blocks of each streaming multiprocessor respectively. This feature enabled us to reduce the vote values fore each radius on a warp basis which indeed gave a big speed up [21].

As a last speed improvement the gradient map (in polar coordinates) resulting from the Sobel operation was stored in texture memory. Texture memory has the advantage that a specific pattern of localized memory access is cached and therefore retrieved faster. We where somehow lucky that this access pattern largely correlates to the usage by the

Gough transformation (memory access should have 2D - spatial locality, e.g. its neighbors). On further advantage of the texture memory was the fact that an access needs not to be discretized (as it is the case for array access) and that an offset is linearly interpolated by the hardware in case of access outside the texture area or between to pixels or texels as they are called in case of texture memory. This has the advantage of limiting control flow structures for boundary detection and allows for coherent program flow of threads within the block (more accurately the warp).

The last step to obtain specific center coordinates involves finding the center of gravity. This is accomplished by averaging over the heat-map in x and y direction. This task can partly benefit from the optimization measures taken for the Hough transform. The task of additive reducing values directly applies to averaging as well. In addition a threshold value works as a low pass filter on the heat-map and discards values that are not significant for finding the center of gravity.

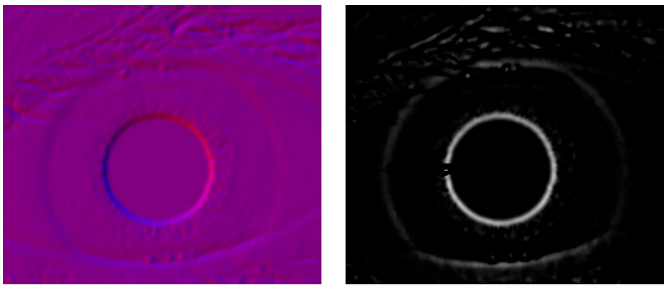
2) Radii detection: The internals of radii detection and unwrapping are closely related. In a first pass we are enhancing the circular features of the iris' edges by computing the target angle (to be part of a circle) for each pixel (x, y) and subtracting the actual angle.

$$\varphi_t = \tan^{-1}\left(\frac{y - y_{\text{center}}}{x - x_{\text{center}}}\right)$$
$$\Delta\varphi = \varphi_{(x,y)} - \varphi_t$$

The angle difference is then fed to a standard Gaussian distribution which favors values close to zero.

$$f(\varphi_{(x,y)}) = \exp\left(-\frac{(\Delta\varphi)^2}{\sigma^2}\right)$$

Subsequently this factor is multiplied with the Euclidean norm of the gradient map at position (x, y). An example of such an enhancement is displayed in figure 6. For further

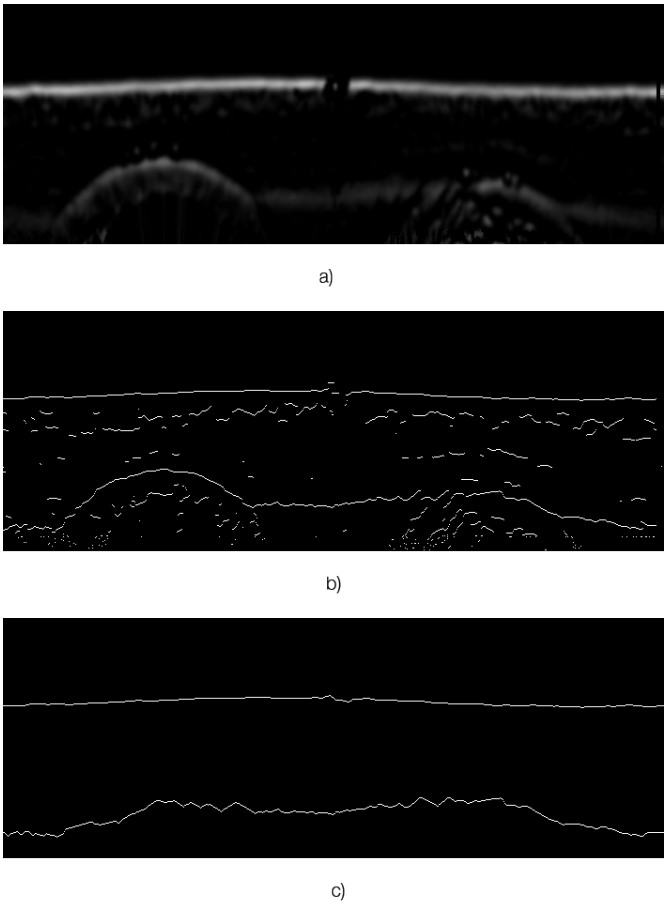


a) b)

Figure 6: a) Gradient map after Sobel operation b) Normalized gradient map, curves that are not concentric to the computed center are suppressed

processing, the iris is projecting from a circular representation to a quadratic image e.g. it is unrolled with regard to its previously calculated center coordinates. In this first step where the exact iris borders are still unknown it is unrolled with respect to the greatest possible radius. The unrolled iris from figure 6 can be seen in figure 7.

The unrolling algorithm directly interacts with the gradient map stored in the texture memory which was used in the previously performed Hough transformation. This has the advantage that memory access to 'unspecified' memory locations is transparently handled and linearly interpolated. This



a)

b)

c)

Figure 7: a) unrolled and blurred iris b) local maxima are set to the highest value c) traces of iris boundaries

makes a lot of sense when considering an unroll operation: Since samples taken from regions close to the center are over-sampled whereas iris regions which are further away from the center tend to be under-sampled. This task is easily parallelizeable by launching kernels for each pixel in the target image (the image size of the unrolled iris is fixed). Unrolling corresponds to a *transformation to dimensionless polar coordinates* (where the image width corresponds to the angle $\varphi \in [0, 2\pi]$ whereas the height corresponds to the radius $r \in [0, 1]$).

It is quite unpractical for decisions on iris boundaries to work with blurred contour lines, ergo we want to reduce the contour lines to sharp decision boundaries. This is accomplished by searching for local maxima and setting these values to a easily distinguishable value. The output of this step can be seen in figure 7 b). Basically the algorithm compares each value with the value of its predecessor and successor. If the current value is greater than the other two values it implies that it is a local maximum. The algorithm additionally uses a certain threshold to avoid unintentionally considering noise as an extrema.

The last step that remains for the task of radii detection is to decide for the correct contour lines of the iris. This is done with an Bayesian decision approach that assigns costs to each individual trace. Two of the traces with the least costs are then chosen as possible iris boundaries. The image is traversed from left to right. For each pixel in the first column a corresponding pixel in the next 60 columns are

searched. The costs for the route increase proportional to the distance of the next pixels to the estimated route (e.g. a straight line). The search space has a triangular shape which implies that only pixels within this window are considered for potential successors of the line segment. One such an

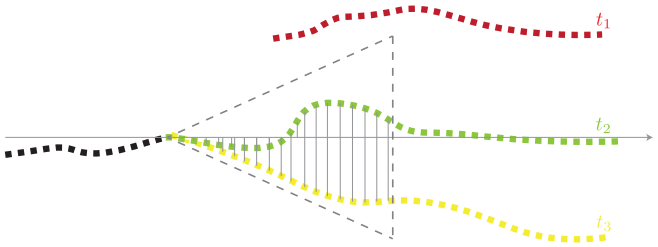


Figure 8: The tracing process is depicted for 3 different traces t_1, t_2 and t_3. While t_1 is not recognized t_2 has a lower cost function assigned than t_3.

example of line tracing can be seen in figure 8. The first trace (t_1) is not recognized since it lays outside the decision window. This allows for successfully suppressing artifacts that can accidentally occur within the iris. The second trace t_2 has a lower cost function than t_2 since it progresses closer to the optimal line segment.

The resulting traces and the center are then calculated for the original image by multiplication with the image compression factor. The original image and the traces shown in the original image form a cut mask (see figure 9).

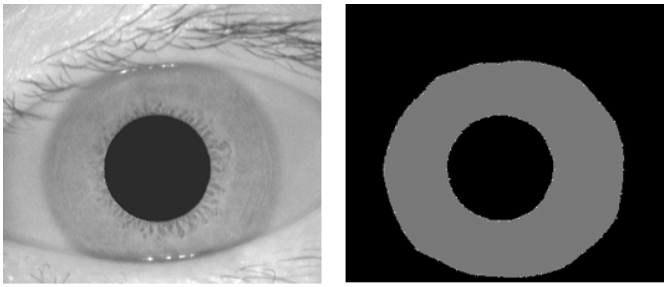


Figure 9: a) original iris image b) generated cut mask

C. Unrolling

The same unrolling operation used in the previous step is used for unrolling the final iris from the original image given the cut mask. Such an unrolled iris can be seen in figure 10 a). As one may notice: The iris unrolling is imperfect due to various reasons, one being partial coverage of the iris by the eyelid. Some algorithms account for that imperfection by ignoring the covered part in the later step of feature generation and pattern matching (for example Daugman [5]). In the current implementation we do not account for that.

It turns out that the spectrum used for representation of the iris is rather narrow. A histogram for the example image is depicted in figure 11. In order to enhance the performance in the later step of pattern matching we are equalizing the image over the whole grey space of 256 bit in the current implementation. This approach usual increases the global contrast of many images. The process of histogram

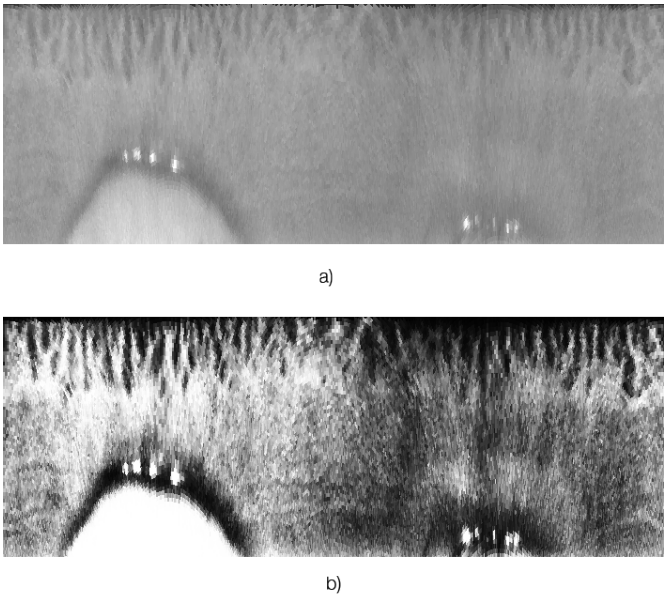


Figure 10: a) original cropped iris pattern b) equalized iris pattern - single features are more easily detectable

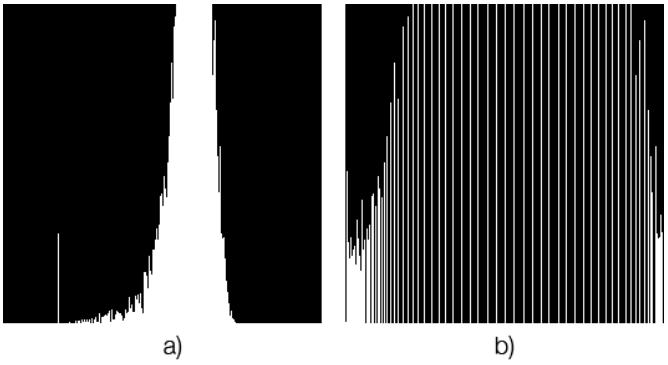


Figure 11: a) Histogram of the unrolled iris image. 256 grey values on the x-axis and overall count of the respective values on the y-axis b) Histogram of the equalized iris image

equalization generally distinguishes two concepts of thinking. Either the processes is viewed as a image change or as a palette change. While in the first individual pixels are immediately substituted in the latter the palette is changed. In our implementation first the palette is substituted and afterwards according to the changed palette the image is substituted. A cumulative distribution function (cdf) is generated according to the previously generated histogram.

$$cdf_x(i) = \sum_{j=0}^{i} p_x(j) \qquad i \in [0, 255]$$

And the new palette is then generated by:

$$pal(i) = cdf_x(i) \cdot 255 \qquad i \in [0, 255]$$

The corresponding generated histogram and image are depicted in figure 10 b) and figure 11 b) respectively.

D. Pattern Generation

The next part deals with the generation of a pattern (or feature vector) that uniquely determines the iris. The feature vector should have the properties to be as small as possible

(in terms of byte size) in order to be stored, retrieved and matched as fast as possible from a potentially large dataset. In addition the feature vector should be largely invariant to rotations, dilatations and images properties like brightness, hue and contrast. Various approaches to this problem are persecuted in the literature. We decided our implementation in favor of a Log Gabor filter and subsequent calculation of an average absolute deviation of gray values (our implementation is mostly based on the ideas of Seif et al. [26]).

1) Log Gabor filtering: A widely used approach in pattern recognition is to generate a frequency representation of the pattern that needs to be analyzed. This is done by applying variations of the short-time Fourier transform to the image. There exist numerous such transformations, which are able to simultaneously analyze the space and frequency characteristics of a given signal. The Gabor filter, invented by Dennis Gabor, is one such transformation, which has the characteristic to minimize the product of its standard deviations in time and frequency domain. The Gabor filter thus has the advantage of an optimal tradeoff between space and frequency representation [14].

The Gabor filter is in fact the multiplication of a Gaussian function with a complex exponential and is given by the following equation:

$$g(x) = \exp\left(\frac{-(x-x_0)^2}{a^2}\right)\exp\left(-if(x-x_0)\right)$$

where f is the frequency of the filter and a sets standard deviation of the Gaussian and thus determines the bandwith of the filter.

The Gabor filter has the following frequency response:

$$G(f) = \exp\left(-(f-f_0)^2a^2\right)$$

where f_0 is the center frequency of the filter and a determines the bandwith of the filter.

The Log Gabor filter in particular is one improvement of the Gabor filter in the way that fits the statistics of natural images. In fact there is some evidence that Log Gabor filtering performs similar to how the human visual system processes information [12] [7]. In addition the Log Gabor filter has a zero DC component, hence the response does not depend on the mean value of the signal.

The one dimensional Log Gabor function has the frequency response of:

$$G(f) = \exp\left(\frac{-(\log(f/f_0))^2}{2(\log(\sigma/f_0))^2}\right)$$

where f_0 is again the center frequency and σ sets the bandwith of the filter.

Figure 12 shows the difference between the frequency responses of a Gabor and a Log Gabor filter. The Gabor filter over-represents low frequencies and under-represents high frequencies. Additionally, there is a non-zero DC component. The Log Gabor filter equally represents low and high frequencies. In fact, when the frequency response of a Log Gabor filter is shown with a logarithmic frequency scale, one can see that the response is symmetric with respect to the center frequency.

Whereas the Gabor filter also has an equation in the time domain, there is no analytical equation for the Log Gabor filter in the time domain. In order to generate a time-domain representation of this filter, which can then be applied to

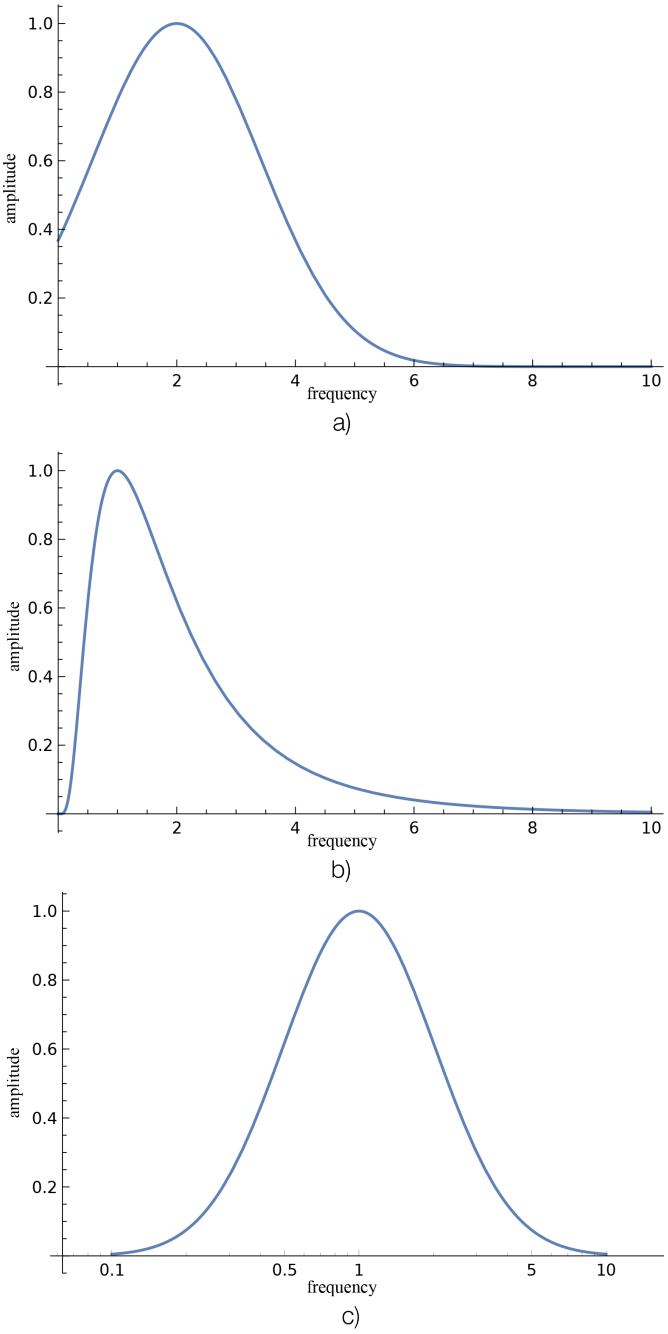


Figure 12: frequency response of a) Gabor filter b) Log-Gabor filter c) Log-Gabor filter depicted with a logarithmic frequency axis

the image, the frequency response must be sampled and its discrete values must be transformed into the time domain using the inverse discrete Fourier transform [11]. Figure 13 shows the time domain representation of a Gabor and a Log Gabor filter. The curve shape of these two filters is similar in the time domain. Although the real and imaginary parts of the filters are shown, we will only use the real part of the Log Gabor filter for further filtering.

So far we only considered one dimensional Log Gabor filters. For the task of image processing it is useful to take a two dimensional version of the Log Gabor filter into consideration. For this purpose the transfer function is multiplied with a second Gaussian that additionally restricts the window in the second dimension and therefore imposing

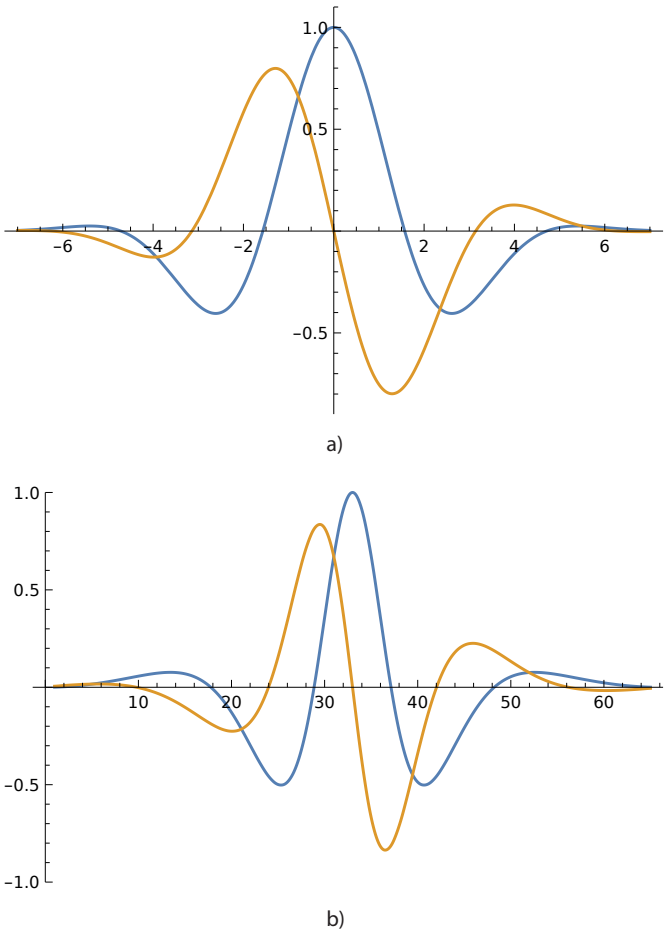


Figure 13: Time domain representation of a) a Gabor filter and b) a Log Gabor filter (blue graph corresponds to the real part of the filter while the yellow curve depicts the imaginary part)

a direction on the filter [10]. A representation of such a two dimensional filter in the space domain is depicted in figure 14. One can identify the Log Gabor function along the x-axis and the Gaussian function along the y-axis.

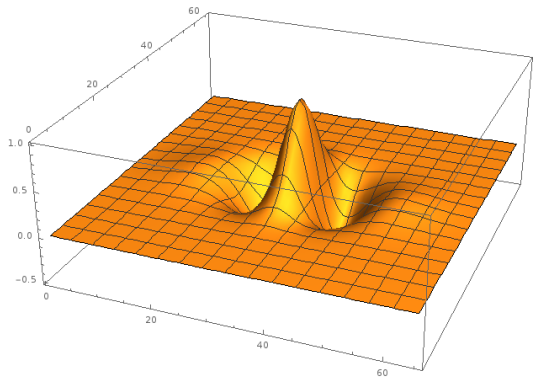


Figure 14: 3D representation of a Log Gabor filter's real part in the space domain.

For this particular implementation we are only considering two directions for reasons explained further below. By varying frequency and orientation parameters it is possible to generate a filter bank that enhances different features of the

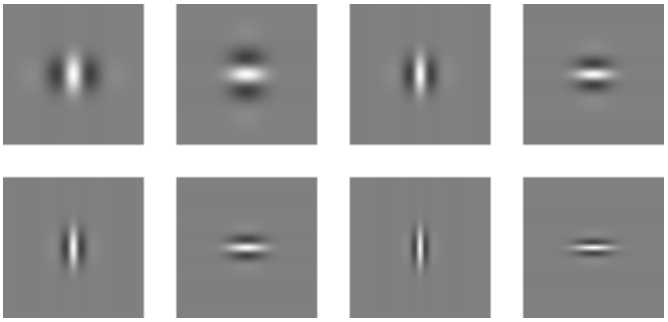


Figure 15: Small subset of the Log Gabor filter bank used, showing different frequencies and orientations

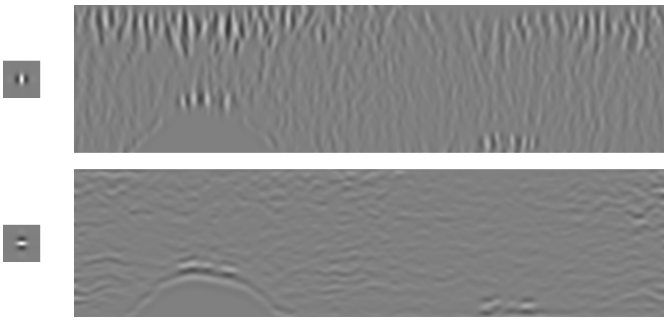


Figure 16: Iris pattern after Log Gabor filtering with two filters of the same frequency but distinct directions.

iris. The creation of the filter bank and therefore the choices of the parameters is somehow unique to the application. No general approach is known for this task.

A small subset of the filter bank used in the implementation at hand is shown in figure 15. One can spot the different filter directions and frequencies in use. The bright parts of the filter correspond to large (amplifying) values while the dark part essentially conforms to suppressing parts of the filter. In particular the results of the convolution with two specific filters is shown in figure 16. One can clearly see that the filter enhances features that correspond to the parameters of the filter (e.g. frequency and direction).

Only considering this two directions imposes the advantage that the Gabor filtering becomes a separable operations thus significantly increasing processing speed while the detection probability remains at about the same level. This characteristic is most likely based on the basic features of an iris that most pigments proceed perpendicular to the inner and outer radius.

2) Feature Vector Generation: The second action required for pattern generation is reducing the filtered iris to scalar values to make storage and retrieval of data from the database feasible. For this step various methods have been proposed. While some are based on the phase information of the filtered image (like Daugman's approach [6]), we are reducing the entire filtered image to a single value: the mean absolute deviation of the grey values. This has among others the advantage that this single value per filter is translation independant. A downside of this approach is that a huge number of filters needs to be applied to the image to generate a enough values for a sufficiently good feature description of the iris. Fortunately, the performance gains resulting from the GPU implementation easily compensate the additional computational complexity.

The mean absolute deviation is given by:

$$D = \frac{1}{W \cdot H} \sum_{x=0}^{W} \sum_{y=0}^{H} |f(x,y) - \mu|$$

where W and H are respectively the width and height of the filtered image, $f(x,y)$ is the grey value of the pixel at location (x,y) and μ is the mean of all the grey values of the image. This value is calculated for each filter. Our current implementation uses a filter bank with 32 filter (16 frequencies with 2 orientations for each), thus generating a feature vector with 32 values.

E. Pattern Matching

The final task of iris recognition is now to match the previously generated feature vectors using a database. A single feature vector consists of 32 floating point values, which means that there is a huge amount of data to be stored and matched. This procedure cannot be done with common SQL database engines because they are not able to perform read/write commands to thousands of big sets of data within few microseconds. As a consequence the conventional SQL approach has to be dropped and noSQL databases, which are explicitly designed for such purposes, were chosen. The disadvantage of such noSQL databases (e.g. mongoDB, OrientDB, CouchDB, Lotus Notes) is the weak data consistency which can be disregarded in the context of iris recognition concerning the relation of one or two bytes in relation to 32 floating point variables. Since the database mongoDB, which is used in this implementation, is completely written in C, there would be the option to implement parts of this database in CUDA to accelerate the processing speed. This kind of custom modification requires deeper knowledge of the database and were already done for specific purposes [2]. NoSQL is not very wide-spread, therefore there are not many projects which deals with CUDA implementations of such databases.

With the generated feature vector it is now possible to search for matching iris vectors in the database. This is done by calculating the distance between to vectors as follows:

$$dist = \sqrt{\sum_{k=0}^{32} (vector1[k] - vector2[k])^2}$$

where $vector1$ and $vector2$ are the feature vectors of two iris images. A threshold is then used to decide whether the images depict the same eye. In our implementation a treshold of 0.07 has shown to differentiate between a hit and a miss.

V. Implementation

The above mentioned algorithm was implemented in NVIDIA's CUDA runtime. The CUDA runtime provides some very efficient intrinsics that give an enormous speed up. Most of them were already discussed at the point where that particular part of the algorithm was discussed. As discussed previously iris recognition mostly deals with images (at least the steps involving feature detection and pattern generation) which gives the GPU a tremendous dominance over the CPU. Costly operations like transferring data from the host system to the device are reduced. In essence the image is copied only once to the device and kept there until the small feature vector is retrieved from the device.

In our implementation we take additional advantage of CUDA specific technology like the texture and constant memory and advanced features like warp functions therefore significantly speeding up the overall procedure. Currently the only thing that is still implemented sequentially on the CPU is the part of pattern matching including the database access. Although there are some (unsupported) CUDA extensions to MongoDB it only makes sense to parallelize some part of the database access.

The implementation can be accessed directly from GitHub[3]. The program is split into parts that are executed on host system (*.c files) and device files (*.cu files). The Makefile includes a test run (`make test`) that compiles and compares various images from the CASIA Iris Image Database. The following listing gives an overview of the most important algorithm implementations (approximately ordered by calling sequence):

- `pixel.cu`: Contains diverse helper image manipulation functions, e.g. color to grey conversion, palette substitution for histogram equalization, gradient normalization, image resize and coordinate transformation.
- `convolve.cu`: Accommodates separable convolution code used for Sobel, Gabor and Gauss filtering.
- `hough.cu`: Performs the Hough transform.
- `histogram.cu`: Parallel histogram generation.
- `unroll.cu`: Image unrolling based on a given center and outline trace.

VI. Results

As at the beginning of the porting process to a parallelized version we had most of the feature extraction part implemented in a sequential reference algorithm. While the sequential algorithm operates poorly in terms of computation time, the CUDA version significantly outperformed the CPU version of a factor of 20 only on the part of feature extraction. The implementation of separable convolution on the GPU imposes additional computational advantages on (Log-)Gabor filtering therefore exceeding the running time of a sequential implementation.

VII. Conclusion

Summarized it was shown by the present implementation that iris recognition is another field that can largely benefit from the parallel architecture of general purpose graphics computing. In addition evolving system on a chip technology (like NVIDIA's Tegra architecture) make an iris recognition that performs most of its costly operations on the GPU particularly interesting. Because it frees the central processing unit from the rather cumbersome task of image processing, embedded systems (e.g. wearables) can gain advantages in terms of computational power and energy usage.

Furthermore the parallelized version makes it possible to generate large biometric datasets in relatively short time by using GPUs in high performance computing (HPC).

In addition the CUDA implementation provide a low-cost alternative to systems based on FPGA's or ASICS.

[2] https://github.com/harishd10/mongodb

[3] https://github.com/be4web/cuda-iris/

REFERENCES

[1] T. Akenine-Moller and J. Strom. Graphics processing units for handhelds. *Proceedings of the IEEE*, 96(5):779–789, May 2008.

[2] Wageeh W Boles and Boualem Boashash. A human identification technique using images of the iris and wavelet transform. *IEEE transactions on signal processing*, 46(4):1185–1188, 1998.

[3] Jaroslav Borovicka. Circle detection using hough transforms documentation. *Image Processing and Computer Vision*, 2003.

[4] Jiali Cui, Yunhong Wang, Tieniu Tan, Li Ma, and Zhenan Sun. A fast and robust iris localization method based on texture segmentation. In *Defense and Security*, pages 401–408. International Society for Optics and Photonics, 2004.

[5] J. Daugman. New methods in iris recognition. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, 37(5):1167–1175, Oct 2007.

[6] John Daugman. How iris recognition works. *Circuits and Systems for Video Technology, IEEE Transactions on*, 14(1):21–30, 2004.

[7] John G Daugman. Uncertainty relation for resolution in space, spatial frequency, and orientation optimized by two-dimensional visual cortical filters. *JOSA A*, 2(7):1160–1169, 1985.

[8] John G Daugman. High confidence visual recognition of persons by a test of statistical independence. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 15(11):1148–1161, 1993.

[9] John G Daugman. http://spie.org/x108321.xml, July 2015.

[10] Fischer et al. Self-invertible 2d log-gabor wavelets. *International Journal of Computer Vision*, 75(2):231–246, 2007.

[11] Fischer et al. How to construct log-gabor filters? 2009.

[12] David J. Field. Relations between the statistics of natural images and the response properties of cortical cells. *JOSA A*, 4(12):2379–2394, 1987.

[13] L. Flom and A. Safir. Iris recognition system, February 3 1987. US Patent 4,641,349.

[14] Dennis Gabor. Theory of communication. part 1: The analysis of information. *Journal of the Institution of Electrical Engineers-Part III: Radio and Communication Engineering*, 93(26):429–441, 1946.

[15] A. Hematian, A.A. Manaf, S. Chuprat, R. Khaleghparast, and S. Yazdani. Field programmable gate array system for real-time iris recognition. In *Open Systems (ICOS), 2012 IEEE Conference on*, pages 1–6, Oct 2012.

[16] R. Hentati, M. Bousselmi, and M. Abid. An embedded system for iris recognition. In *Design and Technology of Integrated Systems in Nanoscale Era (DTIS), 2010 5th International Conference on*, pages 1–5, March 2010.

[17] Zhou Hu-lin and Xie Mei. Iris biometic processor enhanced module fpga-based design. In *Computer Modeling and Simulation, 2010. ICCMS'10. Second International Conference on*, volume 2, pages 259–262. IEEE, 2010.

[18] Junzhou Huang, Yunhong Wang, Tieniu Tan, and Jiali Cui. A new iris segmentation method for recognition. In *Pattern Recognition, 2004. ICPR 2004. Proceedings of the 17th International Conference on*, volume 3, pages 554–557. IEEE, 2004.

[19] Judith Liu Jimenez, Raul Sanchez Reillo, Almudena Lindoso, and Oscar Miguel Hurtado. Fpga implementation for an iris biometric processor. In *Field Programmable Technology, 2006. FPT 2006. IEEE International Conference on*, pages 265–268. IEEE, 2006.

[20] W-K Kong and D Zhang. Accurate iris segmentation based on novel reflection and eyelash detection model. In *Intelligent Multimedia, Video and Speech Processing, 2001. Proceedings of 2001 International Symposium on*, pages 263–266. IEEE, 2001.

[21] Justin Luitjens. http://devblogs.nvidia.com/parallelforall/faster-parallel-reductions-kepler/, July 2015.

[22] F Mohd-Yasin, AL Tan, and MI Reaz. The fpga prototyping of iris recognition for biometric identification employing neural network. In *Microelectronics, 2004. ICM 2004 Proceedings. The 16th International Conference on*, pages 458–461. IEEE, 2004.

[23] Victor Podlozhnyuk. Image convolution with cuda. *NVIDIA Corporation white paper, June*, 2097(3), 2007.

[24] Hugo Proença and Luıs A Alexandre. Iris segmentation methodology for non-cooperative recognition. In *Vision, Image and Signal Processing, IEE Proceedings-*, volume 153, pages 199–205. IET, 2006.

[25] Ryan N Rakvic, Bradley J Ulis, Randy P Broussard, Robert W Ives, and Neil Steiner. Parallelizing iris recognition. *Information Forensics and Security, IEEE Transactions on*, 4(4):812–823, 2009.

[26] A Seif, R Zewail, M Saeb, and N Amdy. Iris identification based on log gabor filtering. In *Circuits and Systems, 2003 IEEE 46th Midwest Symposium on*, volume 1, pages 333–336. IEEE, 2003.

[27] Linda Shapiro and George C Stockman. Computer vision. 2001. *ed: Prentice Hall*, 2001.

[28] I Sobel and G Feldman. A 3x3 isotropic gradient operator for image processing (1968). *a talk at the Stanford Artificial Intelligence Project*.

[29] Biometrics Ideal Test. Casia iris image database for testing version 1.0, July 2015.

[30] Yong Zhu, Tieniu Tan, and Yunhong Wang. Biometric personal identification based on iris patterns. In *icpr*, page 2801. IEEE, 2000.

APPENDIX

Proof. By definition of the 2D discrete convolution:

$$y[m,n] = x[m,n]*g[m,n] = \sum_{i=-\infty}^{\infty}\sum_{j=-\infty}^{\infty} x[i,j]\cdot g[m-i,n-j]$$

By commutativity of the convolution

$$y[m,n] = g[m,n]*x[m,n] = \sum_{i=-\infty}^{\infty}\sum_{j=-\infty}^{\infty} g[i,j]\cdot x[m-i,n-j]$$

and the seperability property of $g[i,j] = g'[i]\cdot g''[j]$ we acquire:

$$\begin{aligned} y[m,n] &= \sum_{i=-\infty}^{\infty}\sum_{j=-\infty}^{\infty} g'[i]\cdot g''[j]\cdot x[m-i,n-j] = \\ &= \sum_{i=-\infty}^{\infty} g'[i]\cdot \sum_{j=-\infty}^{\infty} g''[j]\cdot x[m-i,n-j] = \\ &= \sum_{i=-\infty}^{\infty} g'[i]\cdot \left(\sum_{j=-\infty}^{\infty} g''[j]\cdot x[m-i,n-j]\right) \end{aligned}$$

By plugging in the definition of a 1D discrete convolution $y[n] = x[n]*g[n] = \sum_{i=-\infty}^{\infty} x[n]*g[n-i]$, we get:

$$\begin{aligned} y[m,n] &= (g'[m]\cdot g''[n])*x[m,n] = g'[m]*(g''[n]*x[m,n]) \\ &= g''[n]*(g'[m]*x[m,n]) \end{aligned}$$

□

Proof. We want to show the seperabilty of the Gaussian filter kernel. Seperability would mean that

$$G(x,y) = G(x)\cdot G(y)$$

The 2D $G(x,y)$ and 1D version $G(x)$ of the Gaussian filter kernel are defined as follows:

$$G(x,y) = \frac{1}{2\pi\sigma^2}e^{\frac{x^2+y^2}{2\sigma^2}}$$

$$G(x) = \frac{1}{\sqrt{2\pi}\sigma}e^{\frac{x^2}{2\sigma^2}}$$

Obviously by plugging in the definitions we obtain

$$\begin{aligned} G(x)\cdot G(y) &= \frac{1}{\sqrt{2\pi}\sigma}e^{\frac{x^2}{2\sigma^2}}\cdot\frac{1}{\sqrt{2\pi}\sigma}e^{\frac{y^2}{2\sigma^2}} = \\ &= \frac{1}{2\pi\sigma^2}e^{\frac{x^2+y^2}{2\sigma^2}} \end{aligned}$$

□

Proof. We want to show the seperability of the Sobel filter.

$$G_x = \begin{pmatrix} -1 & 0 & 1 \end{pmatrix}\cdot\begin{pmatrix} 1 \\ 2 \\ 1 \end{pmatrix} = \begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix}$$

$$G_y = \begin{pmatrix} 1 & 2 & 1 \end{pmatrix}\cdot\begin{pmatrix} -1 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{pmatrix}$$

□