Enhancing the Performance of GPU for Face Detection

Hossam M. Faheem, Ph.D
Head of Computer Systems Department, Faculty of Computer and Information Sciences, Ain Shams University, Cairo, Egypt

Assoc. Prof. S. Ghoniemy
Computer Systems Department, Faculty of Computer and Information Sciences, Ain Shams University, Cairo, Egypt

T. A. Yara M. Abdelaal
Computer Systems Department Faculty of Computer and Information Sciences, Ain Shams University, Cairo, Egypt

ABSTRACT
Computer Vision algorithms are considered computationally intensive problems. Face detection is one of the most complex objects to detect due to its variations. The objective is to enhance the face detection time (compared with other approaches) to reach a real-time application that will be later on used in augmented reality applications such as telepresence. The experiments with NVIDIA GTX 560 show that detecting the faces in an image of size [640x480] can process up to 33 frames per second, also this paper shows how the researcher's approach can be generalized to support larger image sizes. This in turn reflects back the achieved speed that exceeds FPGA.

Keywords
GPU computing, Viola-Jones face detection

1. INTRODUCTION
Face detection and tracking is one of the fast-growing subjects in computer vision world, and one of the most processing intensive algorithms due to the variations in the face features. The human face is more complex than any other object since the human face varies in forms and expressions [1].

Many different algorithms exist to perform face detection, each having its own pros and cons. Most of these algorithms are computationally expensive [2].

HAAR like features are image features used for object detection and recognition. It is one of the fastest and most accurate object detection algorithms. The term "HAAR-like features" origins from the calculation of Viola and Jones [3] which works with HAAR wavelet transform method, a window of the target size is moved over the input image, and for each subsection of the image the HAAR-like feature is calculated. This difference is then compared to a learned threshold that separates non-objects from objects. The discrimination between objects and non-objects is achieved by the learning phase. A set of both positive and negative images is fed to the HAAR trainer, from which the classifier is extracted. After the training phase, the classifier could be applied to a region of interest on the captured frame from the video. The classifier outputs "1" if the region contains a face (i.e. object) and "0" otherwise. To search for faces in the whole frame, sliding window technique is applied, whereas a window (of fixed size as that used during training) is moved across the image [4].

Regions in a rectangular form are considered at a specific window location, HAAR like features sums up the intensities of these pixels in each region and calculates the difference between these sums. This difference is then used to categorize subsections of an image. There are several types of features; a two-rectangle feature, which is the difference between the sum of pixels within two rectangular regions, a three-rectangle feature computes the sum within two outside rectangles subtracted from the sum in a center rectangle. Finally a four-rectangle feature computes the difference between diagonal pairs of rectangles Figure 1.

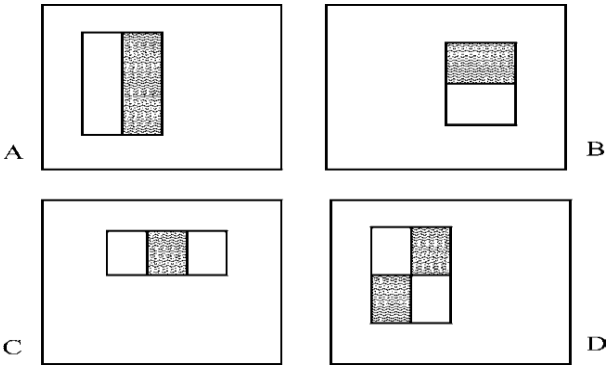

Figure 1: The sum of pixels in the white is subtracted from those in the grey area. Two-rectangle features is represented in (A) and (B), three-rectangle is represented in (C), and four-rectangle is represented in (D)

Viola and Jones constructed a cascade of classifiers that achieves increased detection performance while radically reducing computation time. This is achieved by boosted classifiers which rejects negative windows in early stages, which results in reduction of computation.[3]. Figure 2 represents the cascaded feature structure.

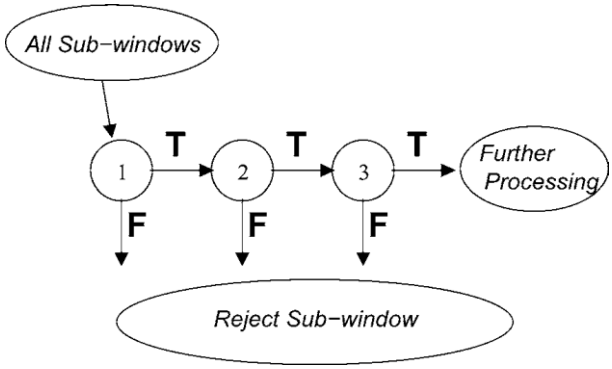

Figure 2: The Cascaded Features

2. PREVIOUS WORK
Previous studies were directed to optimizing the integral image calculation on the General Purpose Graphics Processing Unit (GPU) by dividing the integral image into sub-regions and a local Integral Image is calculated (LII), the Global Integral Image (GII) is calculated by merging the LIIs. The input image is divided into sub-regions such that the data dependency is limited [5].

Endang Setyati proposed improving face detection; the main interest was not to detect human-alike faces but to detect human faces. They did so by processing a two dimensional (2D) face data into a three dimensional (3D) space aiming to improve the weakness of 2D detection. 3D face detection was able to improve the benefits of face detection by increasing the precision and accuracy of the process. The researcher used Active Shape Model algorithm due to its low error rate but this was at the cost of the performance and processing power consumption, along with a limitation of the image environment as it has significant impact on the output [6].

Agrawal proposed a design, which will have a great impact on video surveillance systems, whereas a segmentation is used after background subtraction and background estimation, this was used to reduce noise and locate a moving target in a frame. For multiple moving objects detection in poor lighting conditions, wavelet-based contrast change detector was integrated with locally adaptive thresholding scheme for initial frames. For later frames latest Change detector mechanism was used. This has significant results for surveillance systems, where the camera is mounted in a stable location, but for handheld devices this will not be accurate for our study [7].

Daniel Hefenbrock and Jason Oberg proposed a design by executing a sequence of kernels, functions that run under Single Instruction, Multiple Threads (SIMT) model. They reached a performance of 3.8fps on a single GPU but on the cost of GPU utilization [8].

Bilgic proposed using Histograms of Oriented Gradients (HoG) features on a GPU, by using Cascade-of-rejectors algorithm to make fast rejections for blocks that do not contain a person in early stages. Viola & Jones using HAAR-like features first introduced this, although it has higher error rate, but it has better performance for a real-time application such as Augmented Reality [9].

Edmund Weng lately proposed an improved version of the SURF algorithm, which was used for features extraction from live mobile camera image and recognition of real world objects. Homography techniques were used to determine the pose matrix from extracted features. Different characteristics of virtual objects such as rotation, scaling and translation were controlled by calculated pose matrix [10].

3. PROPOSED IMPLEMENTATION FOR GPU UTILIZATION

The proposed implementation uses a 2D grid although it is based on a 1D grid. The conversion is carried out by each kernel. The kernel's job is to scan for face features on a single window, keeping in mind that the algorithm yields different window sizes (i.e. scales) which requires pre-scaled features and window information such as the x and y position of the window and the window size. The features set consists of different stages cascaded together. Each window will pass through the stages till it passes through the last one. This set of features is done sequentially, meaning that each kernel will not need to scale the features nor scale the image itself, in other words the kernel's job is to evaluate features and stages for its specific window. To avoid memory latency and maximum memory bandwidth, the data is sorted according to the scale following by the x and y position of each window [11].

The number of kernels is equal to the number of search windows which varies according to the image size used. Table 1 shows the number of search windows versus image size in pixels.

Table 1: Image Size (pixels) Number of Search Windows

Size in Pixels	Number of Windows
640*480	177527
900*450	240678
1024*768	487952

Figure 3 shows a graph representing the data presented in Table 1, it is noted that the number of search windows increases in a cubic manner. Which helps in evaluating the proposed design.

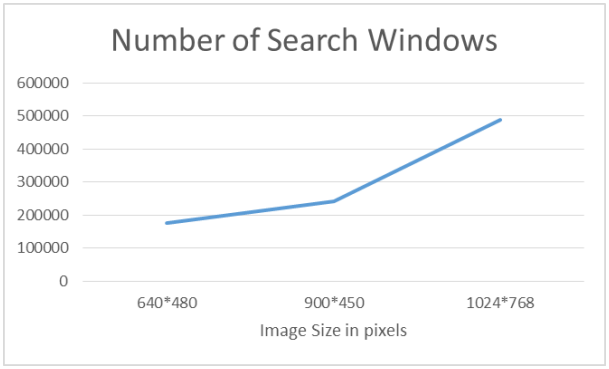

Figure 3: Number of Search Windows VS Image Size (pixels)

Since GPUs are based on PCI Express (PCIe), additional overhead costs are resulted from host-to-GPU data transfers and vice versa, which may cause application bottleneck at data transfer [12]. The researchers believed that this process consumes time and processing power, which derived them to a new approach whereas to minimize the size of data allocated and shared from the host to each working item.

For proper memory utilization and decreasing non-necessary data allocation, all criteria and data passed needs to be studied to determine the dependency and data size, below is a list of all passed data needed for each working item to evaluate the features and stages (on its specific window):

1. Start X position of the window
2. Start Y position of the window
3. Window Step
4. Original Image
5. Integral Image
6. Feature Set (Scaled feature)
7. End X Position
8. End Y Position

Some of the above parameters are indispensable such as the original image, the integral image, and the scaled feature. The rest of the parameters could be substituted by equations. The approach was to find an equation that best fits the data of the below mentioned parameters with respect to window ID (i.e. the Kernel ID which is incremental since we use 1D grid). These parameters are:

1. Scale
2. Step
3. Start X Position

4. Start Y Position
5. End X Position
6. End Y Position
7. Modified Window ID
8. Number of windows per Row
9. Number of Rows
10. Total Number of Windows.

3.1 Scale Estimated Model

The number of windows vary according to the scale value. In order to find an equation that satisfies the Scale values, scatter plot must be applied first, that helps to identify the model or set of models that would provide the least error (highest R Squared value), Figure 4 represents a scatter plot for the scale against the window identifier; which is a sequence of numbers starting from 0 up to 177528 (i.e. the number of all possible windows for all scales).

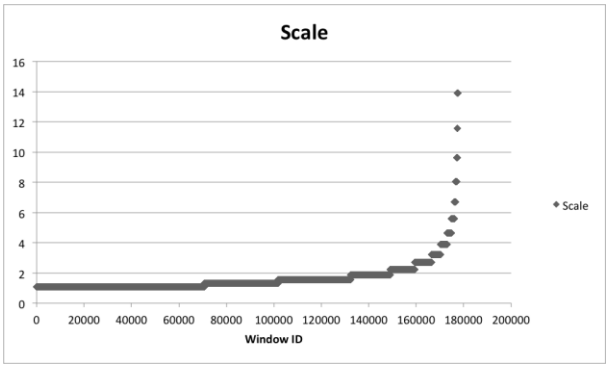

Figure 4: Scatter Plot for the relation between Scale and Window ID

It was found that Polyratio (3, 3) model yields least error values and highest R square of 0.996 equation 1 represents the model produced and Figure 5 represents the model curve fit.

$$Scale = \left[\frac{1.080989 - (9.24E-06)*ID + (1.79E-11)*ID^2}{1 - (1.05E-05)*ID + (2.7429E-11)*ID^2}\right] \quad (1)$$

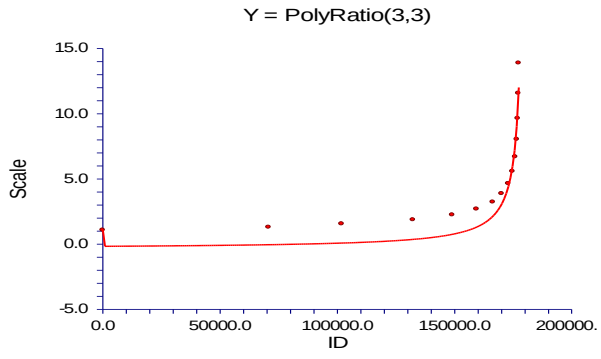

Figure 5: Scale value estimation using PolyRatio (3, 3) Model

Table 2 represents the Scale original values against the ones calculated using the model presented in Equation (1)

Table 2: Sale calculated values using Ratio Polynomials Fit

Row	Scale	Calculated Value
1	1.08176	1.080989
2	1.29811	1.310705
3	1.55773	1.515147
4	1.86928	1.894945
5	2.24313	2.301888
6	2.69176	2.740600
7	3.23011	3.242605
8	3.87613	3.697320
9	4.65136	4.303873
10	5.58163	5.078635
11	6.69796	6.221029
12	8.03755	7.808249
13	9.64506	9.686261
14	11.57410	11.81005
15	13.88890	13.92330

3.2 Step Value Estimated Model

The first step is to scatter plot the step values, the researchers found that it is best to relate the Step value to the Scale, since the step of the window is merely based upon the size of the window. Figure 6 represents the scatter plot of Step against Scale.

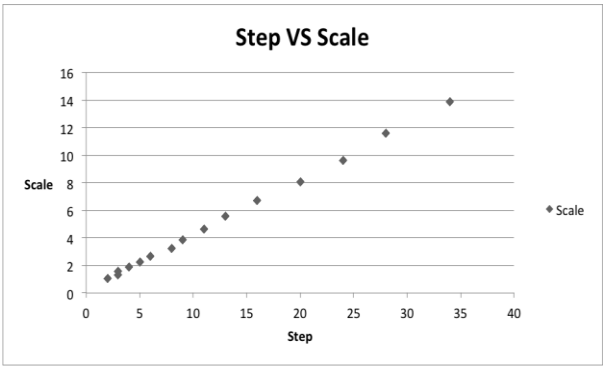

Figure 6: Scatter Plot for the relation between Step and Scale

From the Scatter Plot presented above, it is noted that the relation is a simple linear one. Hence Linear Model was tested and yielded a rather high R^2 value of 0.999562. Figure 7 represents the estimated model graph, while Equation (2) represents the model equation.

$$Step = [-0.66059 + 2.47558 * Scale] \quad (2)$$

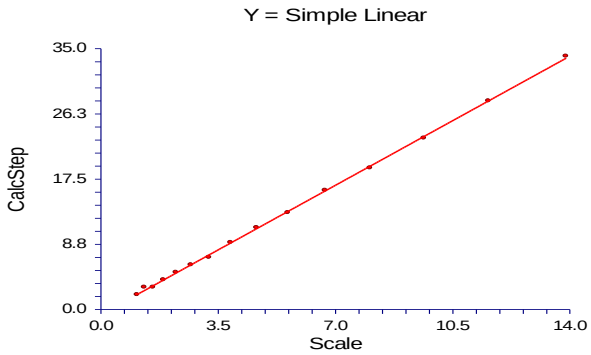

Figure 7: Step Value estimation using Simple Linear Model

3.3 Modified Window ID

This variable was introduced to be able to evaluate both the X Start position and Y Start position of the window in the original image. After trying all 39 models the researchers found that no model will satisfy the whole data set, so the data set was divided into smaller data sets according to the scale, meaning that each scale represents a sub data set. By referring to Figure 5, it is noted that the scale remains constant for a period of time, which represents the number of windows for this scale. Number of windows per scale is calculated by equation (3)

$$WPS = WPR * NumOfRows \quad (3)$$

Where WPR represents the number of windows per each row represented in equation (4)

$$WPR = \left\lfloor \frac{ImageWidth - (windowWidth * Scale)}{Step} \right\rfloor \quad (4)$$

And NumOfRows represents the number of rows represented in equation (5)

$$NumOfRows = \left\lfloor \frac{ImageHeight - (windowHeight * Scale)}{Step} \right\rfloor \quad (5)$$

Where Image Width represents the sample image width (i.e. 640 pixels),

windowWidth represents the size of the window that the feature file was based on (i.e. 20),

Image Height represents the sample image height (i.e. 480 pixels).

The modified window id would be incremental for every sub-data set.

The proposed design is presented by calculating parameter y which indicates whether the current kernel is within the Data Set range or not, y is represented in equation (6)

$$y = \left\lfloor \frac{WindowID - RangeStart}{Abs(WindowID - RangeStart)} \right\rfloor \quad (6)$$

Where RangeStart values represents the data set window ID range, these ranges are represented in Table 3

Table 3: Data Set Range Values

Data Set Num.	Range Value
1	0
2	70761
3	132371
4	149132
5	159485
6	166443
7	170187
8	173022
9	174822
10	1746011
11	176715
12	177099
13	177327
14	177462

By applying equation (6) using the values represented in Table 3 the possible values for y are represented in (7):

$$y = \begin{cases} -1 \\ 0 \\ 1 \end{cases} \quad (7)$$

Y is equal to -1 in case the window ID is smaller than the range;

Y is equal to 0 in case the window ID is exactly equal to the range value;

Finally Y is equal to 1 in case the window ID is larger than the range (out of range).
The Y calculated in equation (6) is used to calculate z value which is represented in equation (8)

$$z = \left\lceil \frac{y+1}{2} \right\rceil \quad (8)$$

By applying the values calculated from y to z, the possible values for z is as represented in (9)

$$z = f(y) = \begin{cases} 0 & y < 0 \\ 1 & y \geq 0 \end{cases} \quad (9)$$

By using the data calculated from (9) the range is easily identified whether it matches or not, the modified window ID could be accumulated as in equation (10)

$$ModifiedWindowID = (windowID - range) * z \quad (10)$$

3.4 X and Y Positions Model

The X and Y positions represents the position of the window. When the data is studied in a sequential manner, it is noted that each window is separated from the previous one by the step size, for example if for step size 2, the first window position will be (0,0) the following one will be (2,0) (i.e. sliding window technique with the step size). Once the X reaches the end of the image (according to the number of windows per row which ensures that the window lies within the image boundaries), the X starts from 0 once more but with an increment of the Y position equal to the step size i.e. (0,2).
To calculate both X and Y positions the researchers found it necessary to use the modified window ID, hence the X and Y positions will also be in a sub-data sets manner, but in a single equation form represented in equation (11):

$$X = MOD[MOD(ModWindowID, WPR) * Step, iWidth] \quad (11)$$

Where iWidth is represented below:

$$iWidth = Image\ Width - (WindowWidth * Scale) \quad (12)$$

The equation for Y position calculations is also presented in a single formula represented in equation (13):

$$Y = Floor\left[\frac{ModWindowID}{WPR * Step} \right] \quad (13)$$

4. EXPERIMENTAL RESULTS

Constructing a fair comparison required a set of parameters that needed to be defined beforehand, to ensure the level of accuracy of the simulation process. Since the objective of the simulation was comparison, the focus was on the congruency of the simulation parameters and the environment. The parameters to ensure a fair comparison are as follows:

1. Simulation parameters, including image resolution, hardware configuration have to be equal.
2. Image Data Set used must be identical, to stabilize amount of noise and lighting conditions that may affect the detection.
3. Feature file used to detect a face must be identical.

By running the previously described approach a Dell Latitude E5530 with a 2.5 GHz i5 processor, 8 GB RAM, and an Intel HD Graphics 4000. Windows 7 64-bit a rate of 9.71 FPS was reached, these results are for a [640*480] Image size.

Comparison to other implementations is not straightforward. Some papers use rather large step size which will dramatically have an effect on both the accuracy and the amount of computation since the number of windows to be searched will increase accordingly. On an additional note, the GPU card used to test the implementation on affects greatly the end As for GPU the proposed design enhances both memory allocation and time to process a frame reaching real time performance. The figure below represents performance change on the different machines used from Table 4 in addition to a Dell Latitude E5530 with a 2.5 GHz i5 processor, 8 GB RAM, and an Intel HD Graphics 4000, Windows 7 64-bit for CPU results in terms of frames processed per second. It is noted that when the GPU computing power and specifications increases the performance is enhanced along. Figure 8 shows the results of testing the implementation on the same image containing 5 faces. As the number of faces increases the processing time increases accordingly, this is because for each search window, in case a face is found the window processes more information by processing all 22 stages for face detection which accordingly increases the processing time. The results below are calculated by negating the kernel compilation time which only runs once on application startup.

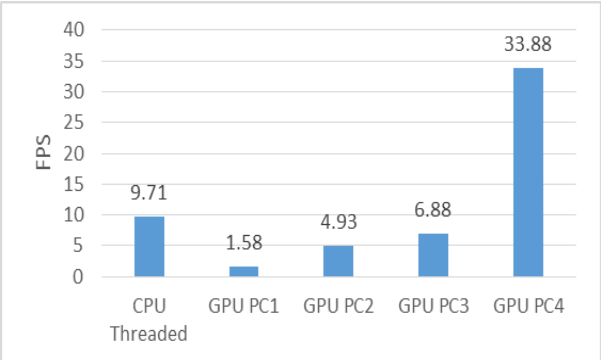

Figure 8: Performance Difference by running proposed implementation on different hardware

Figure 9 represents performance change on the same GPU (PC2) for different images, each varying in the number of faces, starting from an image containing a single face up to 5 faces on GPU GeForce GT 610 (PC2).

results of the processing time, the more a GPU is utilized, the less processing time is achieved on larger and more powerful GPUs.

The experiments were conducted on different GPUs with the same image set. Each GPU specifications are listed in Table 4.

Table 4: Different GPUs Used Specifications

GPU Model	Number of SM	Processor Clock (MHz)	Memory Bus	Memory Clock (MHz)
GeForce 310M (PC1)	16	1530	64-bit	800
GeForce GT 610 (PC2)	48	1620	64-bit	1000
GeForce GT 240 (PC3)	96	1340	128-bit	1000
GeForce GTX 560 (PC4)	336	1620	256-bit	2000

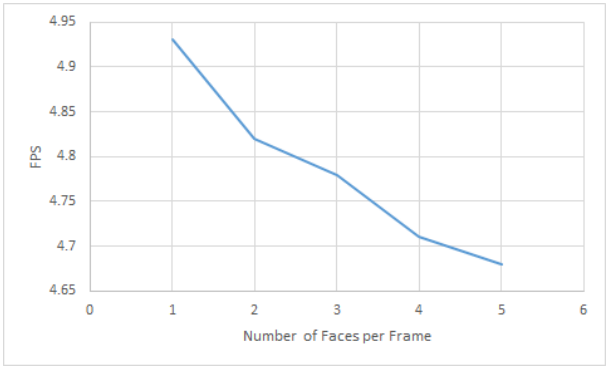

Figure 9: Number of faces per frame against the frame rate required to process each frame.

5. CONCLUSION

The proposed implementation shows that FPGA can be achieved with GPU solutions. The paper has discussed the parallel parts in Viola-Jones face detection algorithm and how to utilize the GPU efficiently. The paper has proposed a new implementation that removes all duplicated effort in scaling the features to the target window size and removes as much parameters passed to the kernel as possible by replacing them with mathematical equations that describes the data required. This minimizes host-to-GPU data transfers and at the same time it provides exact results match. This approach achieves a great performance enhancement compared to other GPU and FPGA implementations.

6. REFERENCES

[1] N. Muller, L. Magaia, and B. M. Herbst, "Singular Value Decomposition, Eigenfaces, and 3D Reconstructions," SIAM Review, vol. 46, no. 3. pp. 518–545, 2004.

[2] P. I. Wilson and J. Fernandez, "Facial feature detection using Haar classifiers," J. Comput. Sci. Coll., vol. 21, pp. 127–133, 2006.

[3] P. Viola and M. Jones, "Rapid object detection using a boosted cascade of simple features," Proc. 2001 IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognition. CVPR 2001, vol. 1, 2001.

[4] O. Community, "OpenCV Reference Manual," October, pp. 1–1104, 2010.

[5] Y.-T. Wu, Y.-T. Wu, C.-Y. Cho, S.-Y. Tseng, C.-N. Liu, and C.-T. King, "Parallel Integral Image Generation Algorithm on Multi-core System," 2011 IEEE Ninth Int. Symp. Parallel Distrib. Process. With Appl., pp. 31–35, 2011.

[6] E. Setyati, D. Alexandre, and D. Widjaja, "Face Tracking Implementation with Pose Estimation Algorithm in Augmented Reality Technology," Procedia - Soc. Behav. Sci., vol. 57, pp. 215–222, Oct. 2012.

[7] D. Agrawal and N. Meena, "Performance Comparison of Moving Object Detection Techniques in Video Surveillance System," Int. J. Eng. …, pp. 240–242, 2013.

[8] D. Hefenbrock, J. Oberg, N. Thanh, R. Kastner, and S. Baden, "Accelerating Viola-Jones Face Detection to FPGA-Level Using GPUs.," FCCM, pp. 11–18, 2010.

[9] B. Bilgic, B. K. P. Horn, and I. Masaki, "Fast human detection with cascaded ensembles on the GPU," Intell. Veh. Symp. (IV), 2010 IEEE, 2010.

[10] E. Weng, R. Khan, S. Adruce, and O. Bee, "Objects Tracking from Natural Features in Mobile Augmented Reality," Procedia-Social …, vol. 97, pp. 753–760, Nov. 2013.

[11] M. Fayez, "G PU -A CCELERATED F ACE D ETECTION A LGORITHM," vol. 4, no. 2, pp. 47–55, 2014.

[12] M. Daga, A. M. Aji, and W. Feng, "On the Efficacy of a Fused CPU+GPU Processor (or APU) for Parallel Computing," 2011 Symp. Appl. Accel. High-Performance Compute. pp. 141–149, 2011.