International Journal of Computer Applications (0975 - 8887)
Volume 176 - No.36, July 2020

Evaluating Embedded GPUs Performance via Computer Vision Applications

Paulo S. S. de Souza
Pontifical Catholic University of Rio Grande do Sul
Porto Alegre - Brazil

Arthur F. Lorenzon
Federal University of Pampa
Alegrete - Brazil

Marcelo C. Luizelli
Federal University of Pampa
Alegrete - Brazil

Fabio D. Rossi
Federal Institute Farroupilha
Alegrete - Brazil

ABSTRACT

Computer vision applications usually present significant demand for computing resources, which limit its usage on embedded systems, since such devices typically have limited processing capacity. In this sense, hybrid embedded architectures are becoming more popular by offering higher levels of parallelism through Graphics Processing Units (GPUs). Despite the similarities with general-purpose architectures that already exploit the benefits of GPUs, this new kind of embedded devices presents some architectural singularities, such as differences in memory access bandwidth. In this paper, we present an evaluation, considering how much these differences affect GPUs' gains in the context of embedded systems. The results show that, despite the architectural limitations, using such devices can lead to a speed-up of 8 times compared to traditional embedded systems processing data only on CPUs.

General Terms

Algorithm, Processor.

Keywords

Embedded Systems; GPGPU; Computer Vision; Performance Analysis.

1. INTRODUCTION

Recently, computer vision applications are becoming more popular by providing solutions to the most diverse scenarios like healthcare, industry development, and urban mobility. The use-cases are many: localizing and tracking vehicles in real-time, monitoring multiple variables in situations requiring strict control like nuclear power plants, or even helping students understand complex concepts school [14][7].

However, there are still problems with exploiting the potential of computer vision applications. They are usually deployed in embedded systems, which are small devices with limited capacity (e.g., sensors and cameras), typically used to gather data from the environment [8].

Embedded systems commonly present architectural limitations due to concerns with temperature and battery consumption. They provide limited processing power, which can affect essential metrics to customers like Quality of Service (QoS) and Quality of Experience (QoE). In this sense, heterogeneous embedded architectures were introduced with the proposal of increasing the performance of some applications by exploiting the massive amount of parallel cores of GPUs [11][17].

However, increasing the performance of image processing algorithms in heterogeneous architectures is not a trivial task since parallelism's effectiveness is highly linked to the application behavior. In this sense, some frameworks to support such features were proposed [4].

In this context, OpenCV (Open Source Computer Vision Library) is an open-source computer vision and machine learning software library that allows the development of parallelism applications. Moreover, this library is available to the most commonly used operating systems nowadays, like Windows, macOS, and Linux [15]. Considering that several embedded systems are based on Linux, OpenCV arises as a feasible alternative to implement parallel computer vision applications in these systems. Also, OpenCV was designed for computational efficiency and with a strong focus on real-time applications. There are several OpenCV algorithms available with the most diverse goals, such as detection of objects, faces, cars, or even specific shapes such as lines, circles, and ellipses. As a consequence, usually, each application has its particular processing demands.

Thus, the utilization of embedded devices with heterogeneous architectures, i.e., that provide multiple processing components like CPUs and GPUs could be a feasible alternative to improve the performance of computer vision applications without sacrificing relevant metrics like power consumption and temperature. The reason for such gains is that modern GPUs support a massive number of logical units, so tasks designed to run in GPUs can take advantage of higher levels of parallelization.

Due to the minimalist architecture of embedded devices, embedded GPUs usually do not have dedicated memory, so the operating system makes a pointer to the data in the main memory, which can cause performance loss. While architectures with high processing power via GPUs can achieve performance greater than 240x compared to traditional CPUs [9], GPUs on embedded devices have not yet been thoroughly evaluated.

Consequently, not every application whose performance is known as being boosted by running on general-purpose GPUs would present a similar behavior by running on embedded GPUs, due to the architectural differences that must be considered. Therefore, **this paper presents a performance evaluation of embedded GPUs during the processing of computer vision applications in order to verify the feasibility of heterogeneous embedded architectures to execute computer-vision applications.**
The remaining of this paper is organized as follows: Section 2 presents a theoretical background in computer vision. Section 3 shows some previous researches that focused on evaluating the performance of computer vision applications in embedded devices. Section 4 presents details of how the evaluation was conducted. Section 5 presents a discussion about the results. Section 6 is reserved for the final remarks and directions for future research.

2. BACKGROUND

Transformation of data from a video or camera into either information or a new representation is called Computer Vision [16]. Several changes are performed to accomplish a specific goal. The input data may include contextual information about cars, people, or the most diverse objects. Computer vision emerged as a solution for several computing problems, and it has been applied in many environments.
Computer vision applications deal with considerable amounts of data and, by consequence, require high processing capacity. Therefore, many processor units are needed, but impossible to embed on in a small quantity of hardware design space. GPUs play a significant role in this context since a considerable part of computer vision involves image processing; one of the areas where GPUs are excellent. Given this opportunity, GPU designer engineers employed strategies to make general-purpose GPUs (GPGPU), GPUs designed to perform non-specialized calculations that were typically conducted by the CPU.
Despite the high computational throughput achieved through GPUs, from a programmability standpoint, GPU programming is still complicated as it requires the use of specific mechanisms that are not always easy to handle [12]. In this sense, CUDA was proposed in order to facilitate GPU programming with a general interface. Using this execution model, the GPU is a co-processor that performs several threads in parallel, such as the single instruction multiple data (SIMD) model of execution. A data-parallel computation process can be offloaded to the GPU to be executed, and the collection of threads is divided into a grid of thread blocks [1].
Advances in digital circuits manufacturing have enabled the development of smaller GPUs at lower costs, which in turn allowed the introduction of such components into the embedded systems landscape. As a consequence, GPUs arose out of the need to offload massive processing from the CPU and handle mathematical calculations required for tasks like 3D rendering [5]. Therefore, this large processing capacity in a small hardware design space drives the use of edge devices to meet the demands of Big Data.
With the popularization of the Internet of Things, these technologies are becoming even more famous by delivering additional capabilities to tasks like camera monitoring. However, as embedded systems usually present architectural limitations, embedded computer vision applications must be designed to provide high performance without sacrificing the battery of the devices [3]. To meet these needs, several heterogeneous architectures were introduced to cope with the computational requirements, such as NVIDIA Jetson TX1 and NVIDIA Jetson TX2, which provide specialized processing capabilities through CPUs and GPUs.

Unlike server architectures where GPUs have dedicated memories, and the data is transferred between main memory and GPU memory, in embedded systems with GPUs, the data to be processed is received by the CPU and stored in the system's memory. In this kind of architecture, shown in Figure 1, usually, there is no dedicated memory to the GPUs due to embedded design issues. Therefore, the operating system allows the GPU to access data in the main memory through programming objects like pointers that stores the memory address of another value located in computer memory.
Thus, as GPUs must fetch data from main memory, the higher the amount of data, the greater the overhead on the bus that will be used in the communication between GPUs and the host system, which will lead to a loss in performance. In this sense, depending on the application's memory access pattern, offloading data to the GPU could lead to performance losses despite the greater parallelism capabilities.
Therefore, this paper aims to analyze the feasibility of using embedded GPUs with shared-memory to execute computer vision applications, besides verifying which characteristics of applications should be taken into account when considering using embedded systems with hybrid CPU-GPU architectures.

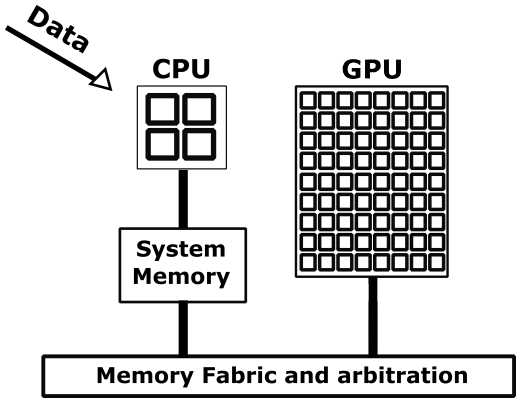

Fig. 1. Data is acquired by CPUs and stored in system's memory. The operating system provides a pointer to the data in main memory, allowing it to be accessed by the GPUs.

3. RELATED WORK

Due to this topic's relevance, improving the performance of computer vision applications on embedded systems has been studied by several researchers. For example, Dekkiche et al. [4] proposed an approach to target both operating-system and kernel-level optimizations in embedded systems real-time computer-vision applications such as ADAS. The authors' approach consists of combining a Domain Specific Embedded Language (DSEL) C++ library and the OpenVX framework.
Hurtado et al. [7] presented a system for tracking vehicles in real time. The authors used computer-vision algorithms in an embedded platform for an ADAS, which uses a monocular color camera. The video is processed using support vector machines (SVM) and convolutional neural networks (CNN). The authors used as proof-of-concept platform a Jetson TK1 and several libraries like CUDA, OpenCV, and CudNN to perform the application optimization.

Table 1. Technical specifications of the NVIDIA Jetson TX2 Developer Kit.

Components	Specifications
Processors	HMP Dual Denver 2/2MB L2 + Quad ARM A57/2MB L2
GPU	NVIDIA PASCAL (256 CUDA cores)
Memory	8GB 128bit LPDDR4 59.7GB/s
Networking	1 Gigabit Ethernet, 802.11ac WLAN, Bluetooth
Operating System	Linux Ubuntu 16.04.3 LTS (kernel 4.4.15-tegra)

Honegger et al. [6] emphasize that the computational power of smartphones has increased substantially in recent years. However, they are still incapable of performing intensive computer-vision tasks in real-time, at high frame rates, and low latency. In this sense, the authors present a combination of FPGA (Field Programmable Gate Array) and mobile CPU to reduce the computational and latency limitations of mobile CPUs. The FPGA actuates as an additional layer between the image sensor and the CPU. The system is capable of improving the performance of computer-vision algorithms to real-time performance.

Che et al. [2] developed and analyzed an application for monitoring stem diameter information without interrupting the natural growth of the crops to provide water-saving irrigation. The authors designed an embedded software for image acquisition, edge detection, and data storage. Such a proposal enables one to automatically measure and record the plant stem's diameter and perform comparisons to show the effect of stem measurement under different illuminating methods.

Meng et al. [13] proposed a human action recognition system suitable for embedded computer-vision applications and can be used in security systems, and intelligent environments. The authors describe that the system was based on a linear support vector machine (SVM), and combines MHI (Motion History Image) and HMHH (Hierarchical Motion History Histogram) to extract a low dimension feature vector to be used in the SVM classifier. In order to evaluate their proposal, the authors performed several experiments through a challenging human action recognition database with six types of social actions played several times by 25 subjects in four different scenarios. The results of their proposal were compared with other methods, and the results showed significant improvements.

Pauly et al. [14] presented an optical measurement device which focuses on using computer vision applications to provide a less costly and more efficient alternative to object measurement in industrial environments. The device consists of an image sensor, several laser-line projectors, and an ARM microcontroller.

Embedded devices are being widely used for computer vision tasks, which can take advantage of specialized processing units like GPUs. However, due to design issues, most embedded GPUs do not have dedicated memory, so the GPU has to access the main memory through pointers created by the operating system.

In this context, to the best of our knowledge, no other work has brought a performance evaluation of heterogeneous embedded systems during the execution of computer-vision applications in order to analyze the optimal processing strategy in such kind of architecture, either keeping data only being processed in the CPU or combining the processing between CPU and GPU.

4. EVALUATION

We conducted experiments to analyze the impact of exploiting the parallelism provided by embedded GPUs during the execution of four computer vision algorithms:

—**Generalized Hough**: used to identify defined types of shapes like lines, circles, and ellipses, or detect arbitrary forms. It uses edge information to establish a mapping of objects from the orientation of an edge point to a reference point of the shape.

—**Hough Lines**: used in image analysis to find instances of objects within a particular class of shapes by identifying its edge lines, which is relevant in tasks such as object recognition.

—**Histogram of Oriented Gradients (HOG)**: identifies objects in an image by analyzing the distribution of intensity gradients. This algorithm is used in many applications, like hand gesture recognition, traffic sign recognition, and human recognition.

—**Super Resolution**: enhances the resolution of images by exploring a sharpness index. This algorithm is employed to combine a sequence of low-resolution frames from a scene and produce a higher resolution picture or series.

In this study, we took into account two well-known parallel algorithm metrics [10]: i) *Speedup*, which is the ratio of the serial runtime of the best sequential algorithm for solving a problem to the time taken by the parallel algorithm to solve the same problem on p processors, and ii) *Efficiency*, which consists in the ratio of speed-up to the number of processors.

Moreover, we adopted the NVIDIA embedded System-on-a-Chip (SoC) development kit Jetson TX2[1]. This board is suitable for high-performance computing applications such as robots, drones, smart cameras, and portable medical devices.

Also, this embedded device comprises a GPU with 256 CUDA cores making it ideal for our experiment. Table **??** presents more details about this embedded platform. In order to perform the experiments, we used implementations of the chosen algorithms in OpenCV version 3.3.0. The results presented next are the mean of 10 executions of each algorithm with a standard deviation of less than 2%.

The results revealed that GPU's parallelism capabilities led to performance increases in the chosen algorithms. Although the speed-up ratio was favorable in all the executed tests, as we can notice in Figure 2, the analysis of the parallelism efficiency showed that the use of the computing resources was not ideal, that is, despite the performance speed-up, some of the GPU processors were kept inactive during some parts of the algorithms' execution.

5. DISCUSSION

The most significant speed-up was achieved by running the Houghlines algorithm in the GPU. This result can be understood by analyzing the structure of such an algorithm, which implementation allows the exploitation of the massive parallelism provided by the GPU.

More specifically, this algorithm manipulates information related to the edge points of the tested image into matrices. At this point, the GPU approach shows a considerable advantage over the CPU by processing higher amounts of arrays data at the same time into their cores. Hence, Houghlines executed eight times faster in the GPU.

The HOG feature descriptor algorithm also presented positive results, where offloading the processing method to the GPU led to a four-time speed-up. This performance gain is explained through the analysis of the algorithm behavior. The Hog descriptor calculates the gradient orientations and magnitude of each pixel of an image.

[1] https://developer.nvidia.com/embedded/buy/jetson-tx2

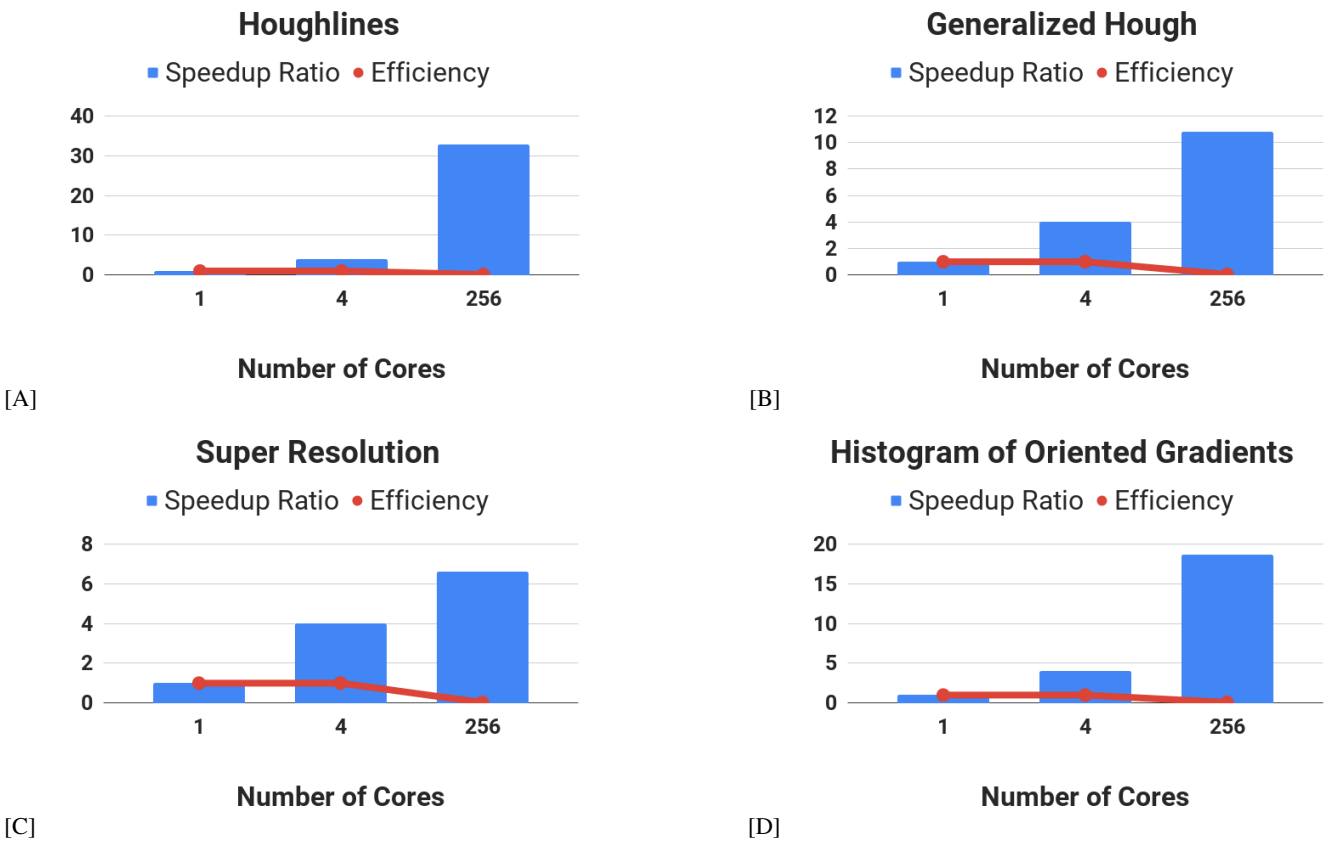

Fig. 2. Performance comparison between CPU and GPU during the execution of different computer vision algorithms in a Jetson TX2 device.

Next, it divides the image into groups of pixels (which are called cells). Then, it creates histograms of gradient orientations of each of these cells and groups them into blocks that will be analyzed by a descriptor. In other words, the Hog algorithm distributes several threads among the GPU cores to process multiple pixels, cells, and blocks simultaneously. This behavior allows the HOG feature descriptor to take advantage of GPUs' parallelism since each of these processing steps can be divided into small pieces and performed simultaneously.

The results also showed that offloading the Generalized Hough to the GPU could generate performance gains since it manipulates information related to the edge points of the tested image into matrices. At this point, the GPU presents a considerable advantage over the CPU by processing higher amounts of arrays data at the same time into its massive number of cores.

Looking at this algorithm's behavior in-depth, we can notice that it detect shapes through processing realized in five parallelizable phases. First, the algorithm recognizes the edges in the selected image using mechanisms that run entirely on the GPU. After, it analyzes the edge points that belong to each line of the image to detect all possible lines passing through it. Such a phase takes considerable advantage of GPU processing power since it does not require communication with the system memory.

In a first moment, each thread converts a part of the image to an array of pixel coordinates in the GPU's shared memory. A second thread then processes the array of pixel coordinates to create a Hough line in the GPU's shared memory. Only when the threads finish processing the entire set of coordinates can the Hough line be copied to the corresponding Hough space in the system's memory.

Executing the Super-Resolution algorithm upon the GPU increases the speed-up in 65% when compared to the CPU. The Super Resolution algorithm firstly analyzes each image sequence, detecting points with poor quality and replacing such points by other compatible ones from the different frame that presents better quality.

The search by similar points can include the analysis of previous or even upcoming image sequences. In this context, depending on the image sequence being processed, the Super Resolution algorithm may have to look for new frames to access several different positions in the system memory. This irregular memory access pattern increases the chances of occurring performance-degrading events due to more cache misses.

The results showed that performance gains could vary significantly depending on the algorithms' behavior. The main reason for the considerable difference is the algorithm's capability of performing its tasks without requiring excessive data transfer between the system memory and the GPU memory.

For example, the best result was achieved by the Houghlines algorithm, which manipulates images information through matrices that can be managed in parallel. This algorithm does not require data transfer among the GPU memory and the system's memory during its tasks.

6. CONCLUSIONS AND FUTURE WORK

Computer vision covers a broad range of applications in the landscape of embedded systems. However, embedded devices usually present architectural limitations due to issues like hardware-design space, temperature limits, and battery consumption.
In this sense, heterogeneous embedded systems that distribute the data to be processed between CPUs and GPUs are becoming more popular by allowing the use of high-parallelized applications. However, embedded GPUs do not have dedicated memory. Consequently, the communication between the GPU and the main memory could lead to performance leaks in applications that rely on random memory access.
Unlike high-performance GPU-based architectures, embedded devices do not support the same diversity of dedicated memories due to reduced design space. Therefore, GPUs must fetch the dataset in the main memory resulting in performance losses. Thus, in this paper, experiments were conducted to verify the feasibility of using embedded GPUs in computer vision applications.
The results showed that this approach presents performance gains in all of the algorithms adopted. The more significant effect indicated an 8 times speed-up when GPUs are taken. As future work, we intend to i) verify the impact of hybrid CPU-GPU architectures in other metrics such as temperature and power consumption, and ii) implement load balancing among various edge devices with hybrid architectures to provide scalability according to demand fluctuations in computer vision applications.

7. ACKNOWLEDGEMENT

We gratefully acknowledge the support of NVIDIA Corporation with the donation of the Jetson TX2 Development Kit used for this research and National Council for Scientific and Technological Development CNPq for the support.

8. REFERENCES

[1] Nicola Bombieri, Sara Vinco, Valeria Bertacco, and Debapriya Chatterjee. Systemc simulation on gp-gpus: Cuda vs. opencl. In *Proceedings of the eighth IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*, pages 343–352. ACM, 2012.

[2] Jiaxing Che, Chunjiang Zhao, Yunhe Zhang, Cheng Wang, Xiaojun Qiao, and Xinlu Zhang. Plant stem diameter measuring device based on computer vision and embedded system. In *World Automation Congress (WAC), 2010*, pages 51–55. IEEE, 2010.

[3] Djamila Dekkiche, Bastien Vincke, and Alain Merigot. Investigation and performance analysis of openvx optimizations on computer vision applications. In *Control, Automation, Robotics and Vision (ICARCV), 2016 14th International Conference on*, pages 1–6. IEEE, 2016.

[4] Djamila Dekkiche, Bastien Vincke, and Alain Merigot. Targeting system-level and kernel-level optimizations of computer vision applications on embedded systems. *Journal of Low Power Electronics*, 13(4):607–615, 2017.

[5] Mark Harris. Many-core GPU computing with nvidia cuda. In *Proceedings of the 22Nd Annual International Conference on Supercomputing*, ICS '08, pages 1–1, New York, NY, USA, 2008. ACM.

[6] Dominik Honegger, Helen Oleynikova, and Marc Pollefeys. Real-time and low latency embedded computer vision hardware based on a combination of fpga and mobile cpu. In *Intelligent Robots and Systems (IROS 2014), 2014 IEEE/RSJ International Conference on*, pages 4930–4935. IEEE, 2014.

[7] Andrés Felipe Hurtado, Jairo Alejandro Gómez, Víctor Manuel Peñeñory, Iván Mauricio Cabezas, and Felipe Elvira García. Proposal of a computer vision system to detect and track vehicles in real time using an embedded platform enabled with a graphical processing unit. In *Mechatronics, Electronics and Automotive Engineering (ICMEAE), 2015 International Conference on*, pages 76–80. IEEE, 2015.

[8] Faria Kalim, Shadi A. Noghabi, and Shiv Verma. To edge or not to edge? In *Proceedings of the 2017 Symposium on Cloud Computing*, SoCC '17, pages 629–629, New York, NY, USA, 2017. ACM.

[9] D. Kumar and M. A. Qadeer. Fast heterogeneous computing with cuda compatible tesla gpu computing processor (personal supercomputing). In *Proceedings of the International Conference and Workshop on Emerging Trends in Technology*, ICWET 10, page 925930, New York, NY, USA, 2010. Association for Computing Machinery.

[10] Vijay P. Kumar and Anshul Gupta. Analyzing scalability of parallel algorithms and architectures. *Journal of parallel and distributed computing*, 22(3):379–391, 1994.

[11] Rui Li, Qiming Hou, and Kun Zhou. Efficient gpu path rendering using scanline rasterization. *ACM Trans. Graph.*, 35(6):228:1–228:12, November 2016.

[12] Jacobo Lobeiras, Margarita Amor, and Ramon Doallo. Designing efficient index-digit algorithms for cuda gpu architectures. *IEEE Transactions on Parallel and Distributed Systems*, 27(5):1331–1343, 2016.

[13] Hongying Meng, Nick Pears, and Chris Bailey. A human action recognition system for embedded computer vision application. In *Computer Vision and Pattern Recognition, 2007. CVPR'07. IEEE Conference on*, pages 1–6. IEEE, 2007.

[14] Nicholas Pauly and Nader I Rafla. An automated embedded computer vision system for object measurement. In *Circuits and Systems (MWSCAS), 2013 IEEE 56th International Midwest Symposium on*, pages 1108–1111. IEEE, 2013.

[15] SV Viraktamath, Mukund Katti, Aditya Khatawkar, and Pavan Kulkarni. Face detection and tracking using opencv. *The SIJ Transactions on Computer Networks & Communication Engineering (CNCE)*, 1(3):2321–2403, 2013.

[16] Sunil Kumar Vishwakarma, Divakar Singh Yadav, et al. Analysis of lane detection techniques using opencv. In *India Conference (INDICON), 2015 Annual IEEE*, pages 1–4. IEEE, 2015.

[17] Rui Wang, Xianjin Yang, Yazhen Yuan, Wei Chen, Kavita Bala, and Hujun Bao. Automatic shader simplification using surface signal approximation. *ACM Trans. Graph.*, 33(6):226:1–226:11, November 2014.