OULU 2014
C 512

ACTA
UNIVERSITATIS OULUENSIS

Miguel Bordallo López

DESIGNING FOR ENERGY-EFFICIENT VISION-BASED INTERACTIVITY ON MOBILE DEVICES

UNIVERSITY OF OULU GRADUATE SCHOOL;
UNIVERSITY OF OULU,
FACULTY OF INFORMATION TECHNOLOGY AND ELECTRICAL ENGINEERING,
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING;
INFOTECH OULU

ACTA UNIVERSITATIS OULUENSIS
C Technica 512

MIGUEL BORDALLO LÓPEZ

DESIGNING FOR ENERGY-EFFICIENT VISION-BASED INTERACTIVITY ON MOBILE DEVICES

Academic dissertation to be presented with the assent of the Doctoral Training Committee of Technology and Natural Sciences of the University of Oulu for public defence in the OP auditorium (L10), Linnanmaa, on 15 December 2014, at 12 noon

UNIVERSITY OF OULU, OULU 2014

Acta Univ. Oul. C 512, 2014

Supervised by
Professor Olli Silvén

Reviewed by
Professor Abbes Amira
Professor Tommi Mikkonen

Opponents
Professor Holger Blume
Professor Tommi Mikkonen

ISBN 978-952-62-0671-4 (Paperback)
ISBN 978-952-62-0672-1 (PDF)

ISSN 0355-3213 (Printed)
ISSN 1796-2226 (Online)

Cover Design
Raimo Ahonen

JUVENES PRINT
TAMPERE 2014

Bordallo López, Miguel, Designing for energy-efficient vision-based interactivity on mobile devices.
University of Oulu Graduate School; University of Oulu, Faculty of Information Technology and Electrical Engineering, Department of Computer Science and Engineering; Infotech Oulu
Acta Univ. Oul. C 512, 2014
University of Oulu, P.O. Box 8000, FI-90014 University of Oulu, Finland

Abstract

Future multimodal mobile platforms are expected to require high interactivity in their applications and user interfaces. Until now, mobile devices have been designed to remain in a stand-by state until the user actively turns it on in the interaction sense. The motivation for this approach has been battery conservation.

Imaging is a versatile sensing modality that can enable context recognition, unobtrusively predicting the user's interaction needs and directing the computational resources accordingly. However, vision-based always-on functionalities have been impractical in battery-powered devices, since their requirements of computational power and energy make their use unattainable for extended periods of time.

Vision-based applications can benefit from the addition of interactive stages that, properly designed, can reduce the complexity of the methods utilizing user feedback and collaboration, resulting in a system that balances computational throughput and energy efficiency.

The usability of user interfaces critically rests on their latency. However, an always-on sensing platform needs a careful balance with the power consumption demands. Improving reactiveness when designing for highly interactive vision-based interfaces can be achieved by reducing the number of operations that the application processor needs to execute, deriving the most expensive tasks to accelerators or specific processors.

In this context, this thesis focuses on investigating and surveying enablers and solutions for vision-based interactivity on mobile devices. The thesis explores the development of new user interaction methods by analyzing and comparing means to reach interactivity, high performance, low latency and energy efficiency. The researched techniques, ranging from mobile GPGPU and dedicated sensor processing to reconfigurable image processors, provide understanding on designing for future mobile platforms.

Keywords: computer vision, energy-efficiency, mobile device, user interface

Bordallo López, Miguel, Energiatehokas kamerapohjainen vuorovaikutteisuus mobiililaitteissa.
Oulun yliopiston tutkijakoulu; Oulun yliopisto, Tieto- ja sähkötekniikan tiedekunta, Tietotekniikan osasto; Infotech Oulu
Acta Univ. Oul. C 512, 2014
Oulun yliopisto, PL 8000, 90014 Oulun yliopisto

Tiivistelmä

Tulevaisuuden multimodaalisten mobiilialustojen sovellusten ja käyttöliittymien odotetaan vaativan käyttäjältä läheistä vuorovaikutusta. Tähän saakka mobiililaitteet on suunniteltu pysymään valveustilassa siihen asti kunnes käyttäjä aktivoi laitteen. Tällä lähestymistavalla on pyritty pidentämään akun kestoa.

Kuvantaminen on monipuolinen aistimodaliteetti, joka mahdollistaa kontekstin tunnistuksen ennakoimalla huomaamattomasti käyttäjän vuorovaikutustarpeet ja suuntaamalla laskennalliset resurssit asianmukaisesti. Näköpohjaiset, jatkuvasti päällä olevat toiminnot ovat kuitenkin epäkäytännöllisiä akkukäyttöisissä laitteissa sillä niiden laskennallisen suoritustehokkuuden ja akun keston vaatimukset tekevät pidemmästä yhtäjaksoisesta käytöstä mahdotonta.

Kamerapohjaiset sovellukset voivat hyötyä interaktiivisten vaiheiden lisäämisestä. Oikein suunniteltuina ne vähentävät käyttäjäpalautetta ja -yhteistyötä hyödyntävien menetelmien monimutkaisuutta, joka saattaa laskennallisen suoritustehokkuuden ja energiatehokkuuden tasapainoon.

Käyttöliittymien käytettävyys on kriittisesti riippuvainen niiden viiveestä. Jatkuvasti päällä oleva aistiva alusta edellyttää kuitenkin tasapainottelua virrankulutuksen vaatimusten kanssa. Hyvin interaktiivisia kamerapohjaisia käyttöliittymiä suunniteltaessa reaktiivisuuden parantaminen saadaan aikaan vähentämällä prosessorin käsittelemien operaatioiden määrää, johtamalla kuormittavimmat tehtävät kiihdyttimille tai erillisille prosessoreille.

Tässä kontekstissa, väitöskirjatutkimus keskittyy tutkimaan ja tarkastelemaan mahdollistajia ja ratkaisuja kamerapohjaiseen vuorovaikutukseen mobiililaitteissa. Väitöskirja tutkii uusien käyttäjäinteraktiomenetelmien kehittämistä vuorovaikutusta, suoritustehoa, alhaista viivettä ja energiatehokkuutta tuottavia keinoja analysoimalla ja vertaamalla. Tutkitut tekniikat mobiilista grafiikkaprosessoreista ja erillis sensoriprosessoinnista uudelleen konfiguroitaviin kuvaprosessoreihin tuovat ymmärrystä tulevaisuuden mobiilien alustojen suunnitteluun.

Asiasanat: energiatehokkuus, konenäkö, käyttöliittymä, mobiililaite

To Hillevi and Henri

Preface

The research work of this thesis was conducted in the Center for Machine Vision Research of the Department of Computer Science and Engineering at the University of Oulu, Finland, between 2010 and 2014.

First of all, I would like to thank Professor Olli Silvén for supervising this thesis. Through all this years, his broad expertise has been invaluable. His trust, support and encouragement have helped me immensely to carry out this work. Working with him has been also extremely interesting.

I would like to thank Professor Jari Hannuksela for their ideas and suggestions and for the really constructive discussions that we had during all this years. Without his previous work, this thesis would just not have been possible.

Many thanks also to Docent Jani Boutellier for his helpfulness and support. He was always there to lend a hand when it was needed, from my first to my last article.

I am grateful to the reviewers of the thesis, Professor Tommi Mikkonen and Professor Abbes Amira, for their constructive feedback. The corrections made based on their comments clearly improved the scientific content of the thesis.

Much of the content of the thesis has only been possible with the cooperation of my co-authors, Dr. Alejandro Nieto, Markku Vehviläinen, Dr. Marius Tico, Dr. Lixin Fan, Henri Nykänen, Dr. Karri Niemelä, Riku Hietaniemi, Dr. Markus Turtinen and Dr. Matti Niskanen.

I would also like to thank Gordon Roberts for the language revision that improved the quality of the text. All the remaining errors are mine.

Financial support for this thesis has come from the Infotech Oulu Graduate School, TEKES, the Nokia Foundation, The Tauno Tönning Foundation, the Walter Ahlström Foundation, and the Ulla Tuominen Foundation. This support was happily welcomed and now it is gratefully acknowledged.

The Center for Machine Vision Research has been an excellent place to work. In general, I would like to thank all my colleagues for creating a pleasant atmosphere during all this years. Especially, I would like to thank Docent Abdenour Hadid for his friendship and advice, not always research related.

Traveling around the world is nicer if you meet with nice people. Thanks to Daniel Berjón and Jaime Medrano for all the fun and serious times at the conference venues.

I would like to thank my friends at the University for all the times shared during lunch, coffee breaks and sports shifts. Also many thanks to the friends in Spain that always have time to see me during the short times that I spend there.

Many thanks to my extended family in Spain that followed my steps with interest, and made an effort to keep me always up to date.

Finally, I would like to thank my parents Eugenio and Angelines and my sister Alicia for their support and understanding from many kilometers away. Visiting me all these times has made all this process much nicer.

And most of all, I would like to thank my wife Hillevi and our son Henri for their unconditional love and support during the past years and more to come.

Oulu, October 2014.

Miguel Bordallo López.

Abbreviations

ADSP	*Application Domain Specific Processor*
ASIC	*Application-Specific Integrated Circuit*
ASIP	*Application-Specific Instruction Processor*
API	*Application Programming Interface*
bpp	*Configurable Flow Accelerator*
CFA	*Configurable Flow Accelerator*
CISC	*Complex Instruction-Set Computer*
CMOS	*Complementary Metal-Oxide-Semiconductor*
CPU	*Central Processing Unit*
CPP	*Cycles per Pixel*
CUDA	*Computing Unified Device Architecture*
DSP	*Digital Signal Processor*
DOF	*Degrees of Freedom*
EPI	*Energy per Instruction*
FIFO	*First-In First-out*
FoV	*Field of View*
FPGA	*Field-Programmable Gate Array*
Fps	*Frames per Second*
FLOPS	*Floating-Point Operations per Second*
GPGPU	*General Purpose computations on Graphics Processing Units*
GPU	*Graphics Processing Unit*
GPP	*General Purpose Processor*
HCI	*Human Computer Interaction*
HD	*High Definition*
HW	*Hardware*
I/O	*Input and Output*
iLBP	*Interpolated Local Binary Pattern*
ISA	*Instruction Set Architecture*
ISE	*Instruction Set Extension*
JPP	*Joules per Pixel*
KF	*Kalman Filtering*

LBP	*Local Binary Pattern*
MIMD	*Multiple Instruction Multiple Data*
MIPS	*Millions of Instructions per Second*
MPSoC	*Multiprocessor System-on-Chip*
OpenCL	*Open Computing Language*
OpenCL EP	*Open Computing Language Embedded Profile*
OpenCV	*Open source Computer Vision Library*
OpenGL	*Open Graphics Library*
OpenGL ES	*Open Graphics Library for Embedded Systems*
PC	*Personal Computer*
PE	*Processing Element*
PIP	*Programmable Input Processor*
PISO	*Parallel Input Serial Output*
POP	*Programmable Output Processor*
RANSAC	*RANdom SAmpling Consensus*
RMSD	*Root Mean Square Difference*
RMSE	*Root Mean Square Error*
RAM	*Random Access Memory*
RGB	*Red, Green and Blue*
RGBA	*Red, Green, Blue and Alpha*
RISC	*Reduced Instruction-Set Computer*
SDRAM	*Synchronous Dynamic Random Access Memory*
SIFT	*Scale-Invariant Feature Transform*
SIMD	*Single Instruction Multiple Data*
SIPO	*Serial Input Parallel Output*
SISD	*Single Instruction Single Data*
SURF	*Speeded-Up Robust Features*
SW	*Software*
SoC	*System-on-Chip*
TCE	*TTA-based Co-design Environment*
TTA	*Transport-Triggered Architecture*
VLIW	*Very Long Instruction Word*
UI	*User Interface*
2-D	*Two-dimensional*
3-D	*Three-dimensional*

$\mathbf{x}_k$	*state vector at time instant k*
Φ_k	*state transition matrix at time instant k*
$\Gamma_k \varepsilon_k$	*uncertainty of the motion model at time instant k*
ε_k	*process noise at time instant k*
Q_k	*covariance matrix at time instant k*
η_k	*measurement noise at time instant k*
$\mathbf{H}$	*observation matrix*
Δ	*difference of*
θ_x	*translation on x-axis*
ϕ_x	*rotation around x-axis*
l_{cam}	*camera latency*
l_{dis}	*display latency*
l_{proc}	*processing latency*
l_{tot}	*total latency*
m_k	*motion vector at time instant k*

Contents

1 Introduction

"Research has shown how to solve the problems, now it is up to the industry to just implement them."

- A computer vision professor, several years ago.

Mobile communication devices have become attractive platforms for multimedia applications as their display and imaging capabilities are improving together with the computational resources. Many of the devices have increasingly been equipped with built-in cameras that allow the users to capture high-resolution still images, as well as lower resolution video frames. Lately, they have been equipped with a large set of sensors, large touch-screens and additional cameras, while future devices are expected to include even 3D displays. Fig. 1 depicts a modern mobile device equipped with several cameras and sensors.

Fig 1. Nokia Lumia 920. Current mobile devices include several cameras, sensors, and a large display within a very constrained space.

The capabilities of mobile phones in portable imaging applications are on par or exceed those of laptop computers, despite the order of magnitude disparity between the computing power budgets. In particular, for interactive purposes, the size and semi dedicated interfaces of hand-held devices pose significant opportunities over general purpose personal computer technology. Table 1 summarizes the differences between a laptop and a hand-held mobile device.

Table 1. Characteristics of typical laptop computers and hand-held mobile devices and typical laptop/mobile ratios.

	Laptop computer	Hand-held device	Typical ratio
Still image resolutions	up to 2 Mpixel	up to 12 Mpixel	0.20x
Number of displays	1	1-2	0.5x
Number of cameras	0–1	1–3	0.5x
Video resolution (display)	1920x1080/30Hz	1920x1080/30Hz	1x
Display size (inches)	10–15	2–5	5x
Processor clock (GHz)	1–3.5	0.6–2.7	1.5x
Display resolution (pixels)	1024x768–2408x1536	320x240–1920x1080	4-16x
Processor DRAM (MB)	1024–8192	256–2048	4-16x
Storage(GB)	80–1000	8–64	10–40x
Processor TDP (W)	9–55	0.5–1	10–100x
Battery capacity (mAh)	5000–13000	1000–2800	2–10x
Active-use battery life (h)	3-10	3-10	1x

On the other hand, even the most recent mobile communication devices have not used their sensing and computing resources in a truly novel manner, but are merely replicating the functionalities already provided by other portable devices, such as digital still and video cameras (Hannuksela 2008). Also the popularity of laptop PCs and portable video players, as a means to access multimedia content via WiFi or mobile networks, has clearly influenced the hand-held application designs. These design decisions combined with the battery-powered characteristic of hand-helds result in devices designed to remain in a stand-by state, until the user actively turns it on in the interaction sense.

However, if cleverly designed, mobile devices can be used in manners that essentially deviate from laptop and desktop computers. Their small size, multiple cameras, and

semi-dedicated user interfaces are still under-exploited assets. Properly combined, these characteristics should be used to build concepts that can result in novel user interfaces and applications (Ronkainen 2010). For example, the sensory capabilities of current devices allow certain control operations with taps, swipes and device shaking. Nonetheless, the battery-powered quality of these devices poses an additional challenge related to power consumption and energy efficiency.

1.1 Motivation

As it has been shown in the case of interactive games, it can be observed that when sufficient feedback is offered, the user will collaborate to ease the task at hand, improving the results and user experience. Future mobile platforms are expected to allow *always-on* interactivity in their applications and user interfaces. A key advantage of using cameras as an input modality is that it enables recognizing the context in real-time and, at the same time, provides for single-handed operations in which the users' actions are recognized without needing interactions with the screen.

Computer vision enables camera data to be utilized in user interfaces to analyze the context and automatically detect the user intentions. However, many times computer vision methods are created independently from the system where they are intended to be used. The small size and relatively reduced computing capabilities of a mobile device introduce important additional constraints. The result is that many advanced vision algorithms are still unrealizable in real devices.

Mobile devices have been identified as promising platforms for vision-based applications (Zhou *et al.* 2008). As their computational resources grew, they became increasingly suitable for tasks related to the analysis and understanding of images and videos. However, this type of applications still pose significant challenges. The **key challenges** posed by this scenario of vision-based interactivity on mobile devices are two-fold:

1. **Computing and interactivity:** Mobile device developers do not always consider the needs of computer vision. This results in vision-based applications that are in fact impractical, since their requirements of computational power and energy make them unusable for extended periods of time. Consequently, their **computational throughput** needs should be carefully balanced with the **energy-efficiency** of the system. The computational load can benefit from the addition of interactive stages

that, properly designed, can reduce the complexity of the methods utilizing user feedback and collaboration.

2. **Sensing and interactivity:** The lack of understanding of the needs of the camera-based applications make the designers opt for sub-optimal solutions, which do not necessarily include the functionality that is very much needed. This is patent in the fact that cameras and sensors have been kept off for most of the time. This becomes even more apparent in the case of mobile user interactivity, where the **latency** and **energy-efficiency** needs are even more pronounced. Most mobile devices, even now, are unable to notice if they are being held or watched, ignoring even the context that surrounds them. The consideration of the sensing platform and its relationship with the interactivity needs is paramount to improve the user experience.

There is a very notable gap between opportunities and reality. The future mobile designers need to establish a relationship between **mobile device development** and its constraints in power dissipation and energy efficiency, and **vision-based interactivity** that requires real-time computationally expensive algorithms. Both problems are intrinsically interrelated and should not be considered separately. The work presented in this thesis, situated in the intersection between mobile computing, embedded systems development and computer vision, aims to **bridge the gap**, providing understanding on how to build the future mobile platforms in order to satisfy camera-based interactivity requirements. Figure 2 depicts a diagram of the scope of the thesis.

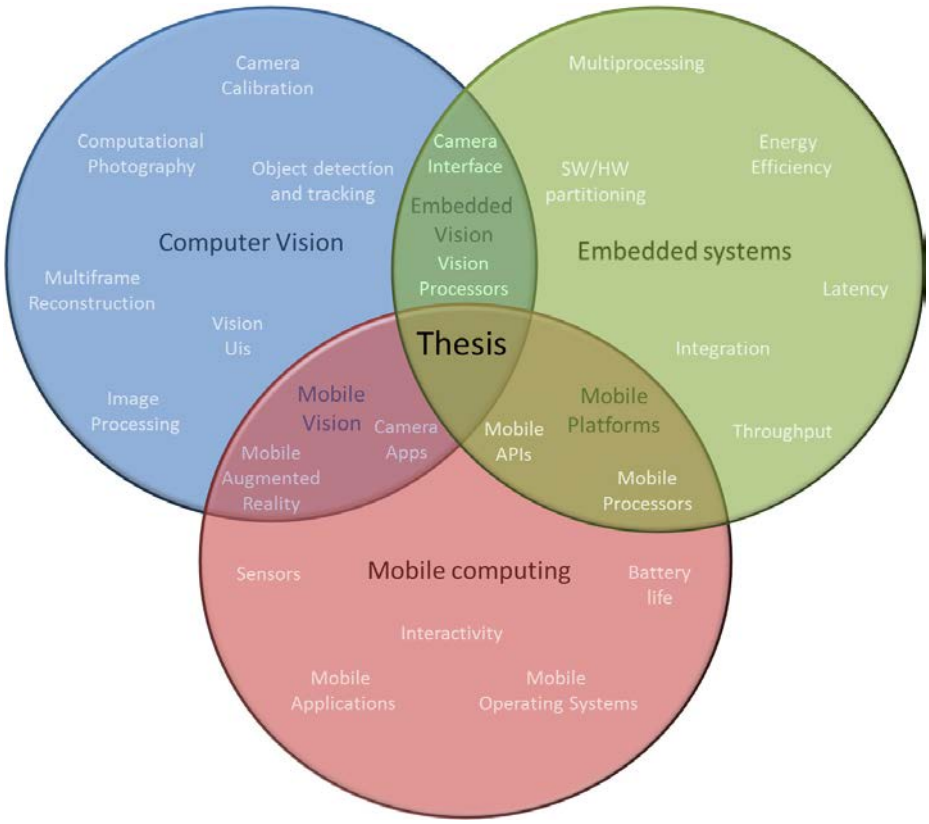


Fig 2. A diagram that covers the scope of the thesis.

The author believes that although many advanced computer vision based interaction methods have been demonstrated, their actual implementations as embedded solutions are still far from being useful. This derives, at least partially, from the implementation challenges of the highly interactive applications due to power efficiency, computing power and cost issues that many times call for a complete redesign of the applications and even the mobile platform itself. This thesis presents a set of concepts that can be used as possible solutions for such implementation problems. These challenges were revealed when the implementation of highly interactive applications was carried out on current mobile platforms. Case studies of these application types are presented and the used solutions are described at an application and system level.

1.2 Research objectives

The main research objective of this thesis is to gain understanding on how to build the future mobile platforms in order to satisfy their interactivity requirements, highlighting the development challenges and trade-offs that need to be dealt with battery powered devices.

In this context, this thesis establishes a **relationship between vision-based interactivity and energy efficiency**, formulating the fundamental principles that appear to lie beneath previous observations on camera-based user interactions of mobile devices. In addition to the analysis of the user interfaces, the work provides insight into the computing needs and characteristics of the future camera-based applications.

1.3 Contributions of the thesis

The thesis contains contributions to several individual topics but also as a whole, since the different computing needs and characteristics of the future vision-based applications, from interactive camera applications to entire user interfaces, are discussed.

Eight original publications, **Bordallo López *et al.* (2007)**, **(2009)**, **(2011a)**, **(2011b)**, **(2011c)**, **(2012a)**, **(2012b)**, **(2014)**, reflect the author's contribution to the research field. The author of this thesis had the main role in all the publications. The article co-authors provided important sub-solutions and assisted in writing.

The work described in this thesis has been mainly performed by the author under the supervision of Professor Olli Silvén, who has provided the guidelines for the work. The measurements and the implementations in sections 7.3.1 and 7.4.1 of the thesis were done in collaboration with Dr. Alejandro Nieto, and their analysis has not been published before.

In the following list summarizes the main original contributions of the thesis, each one depicted in a separate chapter:

1. Showing the importance of real-time feedback, through the implementation of an interactive multi-frame reconstruction application, including the novel concept of integrating quality assessment in the image capturing stage.
2. A novel analysis of the computing and sensing needs of camera-based user interface, through the implementation of new interaction methods on a virtual 3-D user interface based on face-tracking, demonstrated on a real mobile device. The implementation prototype is described at system and implementation levels.

3. Showing the importance of using all the available resources to hide application latency and maximize computational throughput. This was done through the analyses of alternative implementation principles for vision-based interactivity, covering general purpose serial implementations, GP-GPU based parallel implementations and dedicated processors. In addition, the challenges of highly interactive uses are also identified.
4. Bringing vision-based interactive computing to other developers through the identification of missing and unsupported abstractions of the current mobile graphics processing units APIs and tool-chains. This provides for novel insight into efficient high-performance mobile GPGPU implementation of interactive applications.
5. Steps towards always-on vision-based interactivity, analyzing its relationship with energy-efficiency. A novel insight on the use of a dedicated architecture for camera sensor processors that independently carry out most of the necessary analytic processing.
6. Advances towards platform level adaptability, presenting a comparison of reconfigurable architectures applied to interactive computer vision, identifying trade-offs and challenges.

1.4 Organization of the thesis

The thesis focuses on identifying the possible solutions for the challenges posed by highly interactive camera-based applications and user interfaces. Whenever new technological concepts are going to be integrated into an existing application system, the particularities of the implementation can have an impact in the whole platform design, and even the design principles themselves. The solutions range from software and algorithmic selection and system level redistribution of the tasks, achievable with current mobile devices, to the integration of dedicated solutions that call for a complete architecture reorganization or redesign. In this context, the structure of the thesis can be compared to a spiral, starting from the application domain and vision-based user interfacing, and progressing through issues of system organization and platform-specific implementations and designs.

Figure 3 shows the development cycle used for enabling new advanced vision-based applications and the chapters of this dissertation that correspond to each phase. The first phase is the development of vision-based applications and the identification of their computational and energy needs. Their interactivity principles are utilized in the

creation of complete vision-based interfaces that, properly designed, can be used to build novel applications. This is followed by studying the organization of the computing resources in the mobile device, characterizing the mobile architecture in terms of performance and power consumption. The analysis is proceeded by the optimization of the implementations in specific computing platforms, such as a mobile GPUs. Then, a relationship between the computing platform and the sensing devices is analyzed, formulating the principles of an energy-efficient sensing platform. Last, the platforms themselves are studied and efficient reconfigurable processors are proposed. The final integration of the redesigned platforms in future systems results in the enablement of new applications that allows the start of a new iteration of the process.

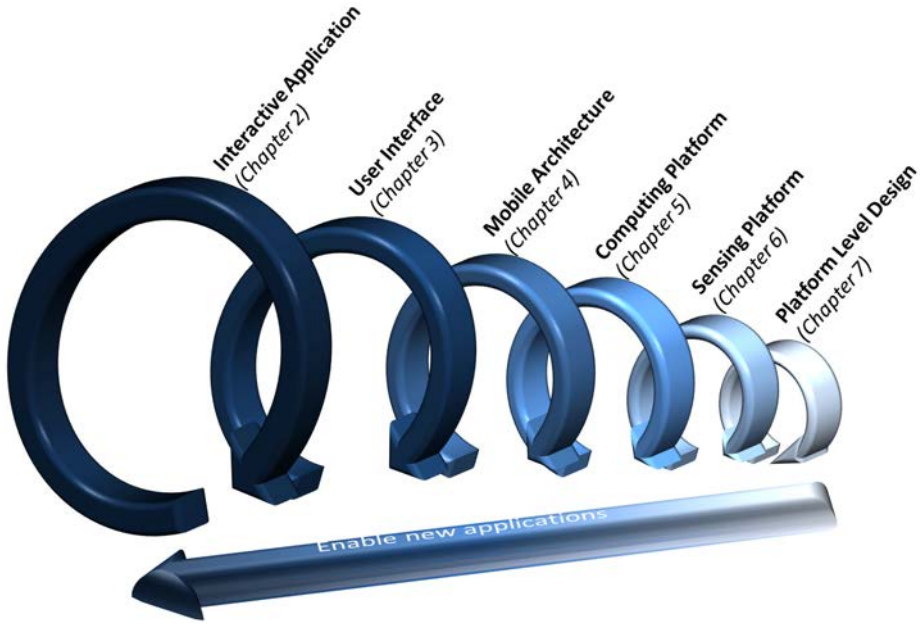


Fig 3. A spiral reflecting the outline of the thesis. The complete development cycle, from application development to hardware design, is considered.

Although the thesis presents a structure as a whole, each part of the thesis can be considered to be self-contained, presenting an introduction to the specific challenge, a review of the related work in the field, and an experimental evaluation of the solutions proposed, identifying the advantages and shortcomings. The thesis consists of six parts presented in dedicated chapters, and it is organized in the following way:

– **Interactive Camera Applications for mobile devices:** in chapter 2, the inclusion of interactive stages in camera applications is discussed. The relationship between computing and interactivity is described, analyzing the challenges presented by the high computational throughput posed by their combination. The characteristics of advanced camera applications are identified. As a result, a novel **interactive camera application** is implemented: a multi-frame reconstructor for mobile devices that includes an interactive capture stage and user feedback.
– **Vision-based user interfaces for mobile devices:** the use of camera-based methods from the viewpoint of interactive applications and user interfaces is studied in chapter 3. As a background to the work, different computer vision-based interaction techniques are reviewed and their characteristics identified. The combination of interactivity and camera-based sensing is discussed with emphasis on its main challenge, a low latency implementation. As a result, a **vision-based user interface** is implemented: a virtual 3-D display concept which utilizes head-tracking. To evaluate the usefulness of the approach, several interaction methods are proposed in the context of the implemented User Interface.
– **Mobile platform challenges in interactive computer-vision:** chapter 4 discusses the mobile device's particularities that lay behind the vision-based user interfaces and interactive applications. The computing and sensing challenges that interactivity poses to battery-powered devices are identified. As a result of this analysis, an **architectural design** based on heterogeneous mobile computing using asymmetric multiprocessing in the context of camera and sensor-based interactivity is proposed.
– **GPGPU-based interaction acceleration for mobile devices:** chapter 5 presents the use of mobile GPUs to perform general computations that can be used to accelerate mobile interactive applications and User Interfaces. As a background, the particularities of mobile GPUs in the context of general computation are described. The computing challenges of interactive vision-based applications are analyzed. The result is the development and implementation of several computer-vision algorithms on a mobile GPU, which has an impact on the consideration of the **computing platform**. Evaluated in terms of performance and energy efficiency, the applicability of the principles to interactive vision-based applications and user interfaces is discussed.
– **Sensing-assisted vision-based interactivity:** chapter 6 shows how sensors can be used to assist the context recognition on mobile devices, and how they improve the vision-based user interfaces and applications. The integration of sensor data with camera data is described. The sensing and interactivity challenges, and their impact

on battery life and energy efficiency is discussed. As a result, the use of specific sensor processors to improve the performance and energy efficiency of camera and sensor-based applications is discussed. Sensory integration poses an impact on the design of the **sensing platform**.

– **Reconfigurable computing for vision-based user interfaces on mobile devices:** in chapter 7, reconfigurable computing for vision-based interactive applications and UIs is described. With an emphasis in the impact of the combined consideration of computing, sensing and interactivity, three reconfigurable architectures, an EnCore processor with a Configurable Flow Accelerator, a hybrid SIMD/MIMD reconfigurable coprocessor, and Transport-Triggered Architecture processors, are presented and analyzed in terms of performance and energy efficiency. The advantages of integrating dedicated reconfigurable resources in mobile devices is discussed. The results have significance in the adaptation of the **design principles at a platform level**.

Each part of the thesis brings understanding on how computing, sensing and interactivity are intertwined and closely related with the energy efficiency of the system on the application, interface, architecture and platform levels. The consideration of the thesis as a whole provides insight into the process of creating novel and applicable concepts in the context of vision-based applications and user interfaces.

2 Interactive camera applications for mobile devices

"Particularly frustrating is that even in platforms like smartphones, which encourage applet creation and have increasingly capable imaging hardware, the programming interface to the imaging system is highly simplified, mimicking the physical interface of a point-and-shoot camera."

- Adams *et al.* (2010)

This chapter introduces concepts on vision-based user interactivity applied to camera applications. Cellular phone cameras have traditionally been designed to replicate the functionalities of digital compact cameras. They have been included as almost standalone subsystems rather than as an integrated part of the device interfaces. However, the programmability of modern devices and their increasing computational resources have enabled the incorporation of the camera systems as a crucial part of vision-based mobile interactive applications. However, this scenario requires low level image processing operations that are computationally costly and power-hungry.

The computational load can benefit from the addition of interactive stages that, properly designed, can reduce the complexity of the methods utilizing user feedback and collaboration. In this context, this chapter shows the importance of real-time feedback through the inclusion of an interactive stage that provides quality assessment on a camera application: a multi-frame reconstructor.

The application, used throughout the thesis as the primary camera application example that integrates the different concepts of high interactivity and energy-efficiency, shows the benefits from an interactive and enriched user experience, giving insight into how the analysis of image sequences captured by the cameras of mobile devices can be used for new self-intuitive camera applications.

2.1 Interactive camera applications

An interactive camera application is essentially an application that uses interactivity and computer vision techniques to enhance the acquisition of the best possible images in certain scenarios. In this context, a suitable solution consists of the incorporation of

interactive stages that facilitate the image capturing process. If sufficient feedback on the quality is offered, the user will collaborate to improve the final results.

The computational stages of interactive camera applications can usually be divided into two types. An online loop faces the analysis of viewfinder frames to determine the conditions of the subsequent image capture, while the second stage should analyze the captured or result image and provide feedback about its suitability for the desired results. Both stages can be considered to be data, memory and computationally intensive. In this context, it is essential that the added processing stages do not substantially add high latency to the capture stage and that the analysis of the captured frame happens in a reasonable time that allows a fast re-capture in case it is needed. Figure 4 shows the two computational stages of an interactive camera application.

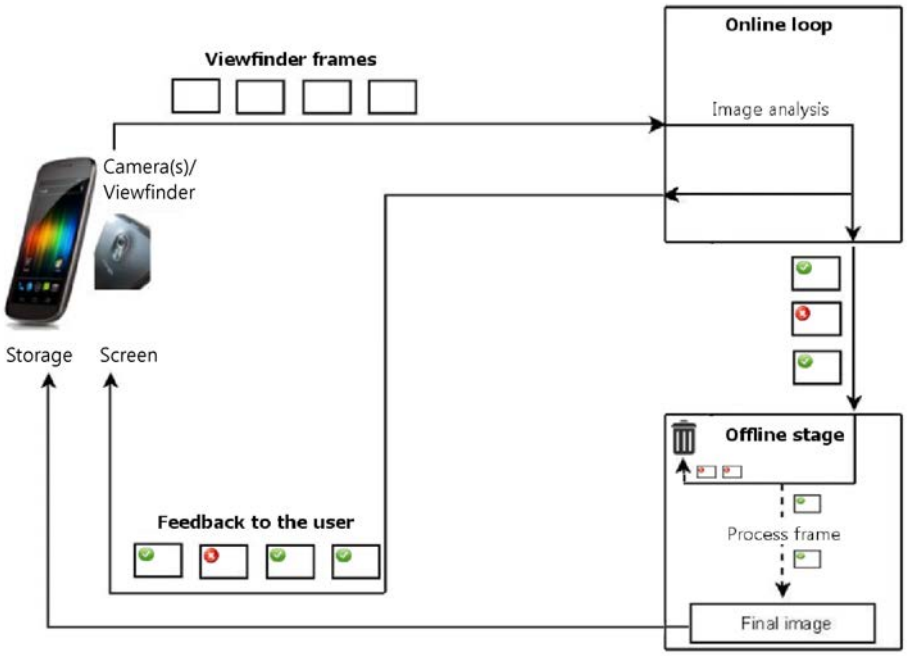


Fig 4. Computational stages of an interactive camera application. An online loop analyzes video frames in real-time, assessing its properties and giving feedback to the user. The offline stage process the selected images to compose the final result.

2.2 Related work

Recently, mobile devices equipped with a camera have been incorporating certain interactive stages to create better pictures. Current built-in camera applications are able to capture, together with regular still pictures, images composed by several frames such as panoramas, motion images, photospheres, high dynamic range pictures, or images with strong *'bokeh'* by simulating lens blur (Google Inc. 2014) (Nokia Inc. 2014a). Standalone applications that can be found in application stores range from document scanners (Intsig-Information. 2014) and photo-collages (Imagination-Unlimited. 2014) to face beautification (Tacoty-CN. 2014) and distortion (Swiss-Codemonkeys. 2014) applications. These applications usually tackle the analysis and fusion of several images into a single one that presents the desired enhanced characteristics.

The principles that lay behind them can be found in the literature. A good survey of work on mobile multi-frame techniques can be found in the work of Pulli *et al.* (2009a). Among the applications utilizing several blended input images, panorama imaging was among the first ones considered for mobile devices. Boutellier *et al.* (2007) and **Bordallo López *et al.* (2007)** introduced a mobile panorama system based on phase correlation. The system was the first one able to compute 360° images in real-time on a commercial mobile device. Later, Adams *et al.* (2008) presented another online system for building 2D panoramas. Based on aligned integral projections of the edges, they used viewfinder images for triggering the camera whenever it is pointed at previously uncaptured part of the scene. Ha *et al.* (2008) also introduced the auto shot interface to guide mosaic creation using device motion estimation. Working in real-time, the application is only able of coping with four high-resolution images, and no quality assessment on the captured set is offered. Other panorama creation applications include the work of Xiong & Pulli (2010) and Wagner *et al.* (2010). Kim & Su (1993) used a recursive method for constructing super resolution images that is applicable to mobile devices. However, no mobile implementation is presented. Another super resolution technique-based on soft learning priors can be seen in the work of Tian *et al.* (2011). Also related to down-sampling and super resolution techniques, recent Nokia Pureview devices include in their image pipeline a pixel binning technique to improve the quality of still images and video capturing, while allowing for high quality digital zooming (Vuori *et al.* 2013).

Bilcu *et al.* (2008) proposed a technique for creating high resolution high dynamic range images, while Gelfand *et al.* (2010) proposed the fusion of multi-exposure images

to increase the quality of the resulting images. Their methods, aimed to be integrated in the camera pipeline, are computationally efficient, but no performance analysis is offered.

Bordallo López *et al.* (2011a) proposed the composition of a feature cloud panorama to be utilized in interactive image recognition applications. Based on real-time feature extraction and registration, the proposed system is able to compose on-the-fly a feature-cloud that could be used in image recognition. Figure 5 depicts the feature-cloud panorama creation process.

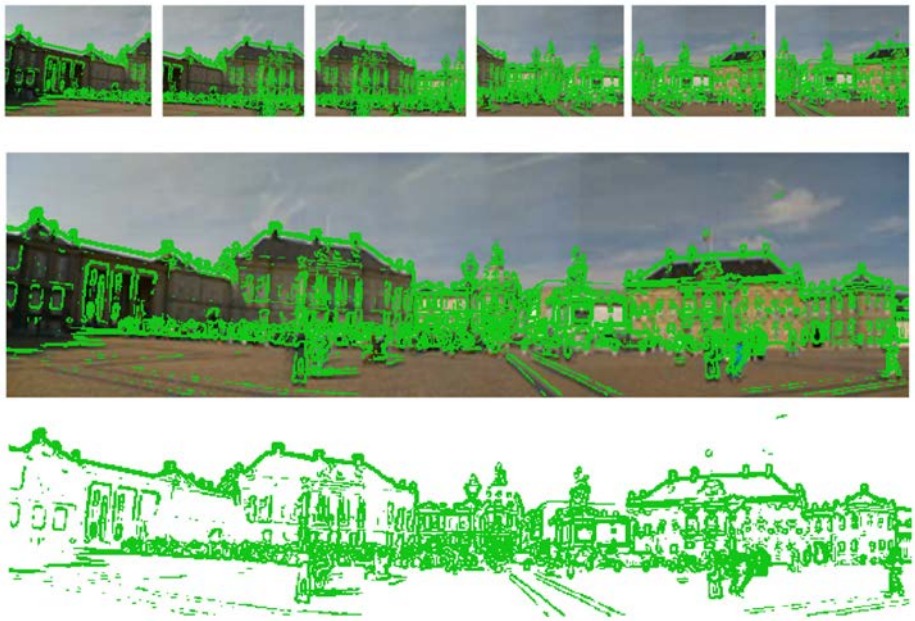


Fig 5. A feature-cloud panorama construction process. The features of a set of video-frames are aggregated into a wide field-of-view feature cloud.

Certain typical tasks, such as the capture of images from documents or business cards, have been incorporated into mobile devices. Related works are the mosaicking technique described by Hannuksela *et al.* (2007c) or the interactive assessment of the quality of the documents proposed by Erol *et al.* (2008). Both applications, implemented for real mobile devices, introduce an interactive stage to improve the capture of the source images. However, the final results rely on a computationally heavy post-processing

stage to compose the final document mosaic. A re-implementation by **Bordallo López *et al.* (2009)** aims at accelerating the interactive stage and the final post-processing stage by including software optimization techniques. Recently, several commercial applications designed to ease the document scanning process have appeared in the mobile application stores. However, those applications still lack interactivity and user feedback and require slow manual input to correct capturing defects. Figure 6 depicts a document captured by CamScanner (Intsig-Information. 2014), the most popular document scanning application for mobile devices.

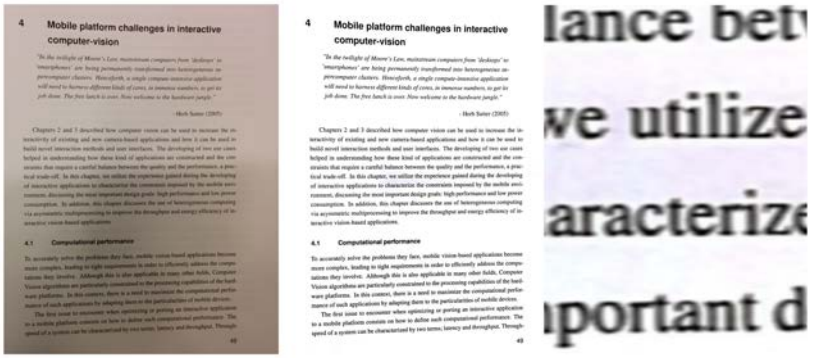


Fig 6. An example document captured by the CamScanner application on an Android device. The non-perpendicularity to the camera is not automatically detected. Details from the text appear blurry.

Other applications have emerged as a way of interactively modifying the parameters of the camera capture applications. For example, Chung *et al.* (2009) created an automatic way of classifying the capture conditions in one of seventy six available modes, offering feedback to the user of the most probable ones. Lipowezky *et al.* (2010) proposed the classification of indoor and outdoor images to adjust the capture conditions while Huttunen *et al.* (2011) implemented the real-time detection of landscape scenes to automatically switch the camera mode. Finally, Luxenburger *et al.* (2012) proposed an interactive way of creating diffusion filters during the image capturing process.

The previous works have been aiming at introducing new interactive camera applications in mobile devices. From the evaluation of the literature, it can be seen that many applications and methods are claimed to be applicable for mobile devices. However, the works analyzed do not offer a real evaluation of the implementation in terms of performance and energy consumption. Without this evaluation, it is difficult to assess the validity of these approaches being included in commercial devices. To fit

on low resource platforms, the real-time implementation of these techniques does not only consist of algorithm tweaking. Many times, the process requires a revision of the approaches, considering the application needs in terms of energy consumed and latency, when not a reimplementation from scratch.

2.3 Interactive multiframe reconstruction for mobile devices

Multiframe reconstruction techniques can be included in several interactive applications, such as scene panorama imaging, hand-held document scanning, context recognition, high dynamic range composition, super resolution imaging or digital zooming. The practical challenges of multiframe reconstruction in mobile devices lay in the implementation of the reconstruction techniques on a platform with constrained resources.

Multiframe reconstruction is a process that merges the information obtained from several input frames into a single result **(Bordallo López *et al.* 2012b) (Boutellier *et al.* 2007)** (Hannuksela *et al.* 2007c). The result can be an image that presents an increased field of view or enhanced quality, but also a feature cloud with combined information obtained from the inputs that can be used for example in object recognition **(Bordallo López *et al.* 2011a)** or even 3D reconstruction (Klein & Murray 2009).

While not a replacement for specific cameras, wide angle lenses or flatbed scanners, a multiframe image reconstructing application running on a cellular phone platform is essentially an interactive camera-based scanner that can be used in less constrained situations. The usage concept of the hand-held solution offers a good alternative, as the users cannot realistically be expected to capture and analyze single shot high-quality images of certain types of targets, such as broad scenes, three-dimensional objects, big documents, whiteboard drawings or posters.

In this context, to demonstrate camera-based interactivity, the author of this thesis has built examples of mobile multiframe reconstruction applications. Traditional multiframe reconstruction applications usually rely on the capture of several still frames that are later heavily processed offline, and blended together. Others rely on the process of compressed video frames. However, both approaches require large amounts of memory that cannot be mitigated with heavy compression, due to the apparition of artifacts. Instead, this chapter presents an approach that relies on real-time user interaction and capturing of HD-720p resolution images that are registered and stitched together. The analysis of video-frames enables the assessment of the quality of the image capturing

process, used in the interactive stage where the user can utilize the feedback offered by the device to improve the final result.

In addition to interactivity benefits, the use of low resolution video imaging can be defended from purely technical aspects. In low resolution mode, the sensitivity of the camera can be better, as the effective size of the pixels is larger, reducing the illumination requirements and improving the tolerance against motion blur (Vuori *et al.* 2013). On the other hand, a single-shot high resolution image, if captured properly, could be analyzed and used directly after acquisition, while the low resolution video capture approach requires significant post-processing effort.

Figure 7 shows the four steps of a multiframe reconstruction application. In the proposed interactive solution, the capture interface sends images to the frame evaluation subsystem and gives feedback to the user. The best frames are selected. The images are corrected, unwarped and interpolated. The final stage constructs the resulting image.

Fig 7. The four steps of a multiframe reconstruction application. Image registration aligns the features of each frame. An image selection subsystem, based on quality assessment, identifies the most suitable input images. A correction stage unwarps and enhances the selected frames. A blending algorithm composes the result image by reconstructing the final pixels. (Bordallo López *et al.* 2012b) ©Springer.

2.3.1 Interactive capture

To reconstruct a single image from multiple frames, the user faces the capture of several good quality partial images. The input frames are often far from perfect, while not unsuitable for the reconstruction. The most relevant problem when capturing several frames that are going to be merged is the camera orientation and perpendicularity to the target among the set of captured frames. The user might involuntarily tilt or shake the camera, causing the frames to have a lack of focus or motion blurriness, which will result in a low quality reconstructed image. Because a hand-held camera is used, it is difficult for the user to maintain a constant viewing angle and distance, so the user interaction scheme just aims at capturing the targets using a free scanning path. Figure 8 shows the typical problems present during the capture stage and the proposed solutions based on interactivity and quality assessment.

The key usability challenge of a hand-held camera-based multiframe reconstructor is enabling and exploiting interactivity. For this purpose, the proposed solution is to let the device interactively guide the user to move the device during the capture (Hannuksela *et al.* 2007d)**(Bordallo López *et al.* 2009)(Bordallo López *et al.* 2012b)**. This novel solution has been adopted in recent commercial products such as the Nokia Panorama software (Nokia Inc. 2014b).

In the interactive capture process, the user starts the scanning by taking an initial image of some part of the target, for example a newspaper page or a whiteboard drawing. Then, the application instructs the user to move the device to the next location. The scanning direction is not restricted in any manner, and a zig-zag style path can be used. Rotating the camera may be necessary to avoid and eliminate shadows or reflections from the target, and it offers a useful degree of freedom.

The allowed free scanning path is a very useful feature from the document imaging point of view; however, it sets significant computational and memory demands on the implementation, and it prevents building final mosaics in real time. For a more complete evaluation, a scene panorama application that limits the scanning path into a unidirectional one was also developed. With this approach, a mobile device can be used to stitch images on the fly, with the resulting images growing in real time with the frame acquisition **(Bordallo López *et al.* 2007)**. The memory requirements are smaller as not all selected frames need to be stored until the end of the panorama blending process.

To obtain a coarse camera motion model, each new video-frame needs to be individually processed to estimate motion. In this context, to provide a comparison, two

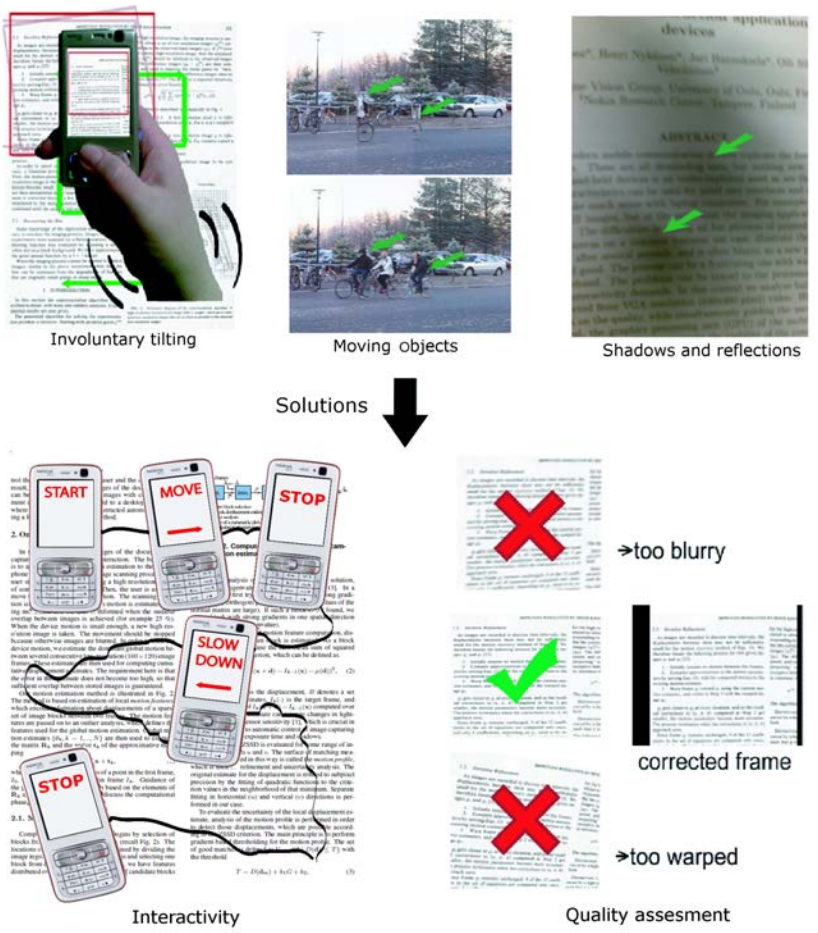


Fig 8. Image acquisition problems and the proposed solutions. (Bordallo López *et al.* 2012b) ©Springer.

different methods have been implemented. In the first one, a feature-based approach, the estimation of the motion is based on modified *Harris' corners* (Harris & Stephens 1988) and a *best linear unbiased estimator* (Henderson 1975). This method is computationally very fast, and able to provide for rotation and change of scale in addition to planar shift. A detailed description of the subsystem can be found in the article from Hannuksela *et al.* (2007b).

The second method, based on phase-correlation, utilizes the *Fast Fourier Transform* to compute the displacements between frames (Vandewalle *et al.* 2006) and has been reimplemented and improved by the author. This computationally efficient method is

also able to handle rotated frames, and it is very robust to motion-blurred images, easily found in hand-held video-frames. Also, since it relies in well-known operations such as complex arithmetics and FFTs, it is easily optimizable for speed.

In addition to the coarse registration, the blurriness of each picture is measured and eventual moving objects are detected (Boutellier *et al.* 2007). Based on shutter time and illumination-dependent motion blur, the user can be informed to slow down when the suitable overlap between images has been achieved (Hannuksela *et al.* 2007d), and a new image for stitching is selected from among the image frames, based on quality assessment. The user can also be asked to go back, or he can voluntarily return to lower quality regions later in the scanning process. As a result, good partial images of the target can be captured for the final stitching stage.

The practical result of the interactive capture stage is a set of high quality images that are pre-registered and aligned. The coarse frame registration information based on motion estimates computed during interactive scanning is employed as the starting point when constructing the mosaic image. The strategy in scanning is to keep sufficient overlaps between stored images to provision for frame re-registration using a highly accurate feature-based method during the final processing step.

2.3.2 Quality determination and frame selection

Taking pictures of documents, posters or other close objects with a hand-held camera is often hampered by the self-shadow of the device, appearing as a moving region in the sequence of frames. In practice, the regions with moving objects, whether they are shadows or something else, are not desirable when stitching the final image. Instead of developing advanced methods for coping with these phenomena, the proposed application mostly counts on user interaction to avoid the problems from harming the reconstruction result.

The treatment of shadows, reflections or moving objects depends on the type of the scene that is being processed. For every selected frame with natural scenes, if a moving object is present and fits the sub-image, the image is blended, drawing a seam that is beyond the boundaries of the object. If only a partial object is present, the part of the frame without the object is the one that is blended.

The proposed novel approach also includes a patented frame selection system that chooses only the best quality ones based on moving objects detection and blur measures **(Boutellier *et al.* 2007)**. This mobile system is the very first implementation

of this selection method. Figure 9 shows the frame selection scheme applied in panorama stitching.

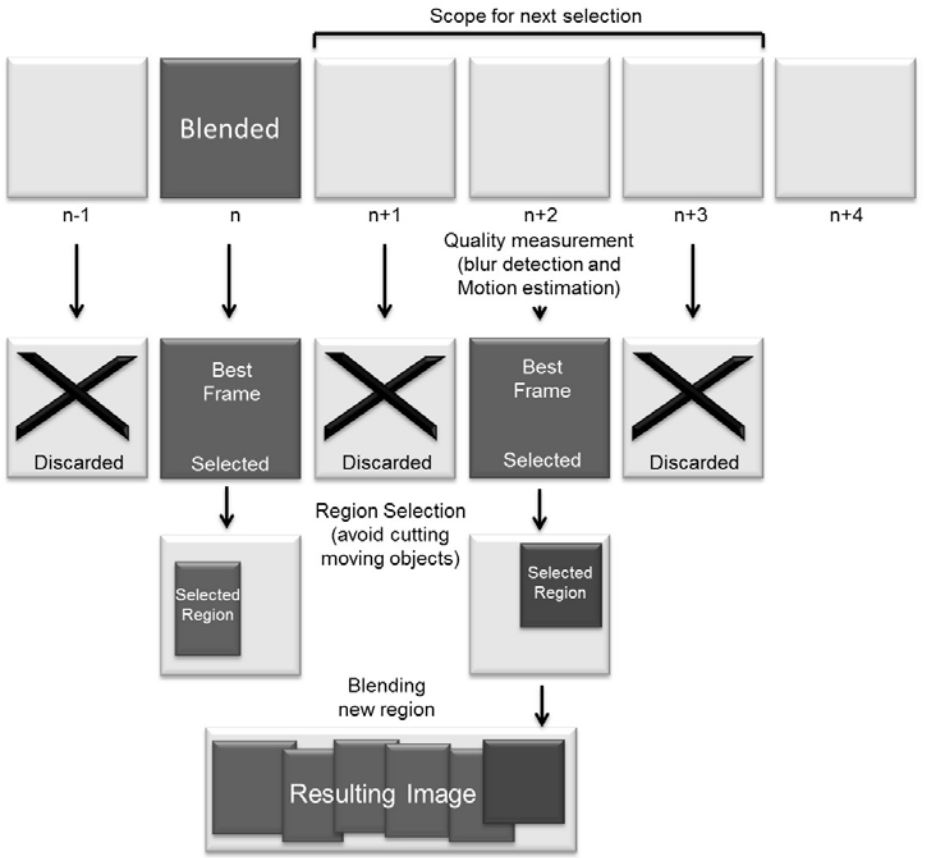


Fig 9. The implemented frame selection scheme. (Boutellier *et al.* 2007) ©SPIE.

In the subsystem, a blur detection estimates the image's sharpness by summing together the derivatives of each row and each column. When an FFT-based method is used for motion estimation, the calculated spectrum amplitude can be used by estimating the amount of high frequency components on it. Motion detection is done in a very efficient fashion to make the process fast. First, the difference between the current frame and the previous frame is computed. The result is a two-dimensional matrix that covers the overlapping area of the two frames. Then, this matrix is low-pass filtered to remove noise, and is thresholded against a fixed value to produce a binary motion map. If the binary image contains a sufficient amount of pixels that are classified as motion, the

dimensions of the assumed moving object are determined statistically. These operations are computed in real time to enable feedback to the user.

However, as the differences in the image content may distort the results, the accuracy of motion estimates used for preliminary registration needs to be reasonable. In practice, this is a computational cost, interactivity and quality trade-off. An increased accuracy implies that less overlap is needed between frames and a decrease of the computing requirements.

2.3.3 Accurate registration and blending

After on-line image capturing, the registration errors between the regions to be stitched can be on the order of pixels that would be seen as unacceptable artifacts. In principle, it would be possible to perform accurate registration during image capture, but building final document images will in any case require post-processing to adjust the alignments and scales.

The fine-registration employed for automatic mosaicking of document images is based on a *RANSAC* estimator with a *SIFT* feature point detector (Lowe 2004). In addition, graph-based global alignment and bundle adjustment steps are performed in order to minimize the registration errors and to further improve quality. Finally, warped images are blended to the mosaic, using Gaussian weighting. A more detailed description of the implementation can be found in the work of Hannuksela *et al.* (2007d).

2.3.4 Performance analysis

The computing requirements of a multiframe reconstructor are quite significant for a battery-powered mobile device, although the application can be broken down into the interactive, real-time frame capture part, and the non-interactive final mosaic stitching post-processor. It has to be noted that while the online loop needs to calculate the motion and quality for every viewfinder frame, the post-processing stage, computationally expensive, is only computed on the selected frames that will be part of the final mosaic.

The application has been developed on a Nokia N9 device. This device is based on an OMAP 3630 System on Chip which integrates a 1 GHz Cortex A8 ARM. The Nokia N9 has a 3.9 inches capacitive touchscreen with a maximum resolution of 854x480 pixels and an 8 Mpixel camera with a maximum video resolution of 2180x720 pixels. The device includes a 1450mAh battery. The implementations and optimizations have

been programmed utilizing standard C code, under a Maemo/Meego operating system. Table 2 shows the computational and energy costs of the most expensive parts of the HD-720p frame-based document scanning application when implemented entirely on the application processor.

Table 2. Algorithm's computational and energy costs per frame on an N9 (ARM Cortex-A8).

	Computation time [ms]	Energy consumption [mJ]
Online loop		
Camera motion estimation	100	200
Quality assessment	50	10
Offline computations		
Image registration	5000	800
Image correction	200	40
Image blending	100-300	40-60

The application has been implemented using only fixed point arithmetic to achieve good performance on most devices. The implementations of the interactive capture stage allow the processing of about 7.5 frames/second on Nokia N9 in HD-720p resolution mode. Compared with a camera loop that operates at 30 frames/second, it means that about one in four frames can be analyzed in real-time. The off-line stage operates at about 0.2 frames/second, and depends both on the available memory resources and processor speed.

High resolution video frames require a processing time that, although suitable for interactivity purposes, might not result in the best user experience. Reducing the resolution of the input frames proportionally decreases the needed processing times, allowing a better user experience. Table 3 shows a comparison of the processing times at different resolutions on a Nokia N9. The experiments show that processing time increases almost linearly with the number of pixels and operations.

Due to the real-time performance of the application, the resulting blended image preview is available immediately after the interactive capture stage, allowing the user to attempt a recapture if necessary. The coarse result, sufficiently accurate for display on a small screen, is then seamlessly replaced by the final high resolution image when the offline stage is completed.

Table 3. Application processing times at different resolutions on an N9 (ARM Cortex-A8).

	320x240	640x480	1280x720	1920x1080
Online loop	15ms.	52ms.	150ms.	330ms.
Offline computations	500ms.	1900ms.	5400ms.	11900ms.

2.4 Discussion

The main contribution of this chapter is showing the relationship between interactivity and improved quality, depicted in the implementation of an interactive multiframe reconstruction application that includes the novel concept of integrating quality assessment and real-time feedback in the image capturing stage.

The development and implementation of camera-based applications such as multiframe reconstruction suggest that relying on heavy, complex methods in a single post-processing stage might not be the best strategy. The addition of interactive stages that help the user collaborating in the acquisition and selection of the best possible images can easily simplify the final image composition stage.

In this context, the selection of the best images on-the-fly utilizing quality assesment and user feedback is a very useful method to reduce the number of images that have to be processed. Reducing the amount of post-processed data is very relevant, since memory needs are a usual implementation bottleneck of fine-registration with current mobile devices and it limits the size of the final reconstructed images and the number of input frames. The possibility of reducing the resolution of the input images, is also a valid strategy, but the lower quality frames cause registration and blending errors that are easy to see, and reveal the eventual shortcomings of the methodology.

In practice, in the interactive stage, there is a resolution and speed trade-off that needs to be explored. For example, while higher resolutions could offer a more accurate evaluation of the input, this usually implies a decreased ratio of images processed, which in turn could result in the selection of worse quality input images. For the offline stage, where the high quality image is composed, high resolution frames result in better final images, but could take too much time to be ready. These two trade-offs reveal the need for extremely high processing speeds in high throughput algorithms that are able to compute huge amounts of pixels in a short time, guaranteeing the best possible inputs.

3 Vision-based user interfaces for mobile devices

"There is no Moore's Law for user interfaces. Human-computer interaction has not changed fundamentally for nearly two decades. Most users interact with computers by typing, pointing, and clicking."

- Turk & Robertson (2000)

This chapter introduces concepts related to vision-based user interfaces in mobile devices. The key ideas for vision-based interactivity rest on the utilization of the hand-held nature of the equipment, and the analysis of video frames captured by the device's camera. Enabled by computer vision methods, the information provided by images allows developers to create new interactivity concepts to build the future applications.

The most challenging scenario of vision-based interactivity consists of the creation of a camera-based user interface that is able to respond to the user actions in a timely manner. The time and latency constraints introduced by the real-time nature of user interfaces adds a layer of complexity to the already computationally intensive computer vision applications.

In this context, this chapter analyzes the computing and sensing needs of camera-based user interfaces through the implementation of new interaction methods on a virtual 3-D user interface based on face-tracking. Demonstrated on a real mobile device, the implementation prototype is described on system and implementation levels. The design is used throughout the thesis as the primary user interface example that integrates the different concepts of low latency interactivity and energy-efficiency.

3.1 Vision-based user interfaces

Traditionally, vision has been utilized in perceptual interfaces to build systems that look at people, and automatically sense and perceive the human users, including their location, identity, focus of attention, facial expression, posture, gestures and movements (Pentland 2000). These systems are usually constructed taking an input from the analysis of the

data obtained by one or more cameras, and providing an output that indicates the effects of the user's actions.

Touch-sensitive screen interaction has emerged recently as a solution for mobile direct manipulation. However, it usually requires both hands and has caused additional attentional overhead (Capin *et al.* 2008). In a touch screen-based approach, the user's hand or finger can partially obstruct the view and compromise the eventual perception of the displayed augmented information. Figure 10 depicts the potential problems of touch screen interaction and view obstruction.

Fig 10. Potential problems with mobile interactivity via a touch screen. The hand or finger partially obstructs the view, and two handed operations are needed.

In vision-based user interfaces (UIs), the camera is used essentially for sensing purposes, and utilized to control some aspects of the device to interact with it. The applications of vision-based user interfaces range from gaming and augmented reality to vision-assisted general user interfaces. In this context, the usability of these kinds of interfaces rests on their robustness and latency. A fast response, perceived by the user as instantaneous, and a very low failure rate are the most critical aspects. The real-time aspect of the vision-based UIs dominates the design decisions during their development.

3.2 Related work

3.2.1 Camera motion-based UIs

Motion information-based user interfaces utilize the ego-movement of the device and its position to control a specifically-built user interface. Navigating large information spaces can be disorienting even on a large screen. In mobile devices with a small screen,

the user often encounters situations where the content that is needed for display exceeds what can be shown on the screen. For example, the need to browse through large digital images is becoming commonplace due to the increasing availability of high resolution imaging and map navigation applications.

A viable alternative for improving interaction capabilities consists of spatially aware displays (Fitzmaurice 1993). The solution is to provide a window to a larger virtual workspace where the user can access more information by moving the device around. Figure 11 depicts an example motion-based user interface (Hannuksela *et al.* 2007b) **(Bordallo López *et al.* 2011b)**. Using motion input, the user can operate the phone through a series of movements whilst holding the device with a single hand. During these movements, the motion can be extracted from one of the device's cameras.

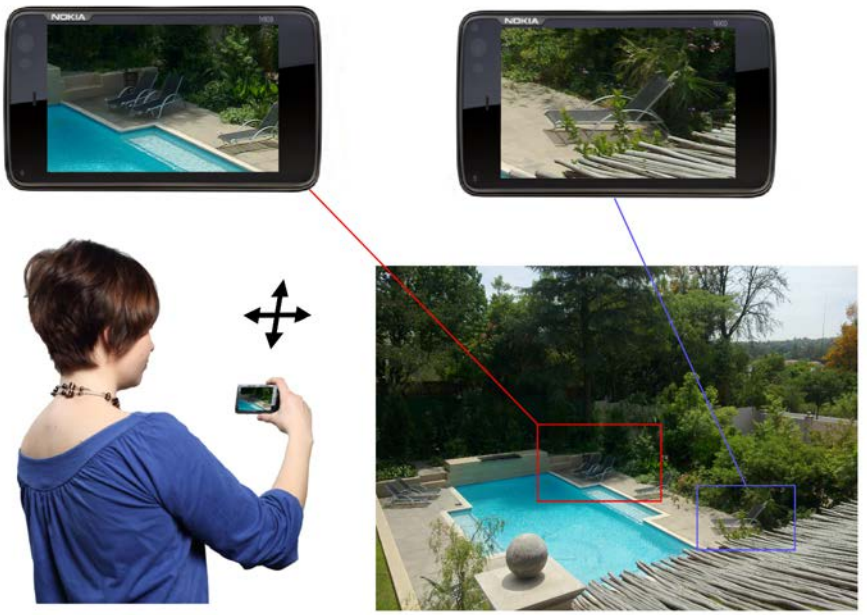


Fig 11. An example of a motion-based user interface estimates the motion of the device relative to the user or the scene, enabling browsing and zooming functionalities. Bordallo López *et al.* (2012a) ©IEEE.

The first mobile application utilizing the camera as a sensor to provide a vision-based user interface, was the augmented reality game called Mozzies, developed by Siemens in 2003 and reimplemented by the author **(Bordallo López *et al.* 2012b)**, where the

motion of the phone was recorded using a simple optical flow technique. Figure 12 depicts the reimplementation of the mosquito killing game. Since then, the rapid evolution of image sensors and computing hardware on mobile phones has facilitated the application of computer vision techniques to create new user interaction methods and a number of solutions have been proposed (Capin *et al.* 2008).

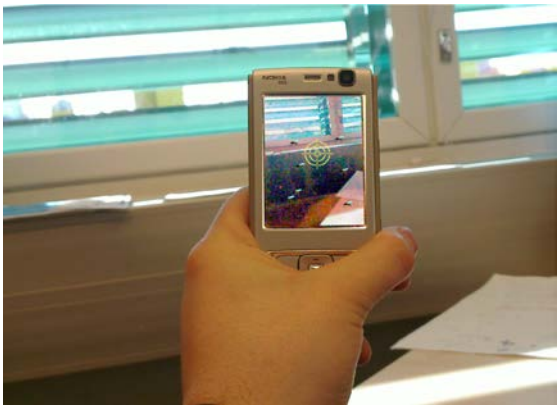


Fig 12. Reimplementation of a camera-based mosquito killing game, similar to Mozzies, which was included in the Siemens SX1 device. (Bordallo López *et al.* 2012b) ©Springer.

Much of the previous work on vision-based user interfaces with mobile phones has utilized measured motion information directly for controlling purposes. For instance, Möhring *et al.* (2004) presented a tracking system for augmented reality on a mobile phone to estimate 3-D camera pose using special color-coded markers. Other marker-based methods used a hand-held target (Hachet *et al.* 2005) or a set of squares (Winkler *et al.* 2007) to facilitate the tracking task. A solution presented by Pears *et al.* (2008) uses a camera on the mobile device to track markers on the computer display. These methods prove to be very robust, since the markers are very clear features that are easy to track. While these kind of markers have applicability in certain applications such as marker-based augmented reality (AR), there is a need for methods able to estimate motion from features extracted from natural images (You & Mattila 2013). Figure 13 depicts the author's reimplementation of three camera-based interaction methods.

An alternative to markers is to estimate motion between successive image frames with similar methods to those commonly used in video coding. For example, Rohs (2004) divided incoming frames into a fixed number of blocks and then determined the relative displacements x,y and the rotational motion using a simple block-matching

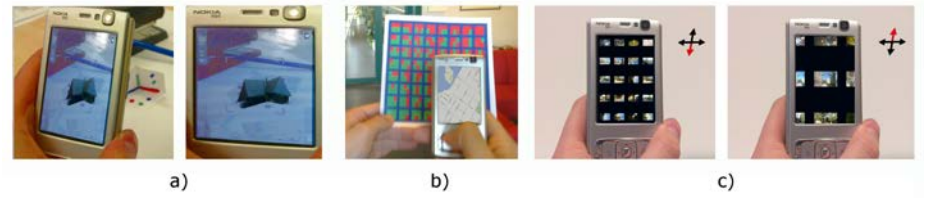


Fig 13. Mobile reimplementation of three examples of camera-based interaction. a) Color-coded markers. b) Hand-held markers. c) Ego motion image browsing. (Bordallo López *et al.* 2012b) ©Springer.

technique. Another possibility is to extract distinctive image features, such as edges and corners which exist naturally in the scene. Haro *et al.* (2005) have proposed a feature-based method to estimate movement direction and magnitude. Instead of using local features, some approaches extract global features such as integral projections from the image (Adams *et al.* 2008). The main advantage of these kind methods is that they are widely applicable in areas such as motion-based browsing or image-registration. However, they are computationally expensive and less robust against illumination changes or blur.

3.2.2 Tracking-based UIs

Tracking-based user interfaces utilize the information related to the position of a certain object to perform controlling tasks that enable interaction with the device. Many times, the object to track is known beforehand. This is the case of markers and wearable user attachments. However, the advance of tracking technology has provided the possibility of tracking certain parts of the user, such as his face, fingers or hands, providing for more natural interaction.

The usability of tracking-based user interfaces is a critical factor in these kinds of applications, and it is determined by several limiting factors. The suitability of a certain tracking algorithm and its implementation for use on mobile devices can be determined by several parameters, such as the robustness of the tracking system, the temporal and spatial accuracy, and the total end-to-end latency.

The spatial accuracy of the system is highly influenced by the camera's resolution, and determines how accurately the interactions can be measured. The accuracy is also proportional to the distance of the tracked object to the camera and the dimensions of the mobile camera sensor, with is usually small. While close objects have a pixel size equivalent to a very small real-world distance and can be very accurate, distant objects

do not allow such detailed tracking. Also, the robustness of the interface implementation can be affected by the camera's field of view (FOV), which practically defines the region in which a face, hand or finger can be tracked. In this context, a typical mobile camera has viewing angles below 60 degrees in both the vertical and horizontal axis. Lastly, the latency and the temporal accuracy are closely related to the computation time and the system architecture of the implementation.

Since Crowley *et al.* (1995) introduced the FingerPaint system, which was able to track pointing devices on a digital desk, several other applications have been used for hand and finger tracking-based interaction. In this seminal work, a finger was tracked using cross-correlation with a reference template. A survey of the techniques used for finger and hand tracking can be found in the work of Rautaray & Agrawal (2012).

The application of these techniques to interaction with mobile devices started with the work of Lumsden & Brewster (2003). Since then, several prototype mobile applications can be found in the literature. Hannuksela *et al.* (2007a) developed a Kalman filter based method that can be implemented on a mobile device. The implementation of a gesture-based mobile user interface can be found in the work of An & Hong (2011), which proposed a unique touch camera idea. Developed as a touchscreen replacement, their work proposes the use of the rear-facing camera to manipulate objects shown on the screen. The work of Hürst & van Wezel (2013) builds on this idea, and investigates the finger and hand tracking methods in order to explicitly manipulate virtual objects on augmented reality mobile applications.

Face tracking-based user interfaces utilize the position of the user's face to perform controlling tasks, or to change the representation of a scene that the user can see on the screen. The head-coupled perspective, as introduced by Rekimoto (1995), tracked the user's head position to dynamically update a 3-D projection matrix. Since then, many works have tried to improve the techniques and performance of such systems. For example, the work of Lee (2008), that used a modified Wii remote to determine the user's position relative to a big screen. This method offers excellent tracking performance although it requires the user to wear a source of infrared illumination to track the user's position.

The real-time constrains of head-coupled systems determines the little work that can be found on embedded or mobile devices. Stavness *et al.* (2010) presented a prototype of a small cube with five displays that corrects the perspective of their projections based on the user's position. The work of Francone & Nigay (2011) evaluated the user acceptability of an OpenCV-based (Viola & Jones 2001) head-coupled-perspective

application using an iPad or iPhone device. This work includes a subjective test with potential users, but does not analyze the challenges of implementing such interfaces on mobile platforms. There is still the need to address the particularities of such embedded implementations and their shortcomings, while offering a description of their computing and sensing needs.

3.3 Mobile head-tracking virtual 3-D display

3-D rendering has been traditionally used in computer graphics, together with joysticks, controllers and mice to control the camera position and point of view in interactive 3-D applications, such as video games and scene viewers. However, the limitation of these kinds of systems is that no realistic intuitive immersion is achieved, since the user has to use his hands to *'look'* at places in the virtual scenes. Head-coupled displays increase the immersive feeling by tracking the user's head position, utilizing the information to interact with the virtual scene in a natural manner.

In this context, this section presents a real-time mobile application prototype where the user's head position is determined in real time **(Bordallo López *et al.* 2012a)**. The prototype renders a customized off-axis projection matrix based on the user's point of view. Determined by the face position, the projection is used as a technique that enables the display of true three-dimensional objects, even on a typical 2-D LCD screen. Defining a novel series of interaction methods where the user's motion and camera input realistically control the viewpoint on a 3-D scene, the head movement can be used to interact with *hidden* objects in a natural manner, just by looking at them.

The user interface concept, a pure software solution, can be coupled together with lenticular display systems to provide for a natural 3D user experience. Figure 14 presents a simple user interface example. The user's point of view determines the projection of the 3-D environment on the screen.

In the proposed scenario, the user and the device can move independently with equivalent results since the computed coordinates express the relative position of the user with respect to the device, whether the movement is from the user's head or from the device itself. This technique allows the user to no longer perceive the objects as flattened, but to estimate the position and depth with respect to the screen level and to reveal objects that hide behind others, or under the borders of the displaying surface.

Based on the parallax effect of the head-coupled perspective technology, the user perceives the depth of the objects and their 3-dimensional position in space, obtaining

an immersive feeling. Although not completely three-dimensional, the user perception using a head-coupled perspective technique has shown to be more comfortable for the user than pure stereoscopic 3D and it does not require additional hardware. On the other hand, if a stereoscopic display is available, both techniques can be combined for a superior 3D effect (Li *et al.* 2012a).

3.3.1 System implementation

The proposed system can be divided into two main tasks: face-tracking and UI rendering. Consequently, the implementation of the system uses two principal threads. A background thread is constantly obtaining video frames from the frontal camera. This thread integrates the face-tracking subsystem, which continuously searches for a face in the field of view. If a face is detected, its relative 3-D position is updated to a global variable.

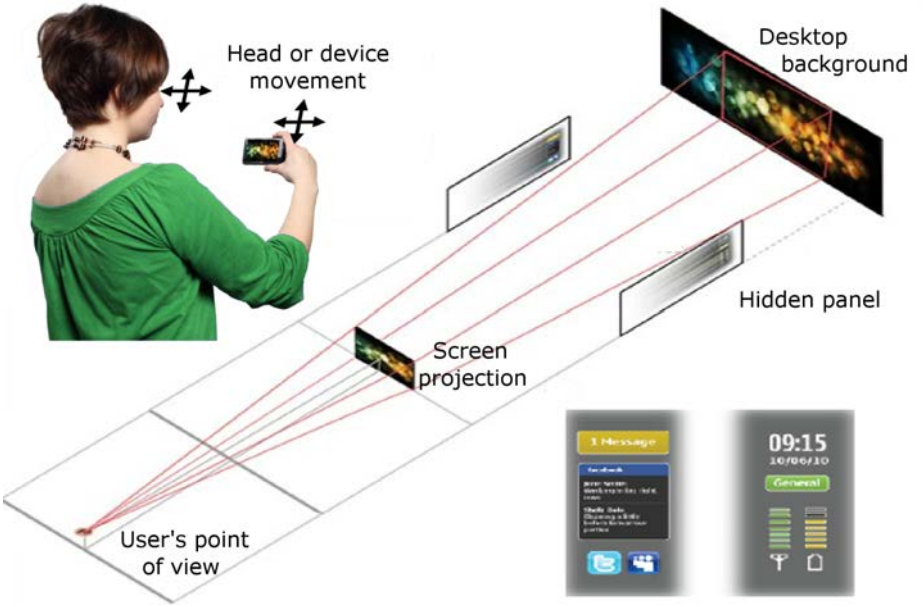


Fig 14. A simple scheme of the application: The user's point of view is determined by the relative position between the device and the head. A part of the 3-dimensional space is projected on the screen according to the position. The user can browse the desktop and two hidden lateral panels by moving the head or the device varying the point of view. (Bordallo López *et al.* 2012a) ©IEEE.

The main thread is in charge of rendering the actual user interface and handling the interactions. In the same fashion as a regular mobile touch UI, the user can interact with the objects present on the screen, and execute certain tasks. The screen rendering is done by calling a graphics subroutine that presents a three-dimensional application in the screen. The graphics subsystem needs to recalculate a customized off-axis projection matrix each time the face position is updated by the background thread.

3.3.2 Interaction design

In a typical three-dimensional use case, there is a background image situated in the infinite plane, and some panels on both sides that offer *hidden* static information that can be looked at by moving the head or the display. The most obvious interaction method consists of moving the head or the display to fix the point of view in a way that the user looks at a *hidden* object that is out of the normal field of view (Francone & Nigay 2011).

This concept can be utilized in several applications such as displaying of an image larger than the field of view (Lee 2008). The movement of the head *navigates* through the image, displaying parts of it, based on the user's point of view. The user gets the immersive sensation of looking at a scene through a window in a very natural manner. Figure 15 depicts the displaying of a landscape image on a virtual window.

Fig 15. The immersive sensation when surfing a large landscape image.

The interaction with 3D virtual objects on a scene represented on a 2D screen does not have a straightforward solution. Common interaction concepts for these kinds of objects have been traditionally limited to pure 2D pointing on the device screen, causing

a partial block of the view and a possible block of the camera view that is in charge of the face-tracking. A possible solution for this kind of interaction consists of using the back camera and finger tracking approaches to interact with the objects in a more natural manner (Hürst & van Wezel 2013). However, this method implies additional challenges such as computing overheads and the analysis of data from two cameras that must be able to run at the same time.

However, making good use of the hand-held characteristics, novel interaction methods can be developed. For interaction purposes, in relatively small screens such as those of mobile devices, it can be considered that the user is always looking around the center of the display. This fact allows the estimation of the users' gaze and intentions. Figure 16 shows a self-intuitive interaction method with *hidden* objects. When the user shifts his head to peek at hidden parts of the background, the object that is looked at can be brought to the first plane to be inspected or read, and to be used with regular touch interaction.

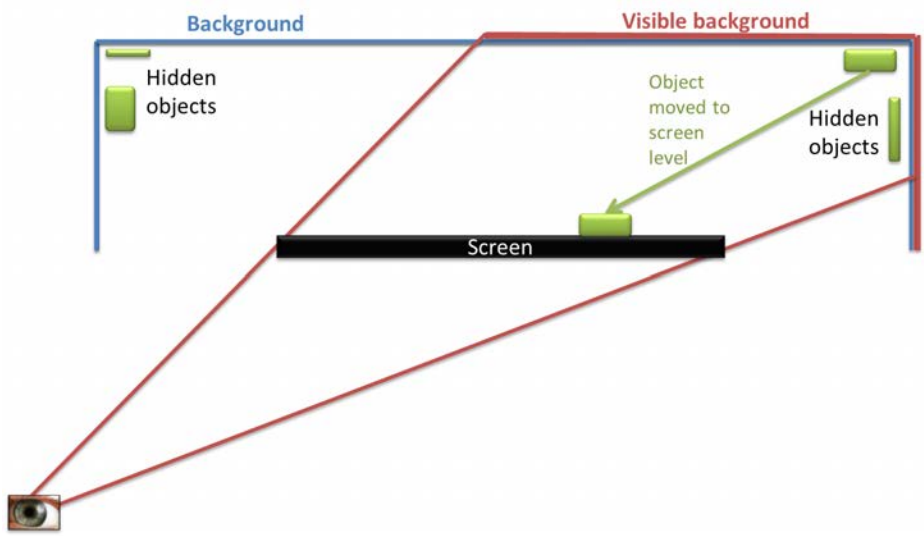


Fig 16. Objects can be moved to the screen level by simply looking at them for a second. (Bordallo López *et al.* 2012a) ©IEEE.

An example usage for the interaction method is the triggering of *hidden* dialogue boxes while using the device's web browser or the image gallery. The user can bring to the front level the history or the bookmarks by tilting the device or moving his head

to one side and keeping that position for about a second. Figure 17 depicts a mobile browser and the triggering of a *hidden* dialog box.

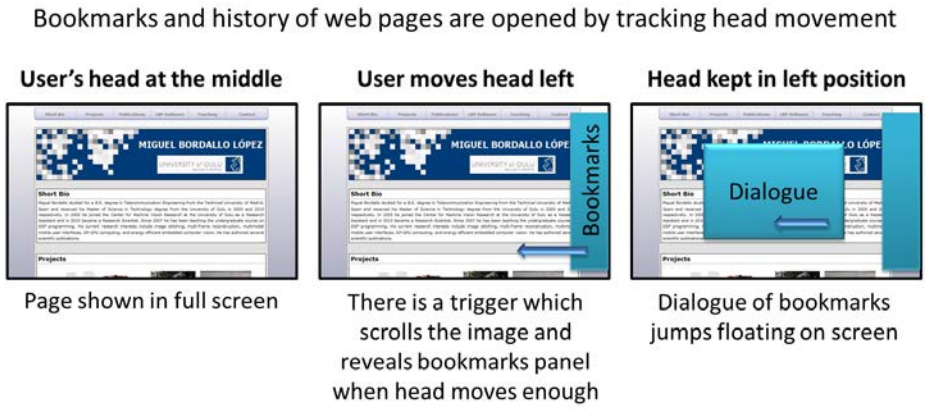


Fig 17. Web browsing use case. A dialog opens when observed for a second.

3.3.3 Face detection

In face detection and tracking, usually a few facial features are extracted from the image and compared to a predetermined feature database. Several detectors apply, for example, Haar features (Viola & Jones 2001) or local binary patterns (LBP) (Ahonen *et al.* 2006). The robust implementation of both solutions presents disadvantages related to the excessive amount of features and data needed, leading to slow processing. Reducing the amount of features or the input resolution, could result in poor consideration of a local structure of neighboring pixels, uneven illumination of the face or varying viewpoint.

Adapting the Local Binary Pattern methodology, the proposed solution utilizes a face tracking method that is based on selecting and weighting neighborhood for each pixel under observation (Niskanen *et al.* 2012). This method can be used in conjunction with the well-known learning algorithms, such as AdaBoost, that can be applied to find the most discriminative features for distinguishing the face patterns from the background.

In the adopted solution, a cascaded classifier structure speeds up the detection. In the early cascade levels only a minority of features are considered to rapidly reject the majority of classified image regions. The image is processed in multiple scales until the face is detected. Similarly, the image may be processed in parts, such as a search

window of 20 by 20 pixels. As a result, the rectangular coordinates of the detected face are obtained.

After the initial detection, only the face scales and spatial locations close to the previous locations are tracked. This improves the tracking speed with respect to the full search. During tracking, the image frame is divided into smaller windows. This way, at a specific moment, only certain windows are under observation.

The accurate determination of the face distance to the screen is highly important. Since it determines the angle from which the screen is looked at, it affects directly the field-of-view (FoV) of the rendered world that can be observed through the virtual window.

The simplest way of estimating the face distance to the screen consists of using the bounding box of the face detected by the face tracking library. This naïve method presents small flickering, or noisy variations around a central value. In addition, it is not accurate if the face is partly out of the field of view, since only the size of a partial face is detected. On the positive side, this method does not need the inclusion of any other computationally expensive operations in the rendering pipeline. Also, it does not incur in any extra latency, allowing for a high frame rate, while still providing sufficient accuracy.

To provide a comparison, we have experimented with the use of a motion estimation library (Hannuksela *et al.* 2007d) to improve the detection of the face position on the Z axis. This library is able to compute only "changes of scale" between two corresponding frames or areas, in this case, between two corresponding faces. Therefore, it also needs to use an approximate face size to compute the distance to the screen.

To test the accuracy and performance of the utilized method to calculate the distance to the screen, a database of images obtained with Microsoft Kinect has been built. The images have been calibrated (Herrera *et al.* 2011) to know the user's real distance to the camera. The same distance is then calculated using only the RGB data, the face tracking library and the calibration data obtained with the face sizes at some known distances. Three images from our dataset with the face tracking results can be seen in Figure 18.

When comparing the results of both methods side by side with the same video sequences at a fixed frame rate (15fps), the Motion Estimation Library offers 10% more accuracy using standard deviation. The Motion Estimation Library also presents less flickering.

However, the addition of extra calculations concurs with a decrease of the frame rate that increases the differences between frames, affecting the accuracy of the face tracker.

Fig 18. Three still images taken with Kinect at different distances to the camera. On the left side, only the RGB data is used to calculate the distance, determined by the face size. On the right side, the 3-D data is used as a ground truth for calibration purposes. (Bordallo López *et al.* 2012a) ©IEEE.

When comparing the performance of the Face Size method at 15 fps against the use of the Motion Estimation library at 10 fps, the accuracy of the methods is comparable, while the latency of the method that uses only face-boundary size is smaller.

3.3.4 Wide-angle lens integration

The frontal cameras of the current mobile devices are mainly intended for video-conference use, and they are usually situated in one of the device's corners. This positioning causes an input image with an asymmetrical, very reduced field of view. Since a camera-based user interface requires the user to be in the field of view, this is a major constraint. The field of view of the frontal camera of the N900 is about 40 degrees

in the vertical field and 45 degrees in the horizontal field. However, the effective field of view is reduced due to the fact that the face is not detected when it is partially out of the field of view. While at longer distances (80cm), this effect is not really appreciable, at closer distances (20cm) the field of view can be reduced to less than 30 degrees.

This problem can be mitigated by the integration of a wide-angle or fish eye lens. This lens enhances greatly the field of view at the cost of a small overhead to undistort the warping that it introduces. We have integrated in our system a mobile wide-angle lens (Photojojo 2012) that can be attached to most mobile devices with a sticker washer and a magnet. A macro lens/wide-angle lens (0.67x) extends the field of view up to 120 degrees while a fish-eye lens (0.28x) extends it to more than 160 degrees.

The distortion of the lens has been modelled and calibrated (Kannala & Brandt 2006), and the correction stage has been implemented as a lookup table that is used to remap the input image. For a VGA final size, the overhead of the correction takes about 20 ms. Figure 19 depicts the device and application when the wide-angle lens is attached. In our implementation, the input image is cropped to avoid the distortions in the borders and the horizontal effective field of view gets extended up to 130 degrees in the closest plane.

3.3.5 Performance analysis

The application has been developed on a Nokia N900 device. This device is based on an OMAP 3430 System on Chip composed by a 600 MHz Cortex A8 ARM and a Power VR SGX530 GPU, supporting openGL ES 2.0. The Nokia N900 has a 3.5 inches resistive touchscreen with a maximum resolution of 800x480 pixels and a 0.3 Mpixels front camera with a maximum video resolution of 640x480 pixels. The device includes a 1320mAh battery. The application has been programmed utilizing native standard C code under a Maemo operating system.

The system is able to run in real time on the device with an acceptable accuracy. A comparison is made between the accurate profile, slower but more robust to recovery when tracking is lost, and the fast profile, with better performance but less robustness. A performance test on an N9 platform (ARM Cortex A8, 1GHz) is also included as a comparison. Table 4 shows the processing times of the application on the N900 and N9 platforms for several tracking image sizes.

The results of the time performance and frame rates and latencies obtained with the developed application show that using a higher resolution than QVGA decreases

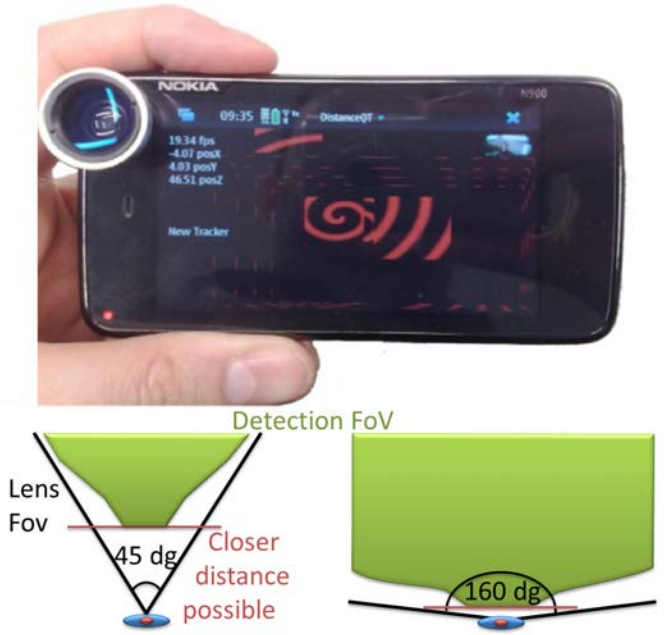


Fig 19. The wide-angle lens is integrated on the phone device to enhance the field of view (up). The effective horizontal field of view with the wide-angle lens is about 130 degrees. (Bordallo López *et al.* 2012a) ©IEEE.

Table 4. Average processing time of the application at different resolutions on N900 and N9 mobile platforms.

Tracking resolution	N900 Fast profile	Accurate profile	N9 Fast profile	Accurate profile
80x60	28 ms	39 ms	16 ms	20 ms
160x120	37 ms	54 ms	21 ms	27 ms
320x240	50 ms	75 ms	27 ms	38 ms
640x480	64 ms	95 ms	36 ms	48 ms

the frame rate and increases the latency beyond the acceptable limits. As expected, smaller resolutions present less accuracy, but can increase the frame rate. However, the experiments show that the use of faster processors, already available on current mobile platforms, would allow higher resolutions, while remaining below the tolerable processing latencies.

3.4 Discussion

The main contribution of this chapter is the analysis of the computing and sensing needs of camera-based User Interfaces, depicted in the implementation of a virtual 3-D display based on face-tracking. In addition, several novel interaction methods deriving from this kind of interface and its timing and accuracy characteristics have been presented.

Vision-based user interfaces could be integrated with current touch-based UIs, providing for complementary ways of interacting with mobile devices. New interaction methods could increase the interactivity of mobile devices in a novel manner, while their requirements in robustness and accuracy could be tackled with a careful selection of the algorithms.

The computational complexity of the computer vision algorithms required in vision-based interfaces is tightly related with the UI robustness and accuracy. There is a practical compromise between the accuracy of the methods used and the need of processing time, in a similar quality-performance trade-off as the one described in chapter 2.

However, even with plentiful computing resources, the processing of large amounts of pixels in a short time is still limited by system timing constrains. In vision-based UIs, the end-to-end latency of the utilized methods takes special relevance. Essentially, with real-time systems, the user experience is directly related to the capability of the system of providing timely results and information, in an extremely fast response time.

4 Mobile platform challenges in interactive computer vision

"In the twilight of Moore's Law, mainstream computers from 'desktops' to 'smartphones' are being permanently transformed into heterogeneous supercomputer clusters. Henceforth, a single compute-intensive application will need to harness different kinds of cores, in immense numbers, to get its job done. The free lunch is over. Now welcome to the hardware jungle."

- Herb Sutter (2005)

Chapters 2 and 3 described how computer vision can be used to increase the interactivity of existing and new camera-based applications and how it can be used to build novel interaction methods and user interfaces. The developing of two use cases helped in understanding the computing and sensing needs of this kind of applications, and the required careful balance between quality and performance, a practical trade-off.

This chapter shows the importance of using all the available resources to hide application latency and maximize computational throughput. The experience gained during the developing of interactive applications is utilized to characterize the constraints imposed by the mobile environment, discussing the most important design goals: high performance and low power consumption. In addition, this chapter discusses the use of heterogeneous computing via asymmetric multiprocessing to improve the throughput and energy efficiency of interactive vision-based applications.

4.1 Computational performance: latency and throughput

To solve the problems they face, mobile vision-based applications become more complex, leading to tight requirements in order to efficiently address the computations they involve. Although this is also applicable in many other fields, computer vision algorithms are particularly constrained to the processing capabilities of the hardware platforms. In this context, there is a need to maximize the computational performance of such applications by adapting them to the particularities of mobile devices.

The first issue to encounter when optimizing or porting an interactive application to a mobile platform is how to define such computational performance. The speed

of a system can be characterized by two terms; latency and throughput. Throughput is defined as the amount of work done per unit time. Latency is defined as the time between the start of a process and its completion. Although interrelated, a system can be designed to optimize one of both parameters, affecting the other (Grochowski *et al.* 2004). For example, pipelining an algorithm could increase its throughput, but actually increase the end-to-end latency.

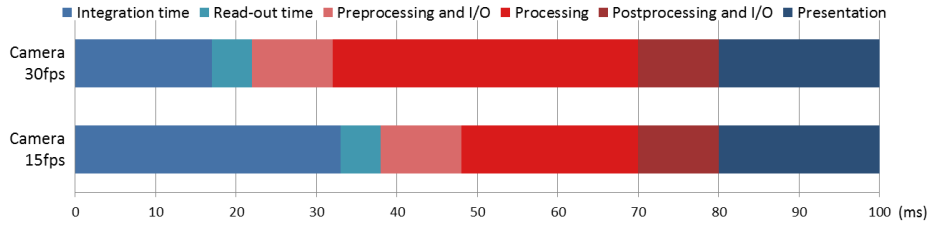
The extensive amount of data processed by vision-based applications implies that the system implementation has a high throughput requirement, since many times it should be able to compute several millions of pixel in less than a second. However, interactive applications and user interfaces are in practice real-time systems, that require a response in a limited amount of time. In this context, the implementation must assure a latency low enough to meet the requirements of interactivity. In practice, the designer needs to carefully balance both parameters in a practical trade-off.

4.1.1 Latency considerations

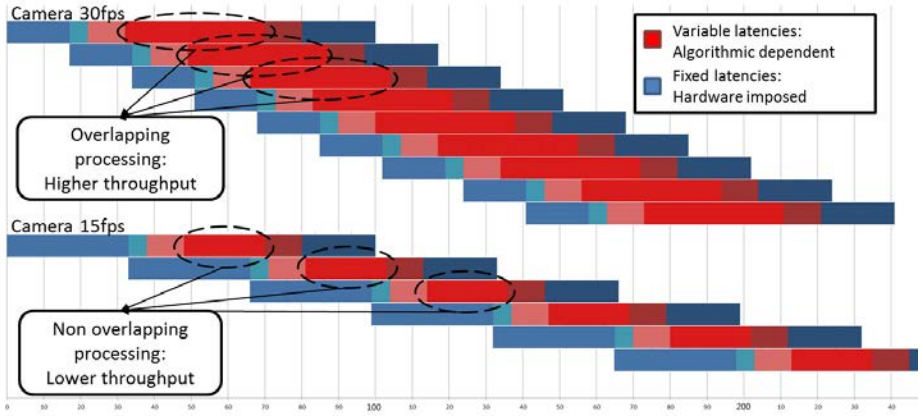
As discussed in chapter 3, the usability of camera-based user interfaces critically rests on their latency. Certain interactive applications such as web browsing can tolerate relatively long latencies (Abolfazli *et al.* 2013) that are unacceptable for others such as thin-client applications (Tolia *et al.* 2006). User interfaces are expected to have even faster response and spontaneous reflection to the arrival of new data. This becomes apparent with computer games in which action-to-display delays exceeding about 100-150 milliseconds are considered disturbing. The low-latency requirement applies even to simple key press-to-sound or display events (Dabrowski & Munson 2001).

Vision-based interactive applications employ a camera as an integral real-time application component. Consequently, in camera-based systems the sensor integration time will add to the latency, as well as the image analysis computing. When added to the presentation latency caused by the graphics overlaying and the display, it can be noted that the mobile hardware imposes a relatively long fixed latency that can not be reduced by algorithmic tweaking. Therefore, the only possible way of reducing the end-to-end latency is to cut the computing times through algorithm or implementation optimization. Figure 20(a) shows the latency budget of a vision-based interactive application. It can be seen that at a lower framerate, the fixed latencies are longer, leaving a smaller time for processing.

In order to keep the throughput of the system as high as possible, the different latencies caused by the different parts of the system can be overlapped and executed at the same time. Figure 20 shows the latency diagram of a vision-based application at 15 and 30 fps.



(a) Latency budget of a vision-based application.



(b) Timing diagram of a vision-based application.

Fig 20. Latency diagrams for a vision-based application at 15 and 30fps. The latencies caused by the different parts of the system can be overlapped to increase throughput. The timing is constrained by the fixed latencies imposed by the hardware and the variable latency that depends on the processing.

At lower frame-rates, each frame can processed sequentially. However, it can be seen, that at higher frame-rates, keeping the highest throughput means that the processing of each frame must overlap with each other. This calls for the parallel or concurrent computation of different frames, which can be seen as an argument for multiple cores or processors.

Virtual 3D display latency analysis

To provide an example of the latency measurement on a vision-based UI that requires a crisp response, the latency budget of the use case described in chapter 3 has been analyzed. As described before, the practical case consists of the projection of a scene based on the real point of view of the user or the device. Since the rendering of the user interface is based on the camera position tens of milliseconds ago, a lack of realism can be perceived. In this case, even latencies below 100 ms can be disturbing. Table 5 depicts the latency budgets of the face-tracking based UI on a Nokia N900.

Table 5. Latency budget of the application on an N900 at different resolutions (ms).

Tracking Resolution	Camera Lat.	Format Conv.	Image Analysis	GPU Rend.	Total Lat.
80x60	17	3	28	15	66
160x120	17	4	37	15	76
320x240	17	6	50	15	91
640x480	17	10	64	15	119

In the example UI, if the scene is sampled at a 30 frames/s rate, the base latency is 33 ms. Assuming that the integration time is 33 ms, the information in the pixels read from the camera is on an average 17 ms old using a rolling shutter approach. A typical touch screen has a presentation time that can be around 10 ms and the latency of the GPU rendering is about 15-20 ms. As the computing latencies need to be added, achieving the 100-150 ms range is challenging even with very fast computations. Again, the trade-off between resolution and the processing times that affect the latency can be observed.

A latency hiding technique for vision-based UIs

The effect caused by high latencies of vision-based UIs can be partially mitigated by hiding them using the accurate knowledge of the system's timing. The processing latency of the vision algorithm can be estimated as the average of $1/Fps$. Then, the total latency l_{tot} can be computed as the addition of the known camera latency l_{cam} , display latency l_{dis} and processing latency l_{proc}:

$$l_{tot} = l_{proc} + l_{cam} + l_{dis} \tag{1}$$

In order to hide the latencies that happen between the capturing and the displaying of the contents, it is possible to estimate the camera position at the moment of displaying. With the data obtained by the video analysis subsystem, and these latencies, a motion vector is constructed.

The motion vector $\mathbf{m}_k$ consists of the position $\mathbf{x}_k(x_k, y_k, z_k)$, velocities $(\dot{x_k}, \dot{y_k}, \dot{z_k})$ and accelerations $(\ddot{x_k}, \ddot{x_k}, \ddot{x_k})$ of the device at time instant k. It is defined as follows:

$$\mathbf{m}_k = [x_k, y_k, z_k, \dot{x_k}, \dot{y_k}, \dot{z_k}, \ddot{x_k}, \ddot{y_k}, \ddot{z_k}]^T. \tag{2}$$

In the beginning, the elements of the state vector are set to zero. The time step between two successive images is set to l_{proc}. The motion vector is updated every frame:

$$\ddot{\mathbf{x}_k} = \dot{\mathbf{x}_k} - \dot{\mathbf{x}_{k-1}} \tag{3}$$

$$\dot{\mathbf{x}_k} = \mathbf{x}_k - \mathbf{x}_{k-1} + \ddot{\mathbf{x}_k} * l_{tot} \tag{4}$$

Finally, the predicted position that should be used for the re-projection is calculated based on the motion vector:

$$\mathbf{x}_{predicted} = \mathbf{x}_k + \dot{\mathbf{x}_k} * l_{tot} \tag{5}$$

This technique reduces the user's perception of a lagged interface by reacting to the predicted current user's position instead of to the calculated one. Figure 21 depicts the effect of the estimated position. The predicted position is closer to the real position than the measured one. The drawback of this approach is that the accuracy of the model is hindered when the device is subject to sudden fast motion. However, in this case, the same device's fast motion makes the user less likely to perceive the possible errors, that are quickly corrected when the device's motion speed is reduced.

4.1.2 Measuring algorithmic performance across architectures

Modern computer performance is often described in MIPS (millions of instructions per second) or FLOPS (floating point operations per second). Since higher clock-rates can make a processor faster, a useful and widely applicable normalization consists of dividing the performance metric by the clock frequency, obtaining a measurement in

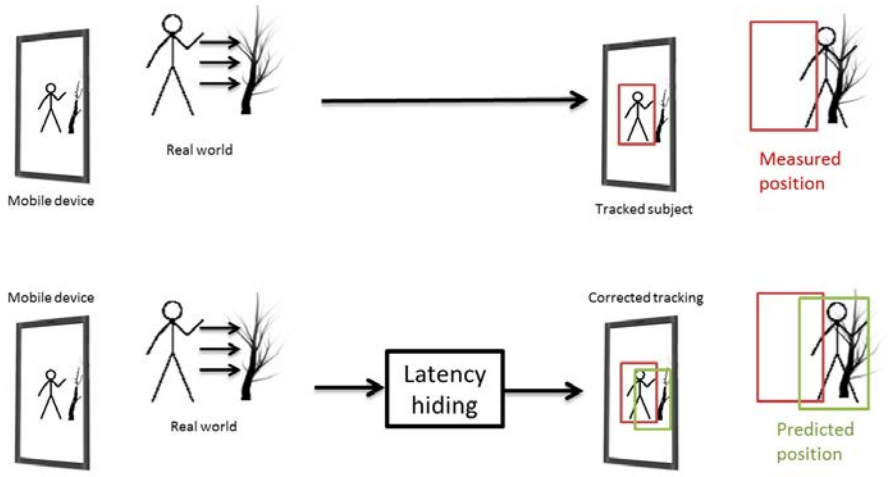


Fig 21. The effect of latency hiding on a vision-based application. The prediction of the movement compensates the latencies.

MIPS/MHz. Not considering memory access bottlenecks, simple modern processors easily reach 1 MIPS/MHz while superscalar ones might achieve rates from 3 to 5. Multi-core and many-core processors can achieve even faster rates by distributing the tasks among all the cores, computing a large amount of data per second.

Vision algorithms operate with input data that mainly consists of images composed by pixels. Since the resolution of the input images often has a direct relation with the speed of the process, a good normalization strategy to measure the performance of certain architecture for a given algorithm consists of making the performance of the metric independent of the resolution. This allows the easy calculation of the maximum possible resolution of an algorithm constrained by latency needs.

The *cycles-per-pixel* (*CPP*) metric measures the number of clock cycles required to execute all the operations of an algorithm on each one of the pixels of the resulting image, normalizing the differences in frequency and, sometimes also by the number of identical cores (*CPP per core*). The metric, used throughout the rest of the Thesis, allows the comparison of several architectures despite of their different implementations.

4.2 Energy efficiency

Along with their small size and comparatively reduced computational power, the main constraint present in mobile devices is that they are, essentially, battery powered

devices (Ferri *et al.* 2008). This implies that the application development trade-offs and challenges that need to be dealt with mobile devices are closely related not only to pure performance, but also to energy efficiency (Fabritius *et al.* 2003). In this thesis, energy-efficiency is defined as the capability of providing high computational power while presenting a low average power dissipation (Balfour *et al.* 2008).

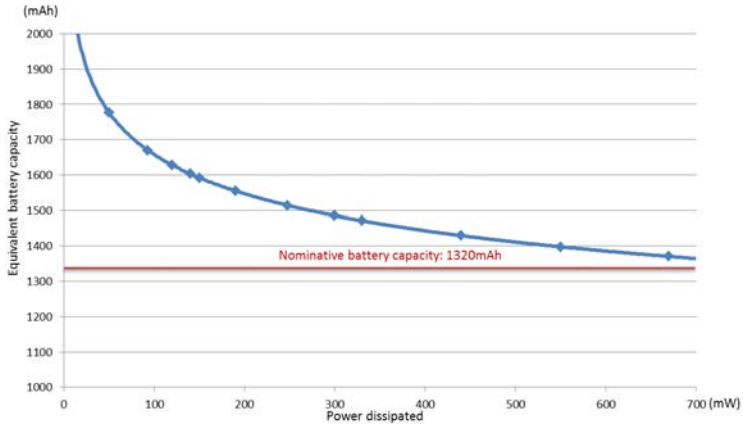
4.2.1 Characterizing battery life

The power offered by the battery is the main resource in mobile devices that requires an external source to be replenished (Miettinen & Nurminen 2010). Although current mobile devices integrate Lithium-ion batteries with moderate capacities between 1000 and 4000mAh, preserving battery life is still considered among the most important design constraints of highly interactive applications.

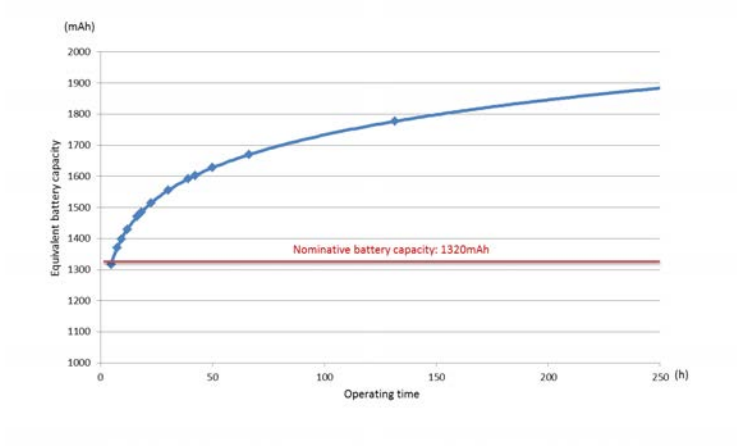
Consequently, energy efficiency is a key characteristic of a realizable interactive mobile application. Since battery-hungry applications can quickly deplete the available energy they can easily limit the usability of applications or even the device itself. In this context, the frequent need for battery recharging has become one of the most important usability issues of current mobile platforms (Heikkinen & Nurminen 2010) (Abolfazli *et al.* 2013) (Ferreira *et al.* 2011).

Although battery manufacturers are improving their technology to offer higher capacity batteries, still current battery cells are already very dense (Satyanarayanan 2005). The battery capacity growth per year can be estimated to be around 5 to 10%, clearly insufficient to catch up with the ever increasing application demands (Neuvo 2004). Alternative technologies to recharge the battery, such as harvesting from solar power or movement, are still far from offering a reliable solution in the immediate future (Pickard & Abbott 2012).

Since the battery life is a nonlinear function of the load current, small improvements in the energy efficiency of the applications can give high improvements in the operation times. Figure 22 shows the equivalent capacity of a 1320mAh Li-ON battery with different discharge times and power consumptions. The curves have been drawn assuming a Peukert coefficient of 1,1 (Peukert 1897)(Doerffel & Sharkh 2006) and utilizing the nominative battery capacity measured at a 5h discharge rate. It can be noted that with lower power consumptions, the perceived battery capacity increases dramatically.



(a) Equivalent battery capacity and power consumption



(b) Equivalent battery capacity and discharge time

Fig 22. 1320mAh Li-ON battery characterization curves. The real battery depends on the power consumption and the time taken by the discharge.

These curves show the importance of reducing power consumption to maximize battery life. However, the availability of a fast application processor enables the straightforward implementation of novel camera-based applications, and even vision-based user interfaces. On the other hand, the versatility and easy programmability of the single processor solution have led to design decisions that compromise the battery life if high interactivity is needed. In an active-state, a fast application processor may consume more than 1200mW with memories, while the whole device can go up to more than 3W.

This can push the life of typical mobile device's batteries below one hour, and increase its temperature to levels beyond the tolerable.

A possible solution has its roots in the employment of the processors at smaller loads or in the utilization of alternative processing devices that consume less power. The difference in power consumption of several mobile processors under different loads can be seen in Figure 23. The curve shows the measurements of the battery discharge times of a Nokia N9 phone under constant power consumption.

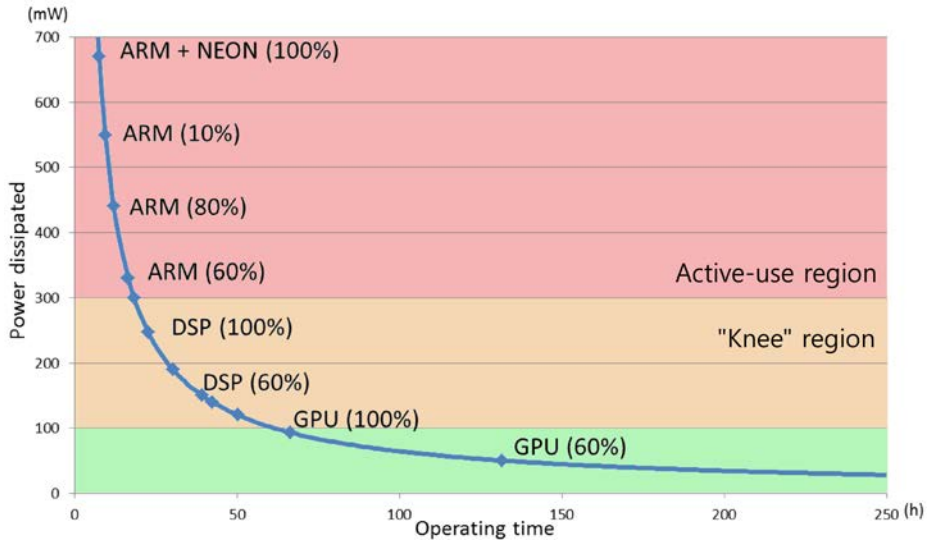


Fig 23. Discharge time of a 1320mAh Li ON battery on an N900 phone. The shape of the discharge curve implies that small improvements in the applications' energy efficiency can achieve high improvements in the operation times. (Bordallo López *et al.* 2014) ©Springer.

For applications with moderate loads, it can be seen that relatively small improvements in energy-efficiency, can push the power consumption out of the knee region, extending the battery life. Similar observations of the battery life and its knee region have been made by Silvén & Rintaluoma (2007) and can be found in the earlier work of Rakhmatov & Vrudhula (2003). Hence, an interactive applications design goal is to execute each part of the algorithm in the most suitable processor, trying to keep its load as low as possible.

4.2.2 Thermal constraints

In the unlikely event of a sudden increase in battery capacities, an additional constraint is that the high-performance computing required by interactive mobile applications has to be performed in a small spatial volume. This requires the discharge of the heat into the environment. A mobile device utilizing all its resources can easily become too hot to handle.

In addition, electronic components at very high loads can quickly run into overheating problems. Therefore, in high-performance computing, thermal issues have gained a great deal of attention in recent years. The power dissipated by a microprocessor chip per unit of area is growing steeply as the transistor densities increase (Borkar 1999). The size form of mobile devices makes typical desktop methods to prevent overheating unacceptable. That is the case of cooling fans or huge dissipators. Figure 24 depicts a set of dissipators and heat-sinks installed on a mobile SoC, the Raspberry Pi. The picture shows the obvious mechanical implementation challenges for thin devices.

Fig 24. A set of dissipators and heat-sinks on an ARM-based SoC, the Raspberry Pi.

4.2.3 Electrical power consumption

The combination of battery-life and thermal issues impose serious constrains in the power consumption of mobile devices. Power consumption on electrical circuits is composed of two main components: dynamic and static power consumption.

Dynamic power consumption, caused by the switches in the states of transistors, has a direct relation with the operating frequency. Dynamic power is used to dominate total power usage in CMOS circuits. However, in recent years, the semiconductor industry

has kept the maximum operating frequencies steady, and has instead started making a big effort to increase the parallelism of most devices. Thus, multiprocessor solutions are becoming popular in high-performance mobile devices (Horowitz *et al.* 2005). The increase of symmetric parallelism through multi-core development has allowed the steady increase of performance while keeping dynamic consumption at a reasonable level.

On the other hand, static power consumption is independent of the system activity. Caused by the leakage currents in the silicon of the circuit, it only refers to the consumption of the system when all inputs are held, and are not changing their state. In recent years, the increase in the chip density and the thinner insulations between wires has made static power consumption very significant (Kim *et al.* 2003).

The number of transistors per processor has increased greatly with impacts in power and design complexity. This complexity has caused only a modest increase in application performance, as opposed to performance due to faster clock rates from technology scaling. Modern desktop chips, with a big form factor, have around 6-7 billion transistors. Current mobile chips with a smaller form factor can contain up to 2 billion. While desktop computers, equipped with fans and dissipators, can keep the processor in full load for long periods of time, in mobile devices this is a losing strategy. Since mobile application processors consume large amounts of power, the solution has been in reducing activity or turning off complete subsystems and parts of the chip in a process known as power throttling. This process is based on two classes of techniques: Voltage/frequency scaling and gating based. Both types of techniques incur on overhead with the extra circuitry and can only cut dynamic power. As a result, large areas of silicon remain inactive most of the time, in a phenomenon named *dark silicon* (Esmaeilzadeh *et al.* 2011).

Therefore, while the bigger cores provide for higher single thread performance, they also have reduced energy efficiency. With this in mind, it becomes apparent that for the current performance increase to hold, it will still be necessary to have a proportional scaling down of the transistors. The recent failure of Dennard's scaling (Dennard *et al.* 1974) suggests that the future reduction in power consumption might not be proportional to the reduction in transistor sizes. In this context, it can be predicted that newer architectures with novel approaches to energy efficiency are likely to appear, increasing the heterogeneity of future platforms (Esmaeilzadeh *et al.* 2012). Figure 25 depicts the increase in the heterogeneity of future platforms. The progress of processor technology

moved from single-core, to multiple homogeneous cores and then to heterogeneous computing.

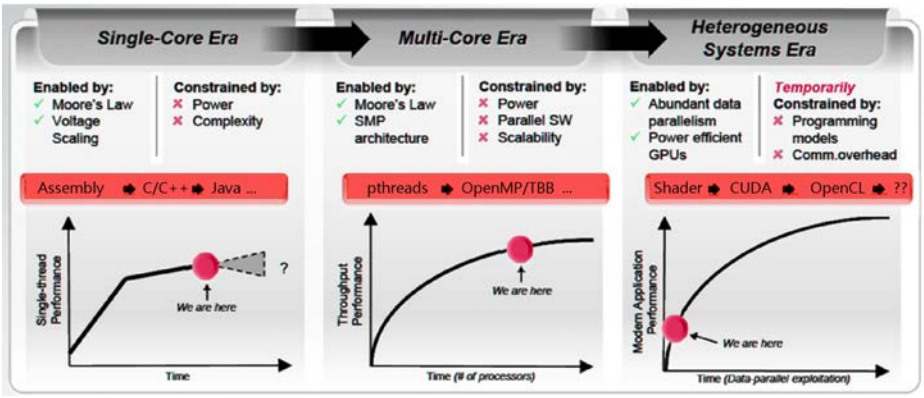


Fig 25. Progress of processor technologies. From single-core to homogeneous multi-core and heterogeneous computing. ©AMD 2014.

4.2.4 Measuring energy-efficiency

Since modern circuits have less power per transistor, as the number of transistors per chip grows, power efficiency has increased in importance. Therefore, it is advisable to know how much energy is required by certain applications when executed on a specific platform. However, predicting or estimating power for a certain algorithm running on a particular device is an extremely difficult task. Therefore, it is typical practice to measure the currents of the processor, translating them to power consumption values. In this context, several works have attempted to analyze the battery life of several mobile devices under different scenarios (Shye *et al.* 2009) (Carroll & Heiser 2010). An important notion arising from these kind of studies is that it can be established that it is sufficient to use the device's battery voltage sensors and knowledge about battery discharge behaviors to accurately estimate power consumption (Zhang *et al.* 2010) (Pathak *et al.* 2011).

The measurement of an application's power consumption is directly related to the energy-efficiency of the platform. Modern computer energy efficiency is often reported in *performance per Watt*, be it MIPS/W (millions of instructions per second

per Watt) or FLOPS/W (floating-point operations per second per Watt). However, this is not a practical measurement metric (Akenine-Möller & Johnsson 2012), since its counter-intuitive nature does not allow the easy computation of aggregate statistics. In the rest of the thesis, a different metric that allows the comparison of different implementations across several platforms is utilized. Analogously to the measurements of performance in the *CPP* metric, the *Joules per pixel* (*JPP*) metric measures the amount of energy consumed to execute all the operations of an algorithm on each one of the pixels of the resulting image.

4.3 Asymmetric multiprocessing for vision-based interactivity

Vision-based applications require very high throughputs, and low latencies. A traditional solution has been to increase not only the clock frequency but also the complexity of the application processor to achieve a better ratio of operations per cycle. However, as discussed before, the energy-efficiency of the system decreases rapidly as the frequency is scaled. An industry solution has been the integration of different processors into the same chip or as independent subsystems. However, the strong heterogeneity of the computing devices and subsystems included on a mobile device, poses a significant challenge.

The exploitation of asymmetric processors is heavily dependent on the heterogeneity of the applications. Characterizing the application needs is not easily done in a simpler manner. While many types of applications can benefit from the speed of a large core or the efficiency of a small processor, the reality is that applications are usually composed of several tasks that could easily be matched to the most suitable platform.

Some application phases might have a large amount of instruction-level parallelism (ILP) where a VLIW architecture that issues many instructions per cycle can be exploited. Other application phases might require a processor that is able to process large amounts of data under the same operation and they are more suitable for processors that exploit the SIMD paradigm. Lastly, many applications require the repetition of a single complex operation that can be executed efficiently with a clever hardware design. In practice, this heterogeneity applied to mobile devices means that a mobile developer should be able to tackle different processors that have been incorporating varied parallelization strategies and optimization (Sutter 2005).

Even with careful partition of the tasks among the asymmetric cores, heterogeneous computing requires communication between them. An important design constraint in

this kind of systems is the bandwidth of the memory access. While a certain amount of memory can be included in the specific processor, many times it is still required to access the memory in a separate component, and this will be limited by the communication bandwidth.

Fortunately, even with relatively infrequent switching among asymmetric cores, a performance and energy-efficiency increase of heterogeneous execution can be obtained (Kumar *et al.* 2005). When considering the inclusion of dedicated hardware accelerators, heterogeneous SoCs are likely to overcome homogeneous systems (Wolf 2004) in terms of performance. The average energy per instruction (EPI) of the system is also likely to be reduced, even by a factor of 4 to 6 fold (Grochowski *et al.* 2004).

4.3.1 Amdahl's law and asymmetric multiprocessing

Parallelizing the computations of a given algorithm has a theoretical limit imposed by Amdahl's law (Amdahl 1967). This principle states that if certain computation has a fraction of the program that is inherently serial and cannot be parallelized, the speedup obtained by parallel computing is limited by a factor directly proportional to the time required by the sequential calculations.

As a counterpoint, Gustafson's law points out that if the sequential part of an algorithm is fixed or grows slowly with the problem size, additional processing units can increase the problem size. In this case, the relative performance of the computation is not limited by the non-parallel part (Gustafson 1988).

Heterogeneous computing is able to tackle the implications of both statements by assigning each part of the processor (serial and parallel) to different processors that are specifically designed for each type of computations. Executing serial parts of the algorithm on a fast low-latency processor and parallel parts on many small cores can maximize the throughput while keeping the latency low. Applications based on asymmetric multiprocessing can then be designed for low latency. This implies high interactivity without compromising the general performance, directly related to the throughput. In addition, the efficiency of the specific processors also maximizes the ratio of performance to power consumption, increasing the total energy efficiency of the system.

4.3.2 Heterogeneous computing and software

The embedded nature of mobile devices implies that they are composed by carefully designed software and hardware, which should be able to work in close collaboration. The main challenge of exploiting heterogeneous computing lies in the lack of tools and models that allow the transparent usage of the asymmetric resources. Programming parallel applications requires awareness of the heterogeneity and a good understanding of the architecture.

A computing architecture can be seen as the interface between this software and hardware. Computing architectures essentially describe a computing system by specifying the relations between the parts that compose a device. In this context, the successful design and implementation of a mobile device with novel capabilities requires careful optimization across all interfaces.

Application developers are many times trained to assume that computational cores will provide similar performance *independently of the task* and that the addition of another core to the system will increase the performance similarly. The heterogeneous nature of mobile SoCs breaks the assumption, resulting in the unpredictability of the performance results.

In this context, a future area of interest is to focus on the development of interfaces that allow full access to the tool-chains, not hiding the complexity of the heterogeneity from the developers, but providing them with abstractions that allow the exploitation of the multiple cores to reduce the application latencies, while keeping the higher possible throughputs in an energy-efficient manner.

4.3.3 Concurrent heterogeneous implementation of computer vision algorithms

As discussed before, the multiple heterogeneous processing cores present in current mobile SoCs offer the possibility of increasing the overall performance of the system by using asymmetric multiprocessing. A simple way of taking advantage of the heterogeneous cores consists of dividing the application into different tasks and offloading the CPU by executing them on the most suitable processor. A more efficient variant of this approach consists of pipelining the different tasks and executing them concurrently in different processors.

However, when a given algorithm is not easily divided but the involved data is, even the same task can be carefully partitioned and scheduled over the multiple heterogeneous cores, if a good knowledge of the performance of every core can be obtained (Leskelä *et al.* 2009). In a case of no data interdependency, the time t_d used by the speed-optimal workload distribution over two different processors can be expressed with the following equation:

$$t_d = (1 - \frac{t_{proc2} + t_{tran}}{t_{cpu} + t_{proc2} + t_{tran}}) x (t_{proc2} + 2 * t_{tran}), \tag{6}$$

where t_{cpu} is the time measured using only the main processor, t_{proc2} is the time measured using only the secondary processor (e.g. mobile GPU or DSP) and t_{tran} is the time to transfer the data between processors, which in this case is considered to be CPU-bound.

In terms of power consumptions, if we define p_{cpu}, p_{proc2} and p_{tran} as the powers consumed by the corresponding actions, the total power drain p_d can be modeled as follows:

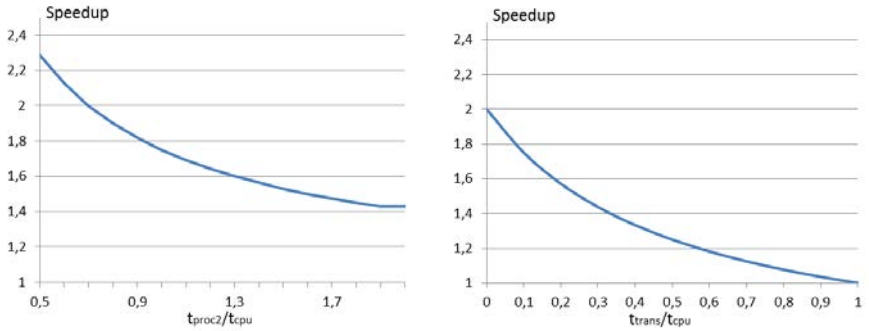
$$p_d = p_{cpu} + p_{proc2} x \frac{t_{proc2}}{t_{proc2} + t_{tran}} + p_{tran} x \frac{t_{tran}}{t_{proc2} + t_{tran}}, \tag{7}$$

and the total energy used can be obtained as:

$$E_d = p_d * t_d. \tag{8}$$

Many computer vision operations can be easily partitioned by simply dividing the input image into sections that overlap several rows and/or columns. In this context, the processing of an image can be distributed on the heterogeneous cores by dividing the data proportionally to the inverse of the computing times while keeping the number of shared neighbors as small as possible. Figure 26 shows the impact of the speed of the secondary processor and transfer times in the algorithm speedup.

It can be seen in Figure 26(a) that even secondary processors slower than the CPU can result in an algorithm speedup if both are utilized concurrently. However, the speedups can be hindered by long transfer times, as can be seen in Figure 26(b).



(a) Speedup with different processor speed (b) Speedup with different transmission times

Fig 26. Concurrent implementation speedups achieved depending on transmission/communication time and secondary processor speed.

4.4 Mobile platform as a set of heterogeneous computational devices

Mobile communication devices are becoming attractive platforms for multimedia applications as their display and imaging capabilities are improving together with the computational resources. Many of the devices have increasingly been equipped with built-in cameras that allow the users to capture high resolution still images as well as lower resolution video frames.

With power efficiency and reduced space in mind, most mobile device manufacturers integrate several chips and subsystems on a System on Chip (SoC). Figure 27 depicts a simplified diagram of the top organization of an example OMAP3430 SoC from a mobile device.

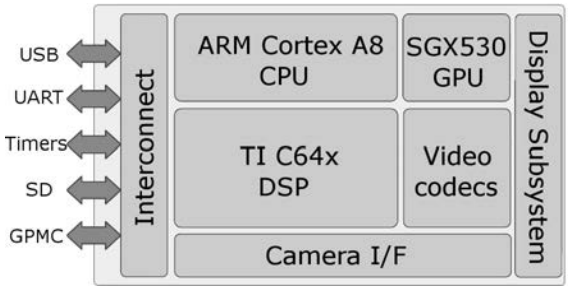


Fig 27. A simplified diagram of an example mobile SoC, the OMAP3430 mobile SoC. (Bordallo López *et al.* 2014) ©Springer.

Along with a general purpose processor (GPP), SoCs usually contain several domain-specific subsystems, such as DSPs or GPUs, packed in an application processor. As independent subsystems, they also include a set of dedicated processors that assist the computations needed by mixed-signal, camera sensors or radio functions. The SoC hardware is often shipped with a set of controlling software and APIs that handle the communication between processors, peripherals and interfaces.

4.4.1 Application processors

A mobile application processor provides a self-contained operating environment that delivers all system capabilities needed to support a device's applications, including memory management, graphics processing and multimedia decoding. Mobile application processors may be independent of other specialized processors in the same mobile device, such as a phone's base-band, camera or processor.

Mobile application processors are typically developed to consume less power and dissipate less heat than desktop computers, while using a smaller silicon size. To preserve battery life, mobile application processors can work with different power levels and clock frequencies (operating points) and it is usually possible to turn off several parts of the chip.

Single-core general purpose processors

The most typical mobile general purpose processors are based on the ARM architecture, which describes a family of computer processors designed in accordance with a RISC CPU design. The ARM architecture includes a *load/store* architecture with restrictions to misaligned memory access and a uniform 16x32-bit register file. A fixed instruction width of 32-bits allows easy decoding and pipelining with a decreased code density. However, to improve compiled code-density, most ARM processors include Thumb, a 16-bit instruction subset. A VFP (Vector Floating Point) co-processor is included to provide for low-cost floating point computations, although later versions of the architecture have abandoned it in favor of more complete SIMD units. The particularities of ARM processors enable some code optimizations to achieve higher performance. General ARM optimization tips include, for example, the use of *do-while* loops and counter decrement.

NEON co-processors and SIMD units

Despite the evolution of the industry, pure Single Instruction Single Data (SISD) microprocessors do not offer adequate performance for a large set of tasks. Many computationally intensive tasks require high performance computations that cannot be carried out efficiently by the mobile application processor alone. In this context, a wide range of accelerator modules have been included as specific arithmetic units or co-processors accessed through special sets of instructions. The inclusion of Single Instruction Multiple Data (SIMD) units is decisive for tasks such as video and image processing. SIMD processors have a unique control unit and multiple processing units. There are several ways of accessing the capabilities of modern SIMD units (Kristof *et al.* 2012), such as in-lining the corresponding assembly language instructions into the code, using array annotations that specify the sections that must be transformed from scalar to vectors, or the inclusion of *pragmas* that help the compilers to automatically vectorize suitable code.

Current mobile devices make use of SIMD units for operation parallelization. Even though the SIMD computing model presents high flexibility, these units rely on a CPU with control code execution. Many ARM-based mobile processors include a SIMD co-processor known as NEON that provides for signal processing acceleration.

NEON co-processors use combined 64- and 128-bit single instructions over multiple data, supporting up to 64-bit integer and 32-bit floating-point data types for a total of up to 16 concurrent operations at the same time. The NEON hardware shares the same floating-point registers with the VFP.

The exploitation of the NEON co-processor can be done in several ways. A compiler flag for automatic vectorization in NEON analyzes the code and vectorizes it where it is possible. A set of pragmas can be defined in the code to give information such as function pointers dependency, or minimum loop iterations, about suitable parallelizable code sections. Other ways of exploiting the NEON architecture are based on the programmer explicitly in-lining the appropriate assembly instructions on the source code, or using the NEON *intrinsics* ARM extension.

The use of a NEON co-processor slightly increases the power consumption of the ARM-based SoCs. For example, Texas Instruments OMAP3530 Power Estimation Spreadsheet(Texas-Instruments 2011) depicts a power contribution of 120 mW for the use of a NEON co-processor at 600MHz, which means about a 20% increase in power

consumption. If the performance gain is higher than such an increase, the utilization of a NEON unit implies a better performance over power.

Homogeneous multi-core architectures

Most of the latest desktop computers include General Purpose Processors with several cores. In the majority of architectures, each one of the cores is usually identical and can include a SIMD unit. Identical or different tasks can be assigned to the cores by using several Application Programming Interfaces (APIs) such as Open MP (Kristof *et al.* 2012). The multiple cores can share the data with several techniques such as shared caches or the implementation of message passing communication methods.

Multiple applications can exploit the multicore capabilities of vision-based applications. A straightforward approach consists of dividing several different and independent tasks among the total number of processors. This task parallelism is very easy to implement, since there is no data shared by the cores, and it leads to an increased throughput with the cost of a higher end-to-end latency.

For many interactive vision-based applications, where the end-to-end latency is a concern, another approach that allows us to keep the latency smaller consists of dividing the input images into equal strips and assigning them to each one of the cores, using a domain decomposition or data parallelism technique. However, several experiments suggest (Humenberger *et al.* 2009) that doubling the number of processors does not double the speed. For each partition, a set of pixels needs to be accessed by two of the cores, causing an overhead.

Experiments show that for pixel-based computations (such as convolutions) on N cores, the time consumed per frame gets reduced by a factor of approximately $0.8 * N$ to $0.9 * N$ **(Bordallo López *et al.* 2014)** The results show that using the four cores on a processor, the computations are about 3.6 times faster. The overhead in using more than one core can be caused by multiple factors such as inefficiencies in the operating system, cache utilization or contention in the access of the data shared by several cores. These results are in line with previous observations in mobile multicore systems (Blume *et al.* 2008).

Heterogeneous multi-core architectures

Mobile CPU architectures are starting to include together with several homogeneous powerful cores, different complementary low power processors that are meant to reduce the power and heat dissipation in modern SoCs (Greenhalgh 2011) (Rajovic *et al.* 2013). The different approaches for these kind of architectures can be summarized in three strategies:

In the symmetric clustering approach, the CPU architectures couples a relatively slow energy-efficient processor with a powerful core, clocked at a higher frequency. This processor couple is able to adjust better to dynamic computing needs, increasing the energy efficiency in respect to the approaches that use only clock scaling. In this kind of coupled architecture, a cache memory is shared between both cores, and the tasks are transferred from the slower to the faster processor depending on the load. Only one of the processors is powered and running at the same time.

Asymmetric clustering organizes the processors in non-symmetrical groups, where one fast core can be tied together with several low power cores or viceversa. Although the approach offers increased flexibility, only either fast or slow cores can be running at the same time, keeping the others unpowered.

Heterogeneous multiprocessing architectures allow the concurrent use of all cores, assigning high throughput tasks to faster cores and tasks with less priority or computational needs to the slower low power ones. The advantage of this approach is that all physical cores can be used concurrently.

4.4.2 Application domain specific processors

To provide for optimized solutions for specific tasks, mobile SoC has been increasingly integrating several processors directed to the execution of a set of algorithms on an application domain such as signal, graphics or image processing. Application Domain-Specific Processors (ADSPs) rely upon notions of concurrency and parallelism to satisfy performance and cost constraints resulting from increasingly complex applications and architectures. ADSPs are essentially programmable devices that aim to combine the efficiency of the hardware with the flexibility of the software. Several of these ADSPs, such as DSPs, ISPs, or GPUs have been included in mobile SoCs.

A Digital Signal Processor (DSP) is a microprocessor-based system with a set of instructions and hardware optimized for data intensive applications. Although not

exclusively used in mobile devices, DSPs are often used in mobile and embedded systems, where they integrate all the necessary elements and software. DSPs are able to exploit parallelism both in instruction execution and data processing by adopting a Harvard architecture and Very Long Instruction Words (VLIW) that allow the execution of instructions on each clock cycle. Although many DSPs have floating-point arithmetic units, fixed-point units fit better in battery-powered devices. Formerly, floating-point units were slower and more expensive, but nowadays this gap is getting smaller and smaller.

Mobile Graphics Processing Units (GPU) are specialized co-processors for graphic processing, employed to reduce the workload of the main microprocessor. They implement highly optimized graphic operations or primitives. GPUs have several independent processing units (cores) working on floating point data. Due to their higher level of parallelism, when computing graphics tasks, they have a higher operation throughput than modern microprocessors, while running at lower clock rates.

Image Signal Processors (ISPs) are subsystems designed to apply real-time image enhancement algorithms like de-mosaicking and noise reduction to war images taken by a high resolution camera sensor. Mobile ISPs aim to achieve cost, power and performance objectives by implementing most of the algorithmic and compression tasks in dedicated hardwired processing chains using minimal software.

To cope with the future needs of mobile devices, current devices are integrating other domain-specific processors such as audio processors, sensor processors or radio and network processors. Therefore the number of ADSPs included in mobile SoCs is only expected to rise, increasing the heterogeneity of mobile processing.

4.5 Discussion

The main contribution of this chapter is the analysis of alternative implementation principles for vision-based interactivity. The identified challenges in terms of latency, throughput and energy efficiency and their intertwining characteristics result in the proposal of the exploitation of a set of heterogeneous processors using asymmetric multiprocessing, maximizing the use of all the computing resources available on a mobile device.

The guidelines for high performance computing in mobile devices substantially differ from the strategies utilized in non-battery devices, such as desktop computers. The scarcity of the mobile resources calls for the wise use of the available computing

devices. Parameters such as the precision or dynamic range of the computed data do not only depend on the application itself, but also on the number and type of available processors. Designing for high throughputs is intrinsically related to the assignation of the most suitable core to the task in hand.

In this context, the extensive knowledge of the system timing in terms of performance and latency is paramount for the creation of high performance applications. A careful design and distribution of tasks could hide the possible latencies and increase the total algorithmic throughput.

However, even the highest throughput solution might not be the most suitable. The energy-efficiency of mobile SoCs has become the limiting factor in performance. Even if battery capacity problems could be mitigated, the thermal envelope that keeps heat dissipation under control will still require low power consumption. In this context, the inclusion of even more heterogeneous lower frequency components in SoCs is the natural path. Figure 28 depicts the evolution of mobile SoCs towards heterogeneity.

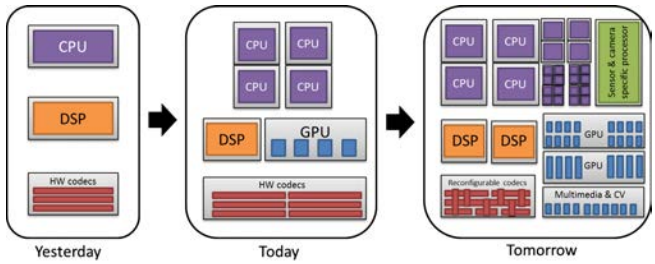


Fig 28. The evolution of mobile SoC. The increased heterogeneity calls for new toolchains. (Goodacre 2009).

The increased heterogeneity calls for a simplification and standardization of the ways of using all the included computing resources, while still providing for low level access to them. The different tools at the designer's disposal to address this task range from macro-micro architecture design, to dynamic resource management and automatic application partitioning, with the granularity of the tasks depending on the specific architecture.

5 GPGPU-based interaction acceleration

"Since real world applications are naturally parallel and hardware is naturally parallel, what we need is a programming model, system software, and a supporting architecture that are naturally parallel. Researchers have the rare opportunity to re-invent these cornerstones of computing, provided they simplify the efficient programming of highly parallel systems."

- Asanovic *et al.* (2006)

Chapters 2 and 3 focused on the better understanding of mobile vision-based interactivity, identifying the practical constrains of vision-based interactive applications, while chapter 4 provided insight into the problems and constraints of including computer vision algorithms in mobile platforms. The rest of this thesis provides an assortment of practical solutions for the challenges of these kinds of applications, especially in terms of computational throughput, latency and energy efficiency.

This chapter analyzes the means to maximize the computational platform performance to reach high interactivity by utilizing the Graphics Processing Units (GPU) included in most of the modern mobile devices.The challenges, drawbacks and opportunities of mobile GPUs are discussed in detail, identifying missing and unsupported abstractions of the current mobile GPUs, APIs and toolchains. Finally, to illustrate the increase in performance and energy efficiency that can be reached, two practical examples of GPU acceleration of interactive applications are presented.

5.1 Mobile GPU as a computing device

Current mobile platforms do not include dedicated energy efficient programmable camera processors that can be employed to develop advanced user interfaces. However, an increasing number of devices already include a Graphics Processing Unit that can be accessed with standard APIs such as OpenGL ES or OpenCL. A good solution that can be integrated into the current platforms is the employment of a GPU for general purpose computing (GPGPU), which can also improve the active-state battery life due to its lower energy consumption in comparison with a general purpose processor (GPP).

Graphics Processing Units (GPU) are specialized co-processors for graphic processing. Employed to reduce the workload of the main microprocessor in PCs, they

implement highly optimized graphic operations or primitives. Because of their very high level of parallelism, they can perform, certain operations faster than a modern microprocessor, even at lower clockrates. Also, their smaller EPI makes them a suitable candidate for reducing the power consumption of computationally intensive tasks. Therefore, using mobile GPUs for camera-based applications is an attractive option.

Graphics Processing Units present clear benefits for intensive parallel computing algorithms in terms of performance, since many of them can be considered compute-bound algorithms, meaning that the algorithm execution time is determined by the speed of the central processor, which has a 100% utilization rate. On the other hand, several algorithms do not require a high amount of operations per data byte, and are considered I/O bound or memory-bound, where the number of memory accesses and the memory access speed are the limiting factor. Since GPUs have very high memory bandwidth, they still present advantages in these kinds of algorithms (Vuduc & Czechowski 2011).

The use of a GPU proves to be very useful when some other tasks need to be done during the desired GPGPU computation. They are especially suitable for long image pipelines with long computations and a relatively low data exchange between computing entities, which also reduce the load of the CPU that does the transfer. In some new platforms, the transfer time between processors is reduced by including a high-end GPU and a multicore CPU on the same chip (Brookwood 2010). In this case, both processors (GPU and CPU) are allowed to access the main memory, eliminating the need for a peripheral bus transfer.

5.2 Mobile GPGPU

Using Graphics Processing Units (GPUs) to perform computationally intensive tasks has become popular in many industrial and scientific applications. As GPU computing is well suited for parallel processing, it is also a very interesting solution for accelerating vision-based interactive applications. Traditionally, the GPU is only used to accelerate certain parts of the graphics pipeline such as warping operations. General-purpose computing on graphics processing units (GPGPU) is the technique of using a GPU to perform computations that are usually handled by the CPU. The addition of programmable stages and high precision arithmetic allow developers to use stream processing of general data (Owens *et al.* 2007) (Che *et al.* 2008).

A mobile GPU is especially useful as a co-processor to execute certain functions, while employing its resources is most conveniently and portably done with a standard

graphics API. On a mobile device platform ,the choice is essentially limited to OpenGL ES, while the emerging OpenCL Embedded Profile is likely to offer flexibility similar to vendor-specific solutions designed for desktop computers, such as CUDA from nVidia (nVidia Inc. 2014). Most of the recent mobile phones include a graphics processor accessible via the OpenGL ES application programming interface (API).

Although the current OpenGL ES API supports a limited set of programmable function pipelines originally designed to render 3-D graphics, it can be used even in implementing image processing functions. Furthermore, future GPU interfacing APIs are likely to provide more flexibility, which helps in implementing and mapping algorithms to the GPU.

5.2.1 GPGPU through graphics interfaces

Mobile graphics APIs have been implemented in mobile devices to provide for optimized access to the graphics capabilities of the mobile SoCs. Allowing the programmers to access the hardware in an abstract way, they have been utilized in all stages of computer graphics generations. Motivated by the increased understanding of the mobile computing needs, the APIs have been evolving together with the mobile hardware to adapt to the new requirements presented by novel graphics and applications.

OpenGL ES

OpenGL (Open Graphics Library) is a multi-platform standard defining a cross-language cross-platform API used for producing 2-D and 3-D scenes from simple graphic primitives such as points, lines and triangles. OpenGL ES (OpenGL for Embedded Systems) is in turn a subset of the OpenGL 3-D graphics API designed for embedded devices such as mobile phones, tablet PCs, and video game consoles. Currently, there are several versions of the OpenGL ES specification. OpenGL ES 1.0 is drawn up against the OpenGL 1.3 specification, while OpenGL ES 1.1 and OpenGL ES 2.0 are defined relative to OpenGL 1.5 and OpenGL 2.0 specification, respectively. OpenGL ES 2.0 is not backwards compatible with OpenGL ES 1.1.

In early 2005, OpenGL ES 1.1 started to be implemented in many mobile phones, some of which included GPU hardware. The fixed point types were supported due to the lack of hardware floating-point instruction sets on many embedded processors. Many functionalities were in the original OpenGL API, although some minor functionalities

were also added. In comparison to the previous version (mostly implemented in software) OpenGL ES 1.0, the 1.1 added support for multi-texture with combiners and dot product texture operations, automatic *mipmap* generation, vertex buffer objects, state queries, user clip planes, and greater control over point rendering. The rendering pipeline is of a fixed-function type.

In practice, these features of OpenGL ES provided for limited possibilities of using the graphics accelerator as a co-processing engine. General purpose image processing capabilities were available via texture rendering. In the OpenGL ES 1.1 model, the image data can be copied to the graphics memory, allowing the application of several matrix transformations and performing bilinear interpolations for the rendered texture.

Around 2010, OpenGL ES 2.0 irrupted in commercial devices as the preferred API. It eliminated most of the fixed-function rendering pipeline API in favor of a programmable one. A shading language allows programming most of the rendering features of the transform and lighting pipelines. However, the images must still be copied to the GPU memory in a matching format and the lack of shared video memory causes multiple accesses to the GPU memory to retrieve the data for the processing engine. Although a programmable pipeline enables the implementation of many general processing functions, OpenGL ES APIs have several limitations. The most important one is that the GPU is forced to work in single buffer mode to allow the read-back of the rendered textures. Other shortcomings include the need to use power of two textures or the restricted types of pixel data.

A new version of the API, OpenGL ES 3.0, was publicly released in August 2012, and provides compatibility with desktop-based OpenGL 4.3. Backwards compatible with OpenGL ES 2.0, the new specification reimplements the GLSL ES shading language, adds computing shader capabilities and offers full support for 32-bit floating point operations and a new set of texture and render buffer objects, increasing the flexibility and portability of new applications.

Direct3D

Direct3D is a graphics and 3-D rendering API implemented by Microsoft designed to be included in all Windows-based products. Although it does not specifically define a subset of functionalities for mobile devices, the supported version for embedded GPUs tends to be reduced compared with versions supported by desktop hardware. The programming of mobile GPUs through the Direct3D API is done by using the High

Level Shading Language (HLSL), a shading description language similar to GLSL. Specific GPGPU support is only available in the newer versions of the API, not yet supported by mobile GPUs.

5.2.2 GPGPU through specific interfaces

Following the path laid by desktop GPU vendors, the mobile industry is starting to implement GPGPU-specific APIs. Designed to bring high-performance parallel computing to mobile devices, these kinds of libraries focus on providing the programmer for an interface that allows access to the architectural advantages of mobile GPUs in a simpler way, obviating the cumbersome translation into graphics primitives that is needed with graphic APIs.

OpenCL Embedded Profile

OpenCL is an API that defines the access and control of OpenCL-capable devices and it includes a C99-based language that allows the implementation of kernels on them. OpenCL simplifies the execution of task-based and data-based parallel tasks on sequential and parallel processors. Currently, there are OpenCL implementations on General Purpose Processors and Graphics Processing Units. However, several efforts have been made to port the code into other processors and platforms, such as application specific multi-cores (Jääskeläinen *et al.* 2010) or multicore DSPs (Li *et al.* 2012b).

OpenCL (Open Computing Language) is essentially an open standard for parallel programming of heterogeneous systems. It consists of an API for coordinating parallel computation across different processors and a cross-platform programming language with a well-specified computation environment. It was conceived by Apple Inc., which holds trademark rights, and established as standard by the Khronos Group in cooperation with others, and is based on C99. The purpose is to recall OpenGL and OpenAL, which are open industry standards for 3-D graphics and computer audio respectively, to extend the power of the GPU beyond graphics facilitating General Purpose computation on Graphics Processing Units.

In the OpenCL model, the high-performance resources are considered as *Compute Devices* connected to a host. The standard supports both data and task based parallel programming models, utilizing a subset of ISO C99 with extensions for parallelism. OpenCL is defined to efficiently inter-operate with OpenGL and other graphics APIs.

The current supported hardware ranges from CPUs, GPUs and DSPs to mobile CPUs such as ARM processors. Through OpenCL, multiple tasks can be configured to run in parallel on all processors in the host system, and the resulting code is portable on a number of devices. The specification is divided into a core that any OpenCL compliant implementation must support and an embedded profile which relaxes the OpenCL compliance requirements, such as data type and precision, for hand-held and mobile devices.

OpenCL defines a set of functions and extensions that must be implemented by hardware vendors. Vendors should provide the compiler and other tools to allow the execution of OpenCL code on their specific hardware. OpenCL implemented on an embedded system allows the distribution of tasks with highly parallel programming through all the processing units present on a chipset (CPU, GPU, DSP,...). Fig. 29 compares three different computational models. The OpenCL model can make use of available shared local memory to reduce the number of memory read-backs.

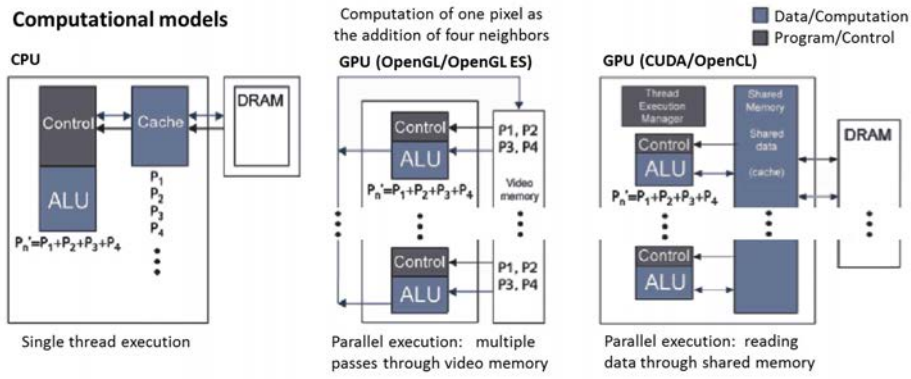


Fig 29. The figure shows three computational models. The use of shared memory reduces the number of read-backs. (Bordallo López *et al.* 2009) ©SPIE.

The execution of image processing algorithms in a single OpenCL kernel offers smaller execution overheads, lower memory bandwidth requirements and better performance comparisons than CPU alone or pure OpenGL ES implementations. Full implementations of the OpenCL standard have been recently released for the PC market and desktop GPUs, while the existing embedded profile implementations are currently being integrated in the most recent mobile device development environments.

RenderScript

Introduced in 2012, RenderScript provides a transparent acceleration engine that operates at the native level on Android platforms. Developed by Google, it can be used to accelerate applications that require extensive computational power. The RenderScript runtime environment, designed to be asynchronous, is able to compute parallel scripts using the Graphics Processing Resources, the multicore CPUs and the DSP present in mobile SoC. In a completely user-transparent way, the script is able to allocate the necessary resources to accelerate certain tasks. A drawback of the approach is that the developer has no control over where and how the code is executed, and its performance varies across implementations, platforms and devices. Also, since RenderScript hides the actual hardware information and properties, many architectural advantages such as the appropriate use of GPU memory hierarchies cannot be leveraged. A subset of RenderScript named FilterScript provides better acceleration of image filtering operations in a similar manner.

5.3 Related work

In personal computers, the use of GPUs for multimedia applications and computer vision has become commonplace. The work of Kalva *et al.* (2011) presents a good tutorial on the advantages and shortcomings of GPU platforms when developing multimedia applications, while Fung & Mann (2008) explain how to use the GPU to perform computer vision tasks.

In recent years, GPU implementations of computer vision algorithms have gained recognition as a viable option. Algorithms like scale invariant feature transform (SIFT) (Heymann *et al.* 2007), speeded up robust features (SURF) (Cornelis & Van Gool 2008), Kanade-Lucas-Tomasi (KLT)-tracker (Sinha *et al.* 2006), Advanced Phase Shifting Algorithm (APSA) **(Bordallo López *et al.* 2012c)**, nonparametric foreground segmentation (Berjon *et al.* 2013) and LBP (Zolynski *et al.* 2008) have been implemented on a desktop GPU with performance far surpassing the corresponding CPU implementations.

However, despite the tremendous popularity that GPGPU computing has obtained in recent years, the use of GPUs as general purpose capable processors has not yet been extensively considered on mobile phones. Until recently, mobile GPUs have been only

utilized for graphics and composition of the presentation layer. The first attempts at utilizing the mobile GPU for general purpose computation started in early 2009.

The first work that can be found in the literature, published by the author of this thesis (**Bordallo López *et al.* 2009**), utilized the fixed rendering pipeline offered by OpenGL ES 1.1 to warp and correct the video frames that would be utilized to compose a planar panoramic scene. The application was implemented on a real device, the Nokia N95. These first results show that it was possible to compute simple geometry transformations about four times faster than using the mobile CPU, although the overheads of copying images as textures to graphics memory resulted in significant slowdowns. Other identified problems of OpenGL ES 1.1, including the slow texture readbacks, the forced single-buffer mode, the fixed power-of-two resolutions and the required memory reordering, decreased the overall performance. The lack of programmability of the graphics interface did not allow, at the time, more complex operations that entail higher performance gains. Almost a year later, and following this first work, Pulli *et al.* (2009b) integrated a similar type of solution into a spherical projected panorama application. Reporting similar results, these works showed that the limitations of the fixed rendering pipeline for general purpose computations were a major challenge.

The introduction of the programmable OpenGL ES 2.0 pushed the development of new applications. Nah *et al.* (2010) utilized the GPU to improve and accelerate a ray-tracer. Integrated in a real application, the GPU was used to create realistic graphics in a programmable way. However, no clear evaluation of the system performance can be found.

The firsts evaluations of the performance of mobile GPUs in a GPGPU context start with the work of Singhal *et al.* (2010), who calculated the gains of mobile GPU implementation of several computer vision algorithms, such as Harris´ corners (Harris & Stephens 1988) detection, median, bilateral and Sobel filters or gradient computation. Kayombya (2010) adapted a desktop GPU SIFT implementation to OpenGL ES 2.0 and evaluated its performance. Both works conclude that OpenGL ES 2.0 is a suitable API for the acceleration of computer vision algorithms, recommending general techniques such as the use of calculations per vertex, instead of per fragment or the pre-packing of the images as suitable textures. Although an increased energy efficiency is mentioned, no specific values are shown.

The first attempt to bring mobile GPGPU computing to face detection was made by Wang *et al.* (2010). In their work, based on a mobile GPU-accelerated FFT, they

analyze along with the possible performance gains, the reduction of the energy consumed by GPU-based applications (Wang & Donyanavard 2010). Their conclusions stress the importance of the efficient partition of the software between GPU and CPU in a hybrid solution.

It has to be noted that all these works utilized developer tools and were never integrated in real mobile devices. Following the ideas of accelerating face and object detection and recognition, in early 2011, the author of this thesis presented an LBP implementation of the LBP operator **(Bordallo López *et al.* 2011c)**, able to run on a real device, the Nokia N900. The work showed that along with a moderate speedup obtained by concurrently using the CPU and GPU, the reduction of the energy consumed could be directly translated into energy savings in real devices.

Since then, the emergence of several devices including programmable GPUs, implied the materialization of applications deployed on real devices. Seo *et al.* (2011) created a mobile visual tracking system that used the GPU to perform rigid 3-D body transformations. Implemented on an Android platform, it could be executed in numerous devices. Cheng & Wang (2011) analyzed the use of the mobile GPUs by analyzing a case of study based on face tracking, emphasizing the energy savings that could be obtained with their approach. Ensor & Hall (2011) re-implemented a Canny Edge detector in real devices.

In 2012, the author of this thesis integrated the existing mobile GPGPU algorithms into a multiframe reconstruction application **(Bordallo López *et al.* 2012b)** and a 3-D virtual display based on face tracking **(Bordallo López *et al.* 2012a)**. The applications demonstrate techniques and strategies to utilize the CPU and the GPU concurrently, even when the GPU is needed for rendering.

Recently, several other algorithms such as SURF (Hofmann *et al.* 2012a) (Hofmann *et al.* 2012b) (Yang & Cheng 2012) or Flocking Boids (Joselli *et al.* 2012) (Joselli *et al.* 2013), have been ported into mobile devices, while others such as the FFT (Wang & Cheng 2012) or the SIFT descriptor (Rister *et al.* 2013) have been re-implemented and optimized with several performance gains obtained by a better use of the mobile architecture.

The slow introduction of the first OpenCL-capable platforms has motivated the apparition of several new implementations. A notable early effort is the emulation of the more generic OpenCL standard on embedded devices. Leskelä *et al.* (2009) created the first OpenCL implementation able to run on a mobile development platform, the OMAP3630, by emulating the general purpose interface using OpenGL ES 2.0.

However, this experimental emulation has not been available for general use in standard devices or developer boards.

General guidelines about the use of OpenCL in mobile devices can be found in the work of Maghazeh *et al.* (2013) and Cheng *et al.* (2013). However, it was Wang *et al.* (2013b) who proposed the first implementation of a computer vision algorithm by using OpenCL in mobile devices, an exemplar-based inpainting algorithm (Wang *et al.* 2014). Other applications using OpenCL are a stereo image rectification system (Park *et al.* 2013), a re-implementation of a hybrid GPU/CPU SIFT detector (Wang *et al.* 2013a), a marker-based virtual reality application (Konrad 2014) and a mobile facial recognition system (El-Mahdy & Elsersy 2014).

The most recent efforts are focused in the implementation of common libraries or structures so they can be used for developers in a transparent manner. Pulli *et al.* (2012) introduced a gpu-capable implementation of the popular OpenCV library that can be used in mobile devices. Although several functions are accelerated when compared to the CPU general version, the developer has no control over the computing device that will execute the functions. Their work also analyzes the use of GPU-based computer vision for real-time applications, by studying the performance in an OpenCV environment. Similar commercial implementations, such as Qualcomm's FastCV, also exist (Qualcomm 2013). In a more generic way, Paralldroid (Blanco 2013) is able to generate C/C++/OpenCL code from standard Android Java code, enabling parallel computing in a simple manner, while the work of Yang *et al.* (2012) can convert OpenCL code into the Android-optimized RenderScript.

5.4 Advantages, shortcomings and opportunities of mobile GPGPU

The architectures of mobile GPUs differ significantly from the well-known desktop GPUs. With mobility in mind, embedded GPUs have been designed to minimize their power consumption and size. To adapt to these characteristics, developers have traditionally taken the approach of restricting the problem size or thinking of mobile GPUs as low-end desktop counterparts. Although this might be practical for certain situations, the reality is that that the best performance is only obtained by tackling the fundamental differences between mobile and desktop GPUs and the particularities and needs of mobile applications.

The fundamental reason for using mobile GPUs for general purpose computing, specifically for computer vision tasks, has its roots in the fact that almost every mobile

device on the market includes one. The integration of programmable stages in mobile GPUs provides for an excellent opportunity to offload the mobile GPU, simultaneously increasing the performance and efficiency of the system.

In this context, the use of a mobile GPU decreases the workload of the application processor and proves to be very useful when some other tasks need to be performed at the same time. Mobile Graphics Processing Units can be treated as an independent entity. Their reduced clock frequency and Energy per Instruction (EPI) can potentially reduce the power needs of image analysis tasks on mobile devices (Akenine-Möller & Strom 2008).

Since mobile GPUs do not have a *shared control code*, they are especially useful as a co-processor to execute certain computationally expensive operations, while keeping the CPU free for sequential tasks. Table 6 shows the different EPIs of CPUs and GPUs in different form factors. In any form factor, the GPU shows its higher energy efficiency

Table 6. EPIs of different CPUs and GPUs in different form factors.

	Form-Factor	Model	EPI (pJ)
CPU	Desktop	Intel i5-3570	6886
GPU	Desktop	nVidia GeForce GTX650	60
CPU	Laptop	AMD C50	935
GPU	Laptop	AMD Radeon 6250	38
CPU	Mobile	ARM Cortex-A8	100
GPU	Mobile	I.T. PowerVR 540	16

5.4.1 Architectural constrains

Mobile GPUs have multiple independent processing units (cores) working on floating point data. The performance of these cores is optimal when all of them are occupied and performing the same operations. Instructions that affect the flow control, looping and branching are thus usually restricted and might cause performance penalties. A good practice consists of eliminating loops by using optimized unrolling vectors to perform the operations, decreasing the number of increments and comparisons.

Due to the high degree of parallelism and the high number of processing units that need to be operational at the same time, memory access is critical to avoid processing downtimes, both in bandwidth and speed. Traditionally, discrete GPUs present in

desktop systems transfer data with the main processor using a peripheral bus. On the other hand, current mobile platforms integrate the memory, the GPU and multicore CPU in the same System on Chip, and reduce costs, power and size by integrating a Unified Memory Architecture (UMA) that allows each processor to access the main memory (Elhassan 2005). However, a very important drawback of this approach is that the GPU needs to share memory access bandwidth with other system parts, such as the camera, network and display, subsequently reducing the amount of dedicated memory bandwidth for graphics rendering (McCaffey 2012). Figure 30 depicts the differences between a UMA and bus-based memory transfers.

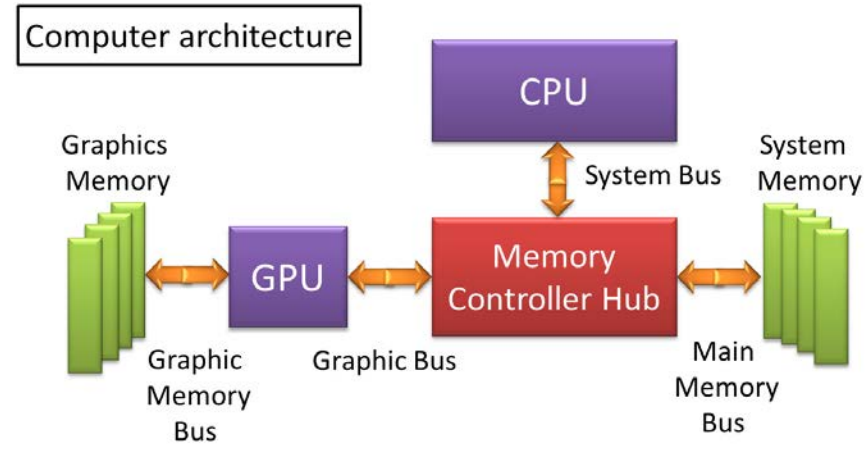
In the context of bandwidth scarcity, mobile GPU architectures have traditionally been implemented as tiled rendering architectures. Texture compression is used to help in the reduction of the memory transfer overheads. When textures cannot be compressed, lower precision pixel formats are used. Lowering the bus traffic between the GPU and the memory is also an efficient way of reducing power consumption. Therefore, several high-level algorithms for bandwidth reduction can be found in the literature (Akenine-Möller & Strom 2008). A good practice consists of exploiting the available APIs to overlap memory transfers.

5.4.2 Numerical representation

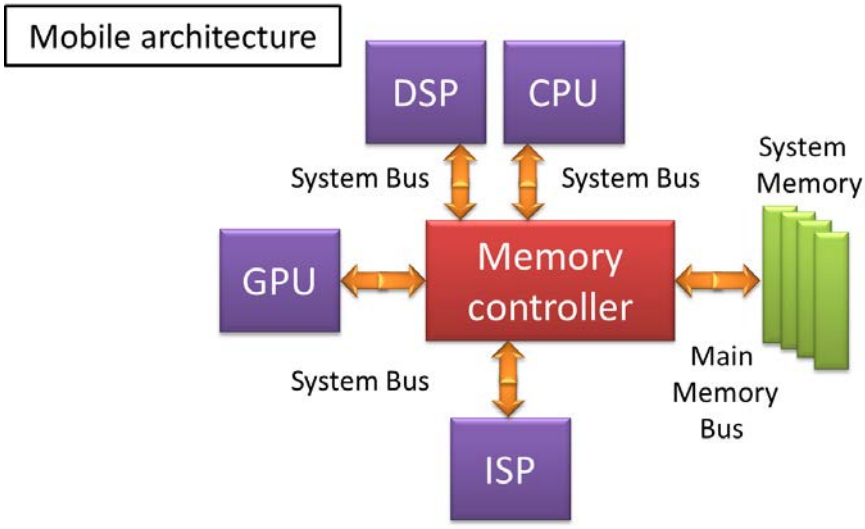
Mobile GPUs are designed to operate in floating-point format. As a result, any image processing algorithm with high floating point arithmetic requirements is a very good candidate for acceleration on a mobile GPU. Operating in other number formats such as integers does not usually have any performance gain, and sometimes it can be treated as a penalty. As example, a 5 x 5 Gaussian filter implemented on a PowerVR SGX540 shows about a 30x speedup in comparison with a floating-point implementation on a ARM Cortex A8 CPU. When compared with a fixed-point CPU implementation, the GPU still presents a 2x speedup, while a fixed-point implementation on the mobile GPU does not present any improvement (Singhal *et al.* 2012).

5.4.3 Form-factor constraints

Other challenges of mobile GPGPU programming are related to architectural design decisions, due to their form factor. The small size of mobile GPUs has pushed the manufacturers to reduce the number of registers. While a high-end desktop GPU can



(a) Bus-based memory transfers



(b) Unified Memory Architecture

Fig 30. Different types of GPU access to memory.

have a register bank size of up to 128KB on each one of its 14 multiprocessors (Tesla M2050), a mobile GPU (PowerVR SGX530, Vivante GC2000) has only 2KB in each one of its 4 cores. This fact causes a smaller number of available thread-blocks and an inefficient scheduler, which in turn results in memory transfers not necessarily well hidden.

Embedded GPUs allow a smaller number of instructions per kernel. Although the next generation of GPUs is likely going to increase the instruction memory size, the limited sizes of instruction *caches* might be a performance limit when more tasks are implemented on the mobile GPUs. This causes a practical trade-off between many smaller kernels with high launching overhead and long kernels that might be inefficient because of small L1 GPU caches. The possible solution to this is to develop heterogeneous programs that concurrently use the GPU and CPU while being able to hide the possible transfers between entities. An example of this type of solution can be seen in the implementations of pattern matching and genetic programming in the work of Maghazeh *et al.* (2013)

5.4.4 Application Programming Interfaces limitations

Employing the resources of mobile Graphics Processing Units is most conveniently done using standard APIs. As discussed before, these APIs are divided between Graphics APIs and general purpose APIs. Most of the recent mobile devices include a graphics processor accessible via the OpenGL ES graphics API, while general APIs are limited to newer high-end devices.

The main disadvantage of Graphics APIs is rooted in their specificity. Designed for graphics processing, OpenGL ES does not provide access to certain architecture specific features. Although, on a mobile SoC, the memory is shared between the CPU and GPU, texture objects used as input data (e.g. images) have to be properly wrapped for the graphics core and cannot be directly accessed as a CPU image array. The texture wrapping operation implies a significant overhead that might be the bottleneck of systems that need large memory buffers. For example, the *practical* memory bandwidth measured on a real system, ARM Cortex A8 and PowerVR SGX530, is about 220MB/s from CPU to GPU and about 30MB/s from GPU to CPU. Table 6 shows the asymmetry in the achieved memory bandwidth with different mobile SoCs (Wang & Cheng 2012). This asymmetry is expected since readbacks from texture memory are not a very common operation in graphics processing. Depending on the computational heaviness of the algorithm, this API implementation can be a major performance penalty. Accessing neighborhood color values in a fragment shader commonly results in dependent texture read, which in turn results in a stall until the texture information is retrieved. To avoid dependent texture read, it is generally good practice to pre-compute neighboring texture coordinates in a vertex shader.

Table 7. Practical memory bandwidth using OpenGL ES in different SoCs.

SoC	CPU to GPU	GPU to CPU
OMAP 3530	220 MB/s	30 MB/s
Snapdragon S2	1100 MB/s	240 MB/s
Tegra2	1100 MB/s	302 MB/s

The implementations of graphics APIs are vendor specific. This causes low precision control in the algorithm, since the implementation of a certain shader on one platform might not give the same results on another. For example, OpenGL ES shaders use precision pragmas such as *highp*, *mediump* and *lowp* to provide the compiler with hints on how the variable is used. Lower precisions increase the shader performance, but the developer has no control over how it is going to be used. An illustration of the lack of precision uniformity can be seen when implementing the following shader code:

```
{
   precision highp float;
   uniform vec2 resolution;
   void main( void )
  {
     float y = ( gl_FragCoord.y / resolution.y ) * 26.0;
     float x = 1.0 "" ( gl_FragCoord.x / resolution.x );
     float b = fract( pow( 2.0, floor(y) ) + x );
     if(fract(y) >= 0.9)
         b = 0.0;
     gl_FragColor = vec4(b, b, b, 1.0 );
  }
```

The results are depicted in Figure 31. The images show a varying fade level from bright to dark over several iterations. The bars are composed using increased floating-point precision further up in the image. It can be seen that different platforms implement the shaders in a different way, with different accuracy. A detailed experiment on floating point accuracy on mobile GPUs can be seen in the articles written by Olson (2013) or Rusell (2013).

General APIs, such as OpenCL Embedded Profile, offer higher flexibility and better access to the architectural features of the mobile GPUs. However, the details of the

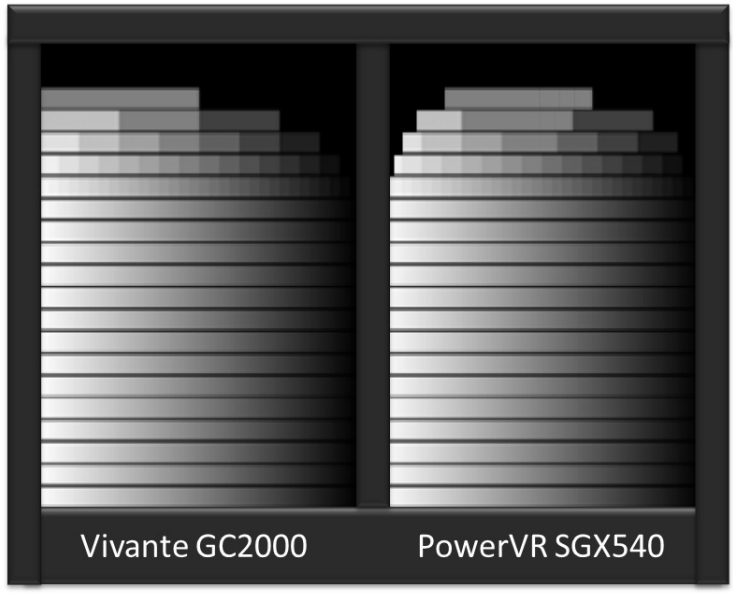


Fig 31. A fade shader implementation to test mobile GPU precision.

standard API implementation might not be transparent to the developer. Standard APIs hide several implementation details from the user by implementing certain abstractions that might have a reflection on the real hardware or might be just emulated. For example, OpenCL 1.1 forces the implementation of software support for local memory. However, current mobile GPUs such as the Vivante or Adreno family do not have on-chip user defined memory that allows data sharing between GPU cores. The result is that the standard is implemented emulating the local memory by using general memory. While the code that uses local memory is still fully compatible, there is a high performance penalty that makes the implementations that use local memory considerably slower. This fact is counter-intuitive as developers expect performance benefits from a memory type that physically does not exist. Future devices are expected to include more and more hardware resources, but for the time being, understanding the actual GPU architecture can result in important performance gains.

5.4.5 Sensor interfacing

An important fact not directly related to the mobile GPU architectures is that the most of the recent mobile devices have not yet taken into account the use of GPU for

camera-based processing. In current mobile architectures, image processing algorithms that use the camera as the main source of data lack fast ways of data transferring between processing units and capturing or saving devices. In this context, to map the algorithms properly on the GPU, the data should be copied to the main memory, then to be later wrapped in the specific model of the graphics APIs. The multiple memory copies result in a latency overhead and the involvement of the CPU in a process that essentially should take place between the camera and the GPU.

5.5 GPU acceleration of multiframe reconstruction

To illustrate the possible performance gains in interactive camera applications when using mobile GPGPU computing, several algorithms that can be used in a multiframe reconstruction application have been implemented using a PowerVR530 mobile GPU, integrated on a Nokia N9 device. The Nokia N9 graphics processor is accessible via the OpenGL ES application programming interface (API). However, as discussed before, the use of GPU as general purpose capable processors has not been extensively considered yet on mobile phones.

In the target platform, the camera image transfers must be done by copying the images obtained by the camera from the CPU memory space to the GPU memory space in a matching format **(Bordallo López *et al.* 2011c)**. Furthermore, the overheads of copying images as textures to graphics memory result in significant slowdowns. The lack of shared video memory causes multiple accesses to the GPU memory to retrieve the data for the processing engine.

Traditional approaches to mosaic building algorithms used to follow a sequential path with multiple accesses to the memory from the processing unit. In the example application, each step of the mosaicking algorithm has to be evaluated separately in order to find the best ways of organizing the data and to reduce the overheads. In order to evaluate the improvements in terms of speed and energy-efficiency, several GPU-implemented relevant parts of the algorithm have been integrated into the multiframe reconstruction application. Both the highly interactive capture stage, the frame selection stage and the blending stage are suitable for acceleration.

5.5.1 Interactive capture acceleration

In the interactive capture stage, the mobile-GPU Harris' Corners detector can be used to accelerate the motion estimation process (Singhal *et al.* 2010). The experiments suggest that the GPU can compute the corner detection of QVGA pictures in less than 50ms, which implies a 30% gain while using a CPU/GPU hybrid approach.

Area-based image registration methods are also suitable for being highly parallelized. For example, the method by Vandewalle *et al.* (2006) uses Tukey window filtering and FFT-based phase correlation computations to register two images. Experiments run on an OMAP 3630 platform show that window filtering and complex division routines increase their execution speed up to three times compared to the CPU when performed on the built-in GPU **(Bordallo López *et al.* 2012b)** while the FFT can be accelerated about two fold (Wang *et al.* 2010).

5.5.2 Accelerating quality assessment

The selection of the best images that will be utilized in the blending stage requires the measurement of the blur contained in each individual frame. The programmable pipeline of OpenGL ES 2.0 enables shader programming in implementing blur detection in a similar way to the feature extraction method. The first stage of the blur detection is a simple derivation algorithm, which can be implemented efficiently with OpenGL ES 2.0 shader **(Bordallo López *et al.* 2012b)**. The designed tests show that on an OMAP 3630 platform the derivation algorithm with HD-720p images can be computed about three times faster on the GPU, while reducing the CPU load by an %.

The most obvious operations to be accelerated using OpenGL ES are pixel-wise operations and geometrical transformations such as warps and interpolations. The stitching process requires the correction of each selected frame with a warping function that must interpolate the pixel data to the coordinates of the new frame. This costly process can be done in a straightforward manner in several steps using any fixed or programmable graphics pipeline **(Bordallo López *et al.* 2009)**. Figure 32 depicts the warping correction process using the vertex shader.

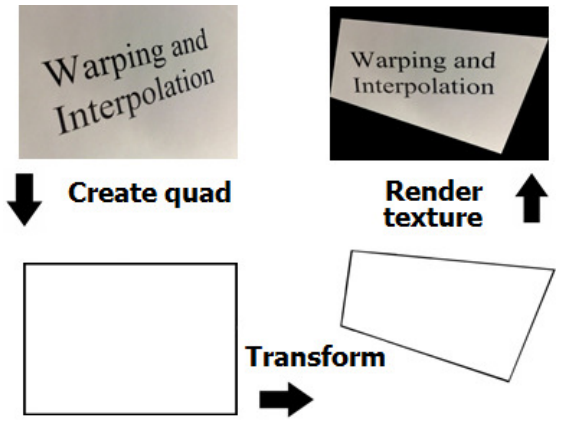


Fig 32. Image dewarping on a mobile GPU.

5.5.3 Accelerating registration and blending

The blending stage requires a re-registration of each one of the selected images and a seamless blending algorithm. Feature extraction can be moved to the GPU through the use of programmable shaders. Previous works shows that desktop-GPU SIFT feature extraction used along with a RANSAC estimator in parallel has shown a 50% CPU load reduction (Sinha *et al.* 2006) and that feature extraction times on VGA frames can be reduced by about ten times (Ready & Taylor 2007). However, the recent CPU/GPU hybrid implementation of SIFT in a mobile (Rister *et al.* 2013) suggests improvements from 6 to 8-fold.

The pixel blending operation can be done utilizing the hardware-implemented blending function. When the blending function is enabled, overlapping textures will be blended together. The transparency can be determined by choosing a blending factor for every channel of both images and then a blending function. The channel values are multiplied with their respective factors. After that, the blending function is applied to each channel pair. Since OpenGL ES 2.0 has a programmable pipeline, blending can also be done with a shader algorithm. In this way, all the needed calculations can be combined in only one rendering stage.

5.5.4 Comparison of performance

As described in Chapter 2, the multiframe reconstruction application can be divided into two parts. The first part, an online loop, is asynchronously executed in each video

frame obtained by the camera, and is the main component of the interactive capture. The algorithms that can utilize the mobile GPU to improve the overall performance are grayscale conversion, scaling, blur detection and motion estimation, using either a Harris' Corners detector or a Phase Correlation method. The improvements in this stage are translated into a higher operating framerate which in time allows the capture of better source images.

The second part of the algorithm, more computationally expensive, performs the accurate registration and blending of selected HD-720p video frames. The algorithms suitable for acceleration are SIFT detection, image warping correction and Gaussian or linear blending.

The computation times and energy consumptions of the main parts of the multiframe reconstruction algorithm are depicted in Table 8. The first two columns refer to pure CPU code, while the other two utilize the CPU and the GPU concurrently.

Table 8. Computational and energy costs per HD-720p frame of several operations implemented on a mobile platform (OMAP 3630).

	CPU time consumption [ms]	GPU time consumption [ms]	CPU energy consumption [mJ]	GPU energy consumption [mJ]
Grayscale conv.	18	8	3,6	1,0
Scaling	24	12	5,3	1,5
Harris Corners detector	60	45	13,5	10,2
Tukey windowing	35	15	5,1	2,1
FFT	70	40	10,2	5,6
Blur detection	80	60	28,2	8,0
TOTAL online (Feature)	182	125	41,1	19,7
TOTAL online (Area based)	207	135	42,5	18,2
SIFT detection	1400	240	770	26,4
Matching	2000	-	1100	-
RANSAC	1600	-	880	-
Image warping	320	105	215	11,6
Image blending	250	105	170	11,6
TOTAL Offline	5470	4050	3235	2030

The results show that the utilization of the mobile GPU increases the performance of the overall application. The online loop can increase the framerate about 4-fold, while consuming 17% of the energy per frame. The GPU accelerated loop is a more energy-efficient solution due to a decreased CPU load and the higher EPI of the mobile GPU. The solution allows increasing the framerate and reducing the power consumption simultaneously.

The GPU implementation of the offline phase is able to accelerate the feature detection, warping and matching to reduce the total computation time by about 20%. The total energy reduction accounts for 30%. In this case, the CPU load cannot get substantially reduced since it is needed for matching and RANSAC.

It can be noted that SIFT detection, matching and RANSAC are the most computationally demanding parts of the multiframe reconstruction application, determining a practical application bottleneck. It would be beneficial to move part of the computations to the GPU by defining a matching algorithm with a high degree of parallelism. Also, the substitution of the computationally demanding SIFT points for faster features, such as SURF, must be considered. Finally, the careful pipelining and scheduling of the application might be able to reduce the idle time of the CPU and GPU, increasing the overall performance.

5.6 GPU-accelerated virtual 3-D display

To analyze the possible performance gains of mobile GPGPU computing in vision-based UIs, this section considers the acceleration of a face tracking approach that uses efficient gray-scale invariant texture features and boosting. The solution is based on the Local Binary Pattern (LBP) features and makes use of the GPU on the pre-processing and feature extraction phase to reduce the computation time and power requirements. The LBP operator is a texture analysis tool that measures local image contrast where the selected pixel's value is defined by its eight surrounding neighbors. The LBP techniques have been identified as a potential methodological basis for implementations due to their high accuracy (Hadid *et al.* 2004). However, the algorithms are pixel-wise and bit-oriented, and as such, difficult to implement efficiently on serial processors. The face-tracking solution integrates the first mobile GPU implementation of LBP extraction. It uses the OpenGL ES 2.0 shading language and its performance is measured on an OMAP3430 platform (**Bordallo López *et al.* 2011c**).

5.6.1 GPU-accelerated face tracking

An efficient face tracking implementation can also be used as a first step for constructing several more complex applications, such as smile-based shutters or virtual 3-D displays. An example scheme of a face tracking algorithm is shown in Figure 33. The face tracking approach under consideration uses efficient gray-scale invariant texture features (Ojala *et al.* 1996) and boosting (Freund & Schapire 1995). Feature extraction is an important part of these systems and many algorithms have been proposed to solve it. The analyzed solution is based on the LBP texture methodology (Ahonen *et al.* 2006) for facial representation.

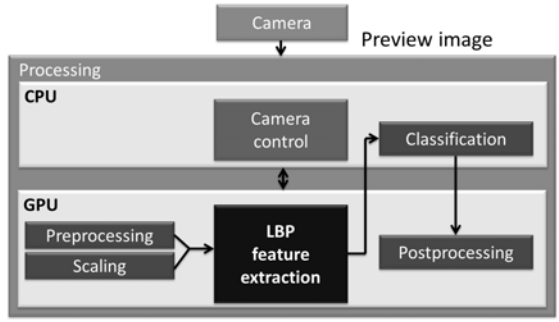


Fig 33. Parallelized face tracker. Tasks are distributed between the CPU and GPU.

After the effective extraction of the face features, a learning system like AdaBoost can be applied to find the most discriminative features for distinguishing the face patterns from the background. This boosting method searches for the faces in the images or image sequences, and returns the coordinates of the detected objects. The resulting information could also be directly used by face-based applications, such as auto focusing or color enhancement.

The face tracking algorithm consists of four phases. First, the incoming viewfinder image is pre-processed. The pre-processing algorithm consists on the multi-scaling of the source image at different sizes, and the preparation of the data in the most suitable format. The extraction of features in different scales allows the detection of multiple-sized objects.

After the pre-processing, the image features are extracted using the LBP operator. Then, classification is performed, and finally the results are post-processed and presented. In this section we describe how the computation of scaling, pre-processing and LBP feature extraction can be implemented on a GPU. The classifier is implemented as a

pure CPU solution, although future work should consider also the GPU acceleration of this computationally demanding phase.

OpenGL ES implementation of the LBP

A detailed implementation of the LBP algorithm on a mobile GPU can be found in a previously published article **(Bordallo López *et al.* 2011c)**[1] The simplest way of implementing LBP on a mobile GPU takes in a basic 8 bits per pixel (bpp) intensity picture. However, a more efficient way consists on taking in a 32bpp RGBA picture. Even if the input picture has only one channel, this approach will offer a better performance since the texture lookup function will always return values of all the four channels.

The 32bpp RGBA texture can be composed in various ways. For example, when low end-to-end latency is not needed, a different gray-scale picture can be uploaded onto each one of the RGBA texture channels. However, since the LBP algorithm is easily separable, another solution is to divide a regular intensity picture into four sections that would be assigned to different color channels.

The required preparation for the input images can be also included at the same stage as the gray-scale conversion and scaling. The uploaded texture can be divided into four sections that can be used to texture each one of the RGBA channels of the rendered quad. The result is suitable as the input of the LBP computation. Figure 34 depicts the preprocessing algorithm.

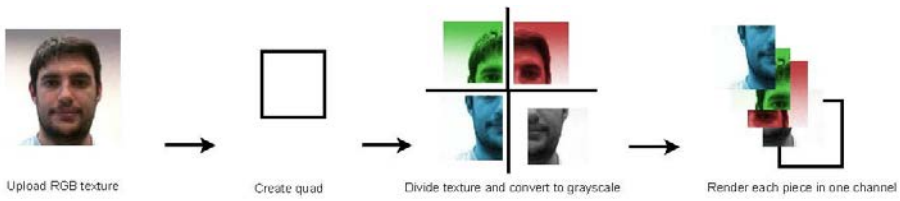


Fig 34. Composition of an RGBA texture.

Similarly to the desktop's OpenGL model, a mobile GPU pipeline is composed of vertex and fragment shaders. The vertex shader operates on vertices and, properly designed, it can be used to transform the coordinates of a quad through matrix multi-

[1]The source code can be downloaded from the website:
http://www.ee.oulu.fi/~miguelbl/LBP-Software/index.html

plications. Depending on the application, these operations can be used to compose a multi-scaled image or to just pass the texture coordinates forward.

The fragment shader operates on fragments (pixels) and it can be used to perform operations such as the LBP calculations. After this, the data goes through various per-fragment operations before reaching the frame buffer. While the quad is textured, bilinear interpolations for each pixel are calculated in parallel on the GPU. The rendering surface is then copied back to the main memory as a native bitmap.

The fragment shader program accesses the input picture via texture lookups. Since this model accepts the use of non-integer indexes, the interpolated version of the LBP that makes use of the built-in interpolation capabilities is as fast as the non-interpolated.

A straightforward solution to calculate the LBP values, in a similar way to desktop GPUs (Zolynski *et al.* 2008), is to form the LBP value by multiplying the binary number's bits with their corresponding weight factors and then sum all products together.

The first operation fetches the selected pixel's value and the second it's neighbors' values. Next, the built-in OpenGL ES 2.0 function step returns a matrix of ones and zeros corresponding to the relations of the pixels' values. The LBP values of all the channels can then be calculated by multiplying the binary matrix with the weight factor vector.

5.6.2 Performance evaluation

Both the standard and interpolated versions of the LBP algorithm were tested on a PowerVR SGX530 mobile GPU. The OpenGL shaders were integrated on a native program programmed using standard C code, running under an embedded linux environment. The implementation was tested with multiple image sizes in order to identify dependencies from the cache efficiency and the level of parallelization, but no significant differences were found. The experiments show that the OMAP3530 built-in GPU is able to compute the LBP of VGA frames in around 40ms. Although the GPU is slower than the CPU on a platform level, an improved performance can be achieved if both are utilized concurrently.

Since GPUs are usually designed with smaller clock frequency than General Purpose Processors, the specialization of its units leads to a smaller EPI. Our experiments show that the PowerVR SGX530 mobile GPU included on the Beagleboard kit consumes about 110mW with the overheads of the memory accesses. Texas Instruments' OMAP3530 Power Estimation Spreadsheet (Texas-Instruments 2011) reports a consumption of about

93mW at a 110MHz frequency which is consistent with our measurements. These values imply a power consumption of about 0.85mW/MHz when the GPU is operated at the intended operating points. When an application does not have heavy time constrains or real-time requirements, the mobile GPU proves to be a good alternative to reduce the total battery drain.

The face tracking process requires the construction of a multi scale 8bpp image that allows the detection of faces in multiple sizes. This costly process can be done in a straightforward manner on several steps using any fixed or programmable graphics pipeline. The process can be mapped onto the GPU, offering an improvement in processing time and energy consumed. Table 9 depicts the computation times and energy consumptions of the relevant algorithms when executed on an N900mobile device.

Table 9. Computational and energy costs per VGA frame of several operations implemented on a mobile platform (OMAP3430).

	CPU time [ms]	GPU time [ms]	CPU ener. [mJ]	GPU ener. [mJ]
8bpp conv.	10	4	6,1	0,5
Scaling	15	6	9,2	0,8
LBP	22	40	13,1	6,1
TOTAL	47	50	28,4	7,4

The test results show that the CPU implementation of the LBP outperforms the GPU implementation. This is due to the fact that mobile GPUs and Graphics APIs are not too suitable for bitwise operations. However, it can be observed that pixel-wise algorithms that require floating-point operations are around three times faster using the GPU.

The energy efficiency of the GPU depends heavily on the algorithm type and mapping, although the smaller EPI of mobile GPUs are usually designed to offer a high energy efficiency per instruction (Akenine-Möller & Strom 2008). However, this feature takes most relevance in algorithms where only floating-point per pixel operations are required.

Measuring the combination of all the algorithms in a single stage produces comparable times in both the GPU and CPU. However, the use of the GPU solution moderately

reduces the energy per frame, while leaving the CPU idle to perform other tasks, such as classification.

When time and energy consumption results are normalized, it can be seen that the lower frequency of the mobile GPU results in a smaller number of cycles consumed per pixel, even in the least favourable of the operations, the LBP extraction. This lower count, together with the smaller EPI, results in an even more pronounced advantage in terms of energy consumed per pixel. Table 10 shows the normalized values, independent of the pixel count and operating frequency. It can be seen that even in the least suitable cases, the GPU acts as a way of trading speed for energy.

Table 10. Normalized computational and energy costs.

	CPU *CPP*	GPU *CPP*	Rate CPU/GPU	CPU *nJPP*	GPU *nJPP*	Rate CPU/GPU
8bpp conv.	19,5	1,4	13,6x	19,8	1,6	12,2x
Scaling	29,3	2,1	13,6x	29,9	2,6	11,5x
LBP	42,9	14,3	3,0x	42,6	19,8	2,1x
TOTAL	91,8	17,9	5,12x	92,4	24,1	3,8x

5.6.3 Concurrent use of CPU and GPU

A practical challenge of using the mobile GPU to perform general computations on a user interface is that the GPU is actually needed to render the 3-D graphics on the screen. An excessive use of the GPU might cause delays in the rendering that would show as unrealistic glitches to the user.

In this context, a straightforward solution is to execute the main tasks sequentially, starting with the camera image acquisition, following with face-tracking and finishing with the render on the screen. However, this causes a very pronounced decrease of the application framerate. A possible solution that increases the overall frame rate of the application consists of the careful scheduling of the application to minimize the idle time of the CPU and GPU. Figure 35 depicts the distribution of tasks between the CPU and GPU to reduce the idle times.

Pipelining the tasks properly, the CPU can start the face-tracking task while the GPU is still rendering the results of the previous frame. The application is implemented by

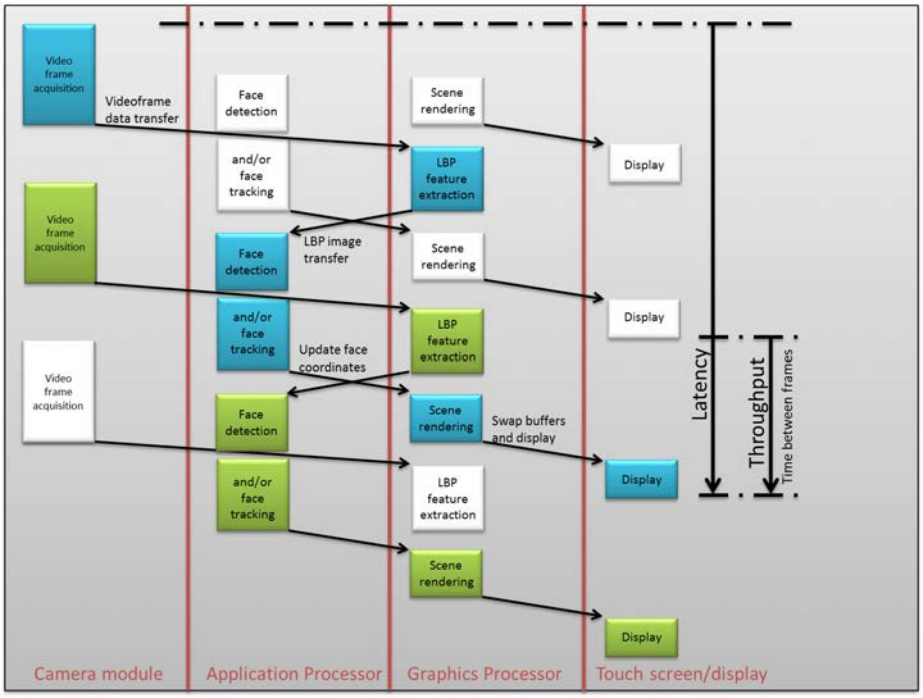


Fig 35. Scheduling of the virtual 3-D UI tasks between the mobile CPU and GPU. The latency of the presentation time remains the same while the throughput and framerate increase.

using three threads. One of them does the LBP feature extraction, the second one does face tracking, and the third one renders the resulting POV onto the screen. While the latency between the capture and the presentation time remains the same, this approach notably increases the frame rate, improving the overall feeling since it allows the rendering of more intermediate frames.

5.7 Discussion

The main contribution of this chapter is the identification of missing and unsupported abstractions in the current mobile graphics processing units APIs and toolchains, achieved through the implementation of several compute-intensive operations on a mobile device. This novel insight into high-performance mobile GP-GPU brings vision-based interactive computing to other developers, showing the limitations and opportunities of the platform.

The high demand for complex graphics applications such as high-end mobile video-games suggests that mobile GPUs will still see future improvements. Consequently, based on the success shown by GPGPU-capable GPUs in desktop environments, the industry will likely include APIs and interfaces to utilize mobile GPUs for the acceleration of intensive floating-point computations. If carefully designed, a single mobile GPU chip could be utilized for both graphics and massively parallel computing. In addition, the use of mobile GPUs in a suitable manner could also have a positive impact in the energy-efficiency of the system.

However, to take advantage of the exploitation of the GPU resources, an understanding of the mobile GPU architecture is paramount. The partition of the application tasks between the CPU and GPU is of great importance. This is in line with the principles of asymmetric processing described in Chapter 4. The most suitable processors should be utilized for each task and they must be used concurrently whenever possible, to increase the maximum throughput.

At this point, it is tempting to believe that *ideal* solutions that have learnt from the experiences will come in a more efficient, elegant and simple manner. However, compromises still exist. Focused on increasing application throughputs, the current utilization of mobile GPUs still presents **problems** of communication **latencies**. These are caused mainly by the specific design and the implementation of the standard APIs. New mobile architectures need to be designed with reduced latency in mind, especially when utilizing camera data. The integration of mobile GPUs in the camera pipeline is the next important step for mobile interactivity (Hakanen 2014).

6 Sensing-assisted vision-based interactivity

"If mobile devices remain unaware of important aspects of the user's context, then the devices cannot adapt the interaction to suit the current task or situation."

- Hinckley *et al.* (2000)

Chapter 5 gave insight on how Graphics Processing Units could improve the energy-efficiency of vision-based interactive applications, while at the same time improving the computational throughput. However, the low-latency and real-time constraints of interactive applications require the consideration of the sensing platform in conjunction with the computing needs of interactive applications.

This chapter addresses the fundamental limitation of using cameras as the only input modality and investigates the use of complementary sensors to enrich the interactivity of vision-based applications, reducing startup latencies and increasing robustness. In addition, the architectural constrains of continuous sensing are addressed, and the use of a dedicated architecture for sensor processors that independently carry out the most of the necessary analytic processing is proposed and evaluated.

6.1 Related work

As described in Chapters 2 and 3, cameras have traditionally been utilized in user interface research to build systems that observe and automatically sense and perceive the human users. However, despite the significant progress made, vision-based interfaces often require customized hardware. They work optimally in more or less restricted environments such as in video game systems (Kinect, Sony EyeToy), where the sensors are stationary and the background is stable. To cope with interactivity needs without the use of a camera, several commercial products integrate sensor processing and interactivity. The Nintendo Wii gaming console is an example of fitting together application interactivity and sensor functionality: the limitations of the three-axis accelerometer are cleverly hidden from the user by the characteristics of each game.

Although to some extent more challenging, the interaction needs of hand-held communication devices in mobile usage are similar. It has been shown that different sensors provide viable alternatives to conventional interaction in portable devices. For

example, tilting interfaces can be implemented with gyroscopes (Rekimoto 1996) and accelerometers (Hinckley *et al.* 2000). Using both tilt and buttons, the device itself is used as input for navigating menus and maps. During operation, only one hand is required for manipulation. A recent and generally interesting direction for mobile interaction is to combine information from several different sensors. For example, a technique to couple wide area, absolute, and low resolution global data from a GPS receiver with local tracking using feature-based motion estimation was presented by DiVerdi & Höllerer (2007).

Ego-motion obtained from sensor data has also been applied to advanced indirect interaction, such as sign and movement recognition. This increases the flexibility of the control system as the abstract signs can be used to represent any command, such as controls for a music player. A number of authors have examined the possibility of using phone motion to draw alpha-numeric characters. Liu *et al.* (2005) show examples of Latin and Chinese characters drawn using the ego-motion of a mobile device, although these characters are not recognized or used for control. Kratz & Ballagas (2007) propose using a simple set of motions to interact with the external environment through the mobile device. In their case, there are four symbols, consisting of a three-sided square in four different orientations. Due to the small size of the symbol set, they report good performance with no user training.

The data obtained by inertial sensor data such as accelerometers and gyroscopes has also been used to recognize the context surrounding the user. Collectively, the set of built-in sensors is enabling new applications in several domains such as transportation (Thiagarajan *et al.* 2009), healthcare (Consolvo *et al.* 2008), or monitoring (Mun *et al.* 2009). The availability of cheap embedded sensors that are included to enhance the user experience (e.g. the accelerometer included to change screen orientation) also motivated the apparition of novel applications that are able to recognize the context. For example, sensor data processing has also been used to recognize human activity with the use of accelerometers (Kawahara *et al.* 2007) or multiple sensors (Gellersen *et al.* 2002). In recent years, we have also seen the irruption of mobile applications (Siirtola & Röning 2012) that have been able to overcome the previous challenges of the task (Randell & Muller 2000).

To classify and summarize context recognition on mobile devices using sensor-processing, Korpipää & Mäntyjärvi (2003) presented an ontology for sensor-based context awareness. A survey of mobile sensing can be found in the work of Lane *et al.* (2010). Their work emphasizes the challenges of mobile sensing, relating them to

technical barriers such as the lack of support of continuous sensing and the difficulty of performing privacy-sensitive and resource-sensitive reasoning with noisy data.

The numerous advances in sensor processing and interactivity have also found their way to commercial platforms. For example, Apple's products make use of the multimodal user interaction technology in different ways. In the iPhone (Apple 2014), users are allowed to zoom in and out by performing multiple finger's gestures on the touch screen. In addition, a proximity sensor shuts off the display in certain situations to save battery power, and an accelerometer senses the orientation of the phone and changes the screen accordingly. Certain Nokia devices turn the screen on and off using the camera and the proximity sensor when the device is inside or outside of the pocket. The Android platform includes, in its latest versions, a way of accessing fused sensors to obtain location and orientation in real time, ready to be included in interactive applications.

However, the information from motion sensors alone might not be sufficient for certain applications. In this context, the need of more accurate data and increased robustness has made camera interactivity the latest target of sensor fusion. Numerous schemes to improve camera interaction and applications with the use of sensors appear in the literature. In their feasibility study, Hwang *et al.* (2006) combined forward and backward movement, and rotation around the Y axis data from camera-based motion tracking, and tilts about the X and Z axis from the 3-axis accelerometer. Labrie & Hebert (2007) utilized accelerometer data to estimate the 3D translations between frames of a video sequence. Clipp *et al.* (2008) utilized a sensor fusion scheme with motion sensors and GPS to assist the reconstruction of large urban scenes using mobile devices. Pons-Moll *et al.* (2010) proposed the stabilization of markerless vision-based human motion detection by using the information of inertial sensors. **Bordallo López *et al.* (2011b)** presented a way of increasing the robustness of camera-based applications, such as panorama capturing and large map browsing, by fusing camera and accelerometer data. Ramachandran *et al.* (2011) improved the results of *Structure from Motion* by using the measurements of the inertial sensors to increase the robustness. Scheuermann *et al.* (2011) tried to mitigate the effects of face rotations due to the ego-motion of the device by integrating the device's analogue information obtained from the motion sensors with the camera data. The previous work has analyzed the incorporation of multiple sensors to increase the interactivity and robustness of vision-based applications. However, the shortcomings and challenges of the implementation are usually not considered and

discussed. There is still the need to consider the platform issues that are brought about by the utilization of cameras and sensors as integral components of the user interfaces.

6.2 Automatic launching of camera applications

In many cases, when a user wants to use a camera application, usually he faces a frustratingly long latency (Kuhmann *et al.* 1987), especially in the case of image captures that require an instantaneous response, such as sudden events or occasional captures of documents. This problem can be traced to the characteristics of the mobile platform and camera subsystems.

An apparently dormant device that is actually on an *energy-efficient responsive sleeping* state (Priyantha *et al.* 2010b), could reduce the starting latency of interactive camera applications and vision-based UIs. The key ideas for the automatic launching of camera applications rest on the utilization of the hand-held nature of the equipment and the user being in the field of view of a camera (Hannuksela *et al.* 2010). In this context, the camera can be used to detect whether the user is watching the device. This is often a good indication of interaction needs. Figure 36 illustrates the user handling the device to launch a camera application.

A key motivation for this concept is to hide the start-up latencies of the camera-based functionalities from the user. In particular, the user could perceive the illusion of camera applications being always on by concealing their often 1-3 second camera application launch delay. In some devices, the high resolution back camera recognizes a screen swipe to act as a trigger for the camera application, while in many other designs, a light push of the camera button powers on the camera. In any case, these events are usually followed by a camera turn-on latency. Figure 37 depicts the benefits of automatic context recognition on a sensing device. The time employed for user actions such as turning on the screen or selecting the desired application and capture mode are reduced by the recognition of the context. The camera configuration latency is also reduced if the camera is always-on.

In a startup latency reduced case, the motion sensors trigger the action recognition sequence that, in turn, continues with the illumination sensor determining the level of ambient light, and if it is sufficient, the cameras are turned on. The frontal camera is used for face detection, while the back camera provides for supplemental contextual information. This can be used to trigger an application such as multi-frame reconstruction or panorama imaging.

Fig 36. Automatic launching of a camera application. When the device is raised in front of the user and a face is in the field of view, the main camera starts the capture. (Bordallo López *et al.* 2012b) ©Springer.

Clearly, the recognition of this context benefits from the coupled use of motion and face sensing, using the frontal camera, provided that it is on all the time, even at a smaller framerate. The motion sensors trigger the action recognition sequence that continues with the illumination sensor determining the level of ambient light, and if it is sufficient, the key lock is released, the back light is turned on, and the back camera is activated automatically. From the user's point of view, it would be most convenient if the device would automatically recognize the type of target that the user is expecting to capture without demanding manual activation of any application. Several targets could be differentiated by, for example, showing a dialog box on the capture screen with the suggested options (Bordallo López *et al.* 2012b).

6.3 The challenges of continuous sensing

The practical challenge of the scenarios described in the previous section is the assumption of having an active front camera and motion sensors. This sort of *continuous sensing* could also enable several novel applications across a number of sectors. The main trade-off is that if the camera and sensors are operated at lower rate, the latency

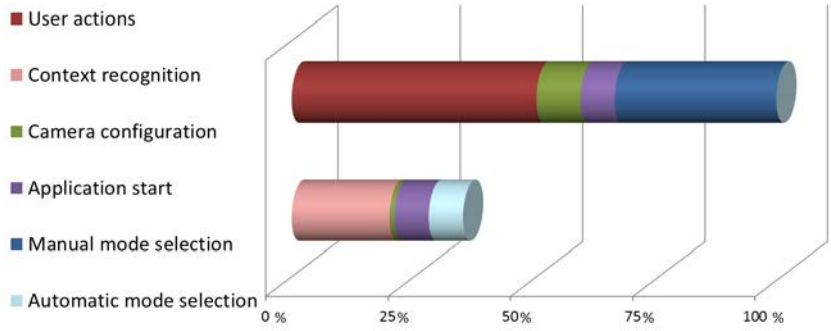


Fig 37. Reduced turn-on latency on a sensing-capable device. The latency originated by the user's needed actions (application start and mode election) is reduced by the automatic recognition of the context.

savings may not materialize, while a higher rate drastically reduces power efficiency. This thesis argues that the roots of the problem are in the involvement of the application processor of the platform in the camera and sensor sampling, and this can be seen as an argument for dedicated sensor-camera subsystems and processors **(Bordallo López *et al.* 2012b) (Bordallo López *et al.* 2011b)**.

As depicted in Chapters 2 and 3, signal processing and computer vision algorithms are often able to utilize all the resources that the mobile platform offers. They are computationally expensive, requiring the CPU to process large volumes of data and many times require real-time operation. As shown in chapter 4, the latency of operation is of vital importance, especially in user initiated operations.

Mobile application processors are usually designed to handle bursts of computations during short periods of time, with the active use of computationally hungry applications by the user. This active-state requires very high use of resources, but it composes a comparatively small part of the total mobile device use. Most of the time, the device will be in a stand-by mode, where the device's application processor is mostly idle, waiting for a user interaction.

However, the stand-by and active-state battery lives of a mobile device are interconnected. High stand-by power consumption means that active use regularly starts with a partially charged battery. As this is a recognized usability issue, the designers optimize for low stand-by currents, primarily by turning off sub-systems such as motion sensors and cameras whenever possible. However, this exposes another usability issue as the

responsiveness to interaction with the device can be compromised. For instance, the device may be unable to detect its handling when in a stand-by state.

To deal with this scenario, several works propose the offloading of the sensor data processing to back-end systems or the cloud (Cuervo *et al.* 2010). The drawback of this approach, which essentially assumes a trade-off of smaller energy consumption in exchange for added latency, is that the energy savings are seldom sufficient for continuous sensing and the reliability and interactivity are seriously diminished.

Future mobile devices should include support for continuous sensing without substantially hindering the current user experience; that means not disrupting the battery life or the responsiveness of the device. Experiments from early tests on real devices show that running continuous sensing applications can reduce the battery life of a stand-by device to 30%, compared with a non-sensing device (Miluzzo *et al.* 2008). To be able to apply continuous sensing in real scenarios with user acceptance, there is a need not only for breakthroughs in energy efficient algorithms, but also in sensing platforms that can keep an acceptable duty cycle on the device while keeping the necessary accuracy, fidelity and low energy consumption (Liang *et al.* 2013).

A straightforward approach to achieve continuous sensing consists of studying the energy-accuracy trade-off while sampling sensor data (Rachuri *et al.* 2010). A possible solution consists of adapting the duty cycle and sampling rate, depending on the activity (Yan *et al.* 2012), increasing the sampling rate when the activity requires rapid responses. However, although the energy consumption reduction with this adaptive method is important, it is still not applicable for an *always-on* solution.

The roots of the limitation are in the involvement of the application processor of the platform, and this can be seen as an argument for dedicated camera and sensor processors. The background is in the typical top level hardware organization of a current mobile communications device with multimedia capability, as the example that is shown in Figure 38. As discussed in chapter 4, most of the application processing functionality, including camera and display interfaces, has been integrated to a single system chip. The baseband and mixed signal processing, such as the power supply and analog sensor control, have their own subsystems. For instance, with the design of Figure 38 the accelerometer measurements cannot be kept on all the time due to the power hungry processor. In comparison, sport watches that include accelerometers, operate at sub-mW power levels, thanks to their very small footprint processors (Epson Inc. 2014).

In practice, the bulk of the sensor processing, including the camera, needs to be moved to dedicated low-power subsystems that can be on all the time. This reduces the

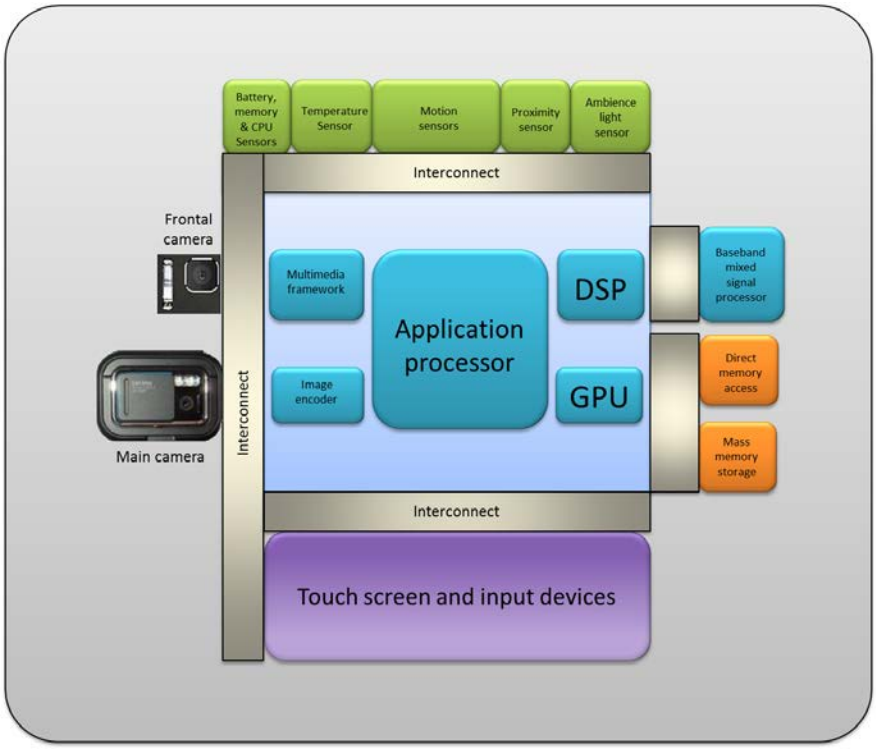


Fig 38. Example organization of a current multimedia device. In current devices, the processing of the motion sensors, frontal and back cameras is mainly done on the application level using a power-hungry main processor.

number of tasks the application processor needs to execute, improving its reactiveness, for example, for highly interactive vision-based user interfaces. In Figure 39, the author proposed **(Bordallo López *et al.* 2009) (Bordallo López *et al.* 2012b)** a possible future design with low power sensor processors. The inclusion of small-footprint processors that operate very close to the camera and sensor units, allows the energy efficiency of the subsystems that can always remain on, enabling new interaction methods.

In this context, several companies in the mobile industry are investigating and prototyping new sensing platforms, with specific processors (Leppänen & Eronen 2011) (Apple Inc. 2013) (Intel Inc. 2014) (CEVA Inc. 2014). Microsoft Research is developing hardware support for continuous sensing (Priyantha *et al.* 2010a) in its LittleRock project (Priyantha *et al.* 2010a). Their prototype features, attached to the battery, a sensor controller and processor (based on a small microcontroller and a DSP) which supports duty cycle management, signal processing and sensor sampling, reducing

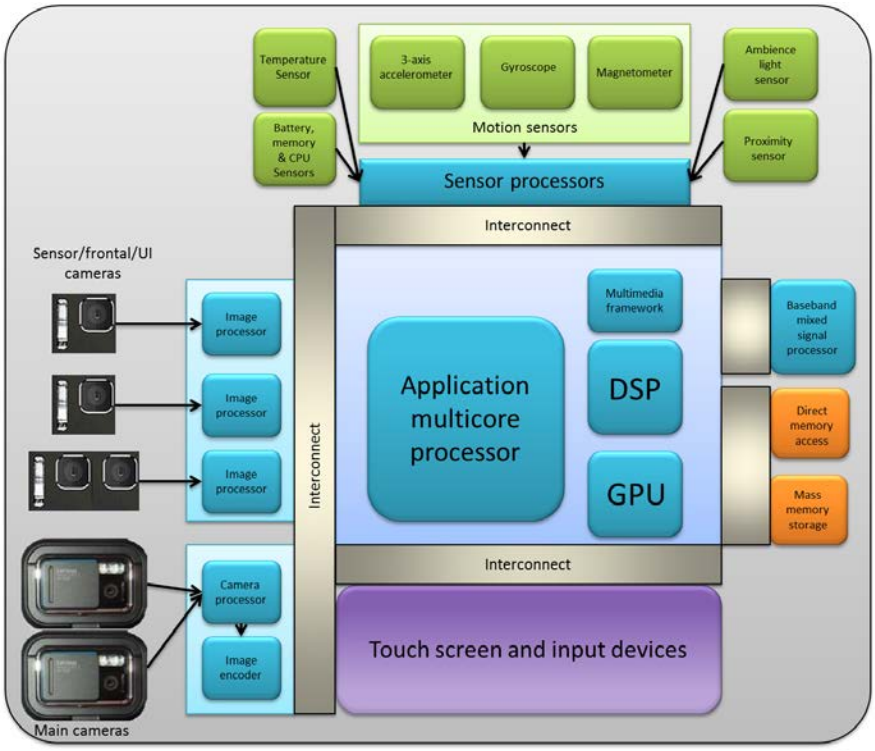


Fig 39. Possible organization of a future multimedia device. The author proposes that future multimedia devices include several dedicated small-footprint processors that minimize the transfers, improving the energy efficiency and allowing being always activated.

the involvement of the CPU and other parts of the application processor (GPU, DSP) to scenarios of high activity (active-state) while supporting continuous sensing in stand-by mode.

Reducing the utilization of the main processor has a huge battery-life impact in sensor data processing, but also in camera-based context recognition. For example face tracking from QVGA (320-by-240) video requires around 10-15 MIPS per frame, using Local Binary Pattern technology (Ahonen *et al.* 2006). Implemented on an ARM7-based camera subsystem, the energy per instruction (EPI) is around 100 pJ; an optimized DSP could implement it with an EPI of 10-20 pJ, while with optimized sensor processing architectures, the EPI can be pushed well below 5 pJ (Nazhandali *et al.* 2005). Consequently, if implemented at a low frame rate of 1 frame/s, the corresponding power needs range from micro-Watts up to 1 mW. Employing the application processor with its

interfaces for the same purposes demands tens of mW, significantly reducing the battery life of *active-state* uses.

Together with the sampling and processing of the data, the designers need to take into account the power needs of the sensors themselves. For example, capacitive touch interfaces demand around 3mW, while a triaxial accelerometer only dissipates in the sub-mW range. Cameras still require lots of power when sampled at a high rate. While a QVGA camera requires about 1mW when operated at a rate of 1 frame/second, an HD frontal camera operated at 15 frames/second can consume more than 60 mW. A possible solution is the inclusion of dedicated cameras that are only usable as *sensing cameras* (Inc. 2014). A small number of cameras could be directly equipped with sensors that are able to compute certain data to assist the visual-context recognition tasks (Lahdenoja *et al.* 2007) (Rodríguez-Vázquez *et al.* 2010).

Another complementary solution to the high energy consumption of the cameras as sensors, consists of the adaptation of the back and front high definition cameras of the current devices. The goal would be to optimize the energy consumption of the sensors to make it proportional to the resolution and the sampling rate. LiKamWa *et al.* (2013) made two proposals that could be applied to current devices. The inclusion of an efficient clock frequency management system on current sensors could reduce the energy consumption of the capture of a low resolution frame from 30% to 50%, while the implementation of a low-power stand-by between frames could reduce it an additional 40%. Their work also proves that 0.1 Mpixel images taken at 3 frames/second could be sufficient for certain context recognition scenarios, such as image registration or object detection. These findings could also have an impact on the sensing and vision ability of future wearable devices (Han & Philipose 2013).

6.4 A sensor and camera data fusion system for vision-based interactivity

The fusion of camera data with the data obtained from the mobile built-in sensors is not necessarily a straightforward process. In this context, finding a way of improving the camera data by increasing its robustness is of vital importance. This section describes a system that determines the motion of the device using both image and accelerometer data. Camera-based motion estimation has a number of apparent performance limitations, caused by lighting conditions and fast movements. Similarly, pure accelerometer measurements lead to errors that increasingly grow with time. These two sensing

modalities can be combined to compensate for each other's limitations and therefore provide for more robust device movement estimates.

In vision-based user interfaces, special attention needs to be paid to the design of a proper lighting system. One possibility is to use special infrared LEDs for illuminating the user's face. Energy consumption is, of course, the main limitation of such designs. Therefore, if cameras become standard user interface components in mobile devices, energy efficiency requires that the bulk of computing is carried out using hardware acceleration.

Another possibility to solve this issue is to use an adaptation method to switch to another input source. Several motion sensors are included in most of the newest mobile devices. For example, linear accelerometers can capture data at a very high rate, increasing the system's robustness when the user input consists of fast movements or the camera image does not present a sufficient amount of valid features.

Combining inertial sensors with camera motion estimation has been an active research area in the past decade. In many cases, the fusion is done using a Kalman filtering (KF) framework (Kalman 1960). For example, Klein & Drummond (2002) and Bleser & Stricker (2009) used a KF-based method for fusing rate gyroscope information with model-based tracking. Jiang *et al.* (2004) proposed a real-time system for outdoor augmented reality integrating gyroscope data and natural line features from images. In the proposed system, KF is used to fuse measurements from accelerometers and camera motion estimation.

The main features of the Kalman filter are modelling the random process under consideration using a system model and recursive processing of the noisy measurement data. A filter is optimal if the dynamic model is linear, the measurement model is linear, and the noise processes involved are Gaussian distributed. Furthermore, the recursive nature of the algorithm makes it convenient to use in real-time systems. A detailed system description can be found in the article by **Bordallo López *et al.* (2011b)**. Figure 40 shows an example result for an image sequence. This information is used in RANSAC style outlier analysis, which provides reliable motion features for 2-D motion estimation.

6.4.1 System model

In sensor fusion, the objective is to recursively estimate the state in the dynamic model. We model the device motion using the following model

$$\mathbf{x}_{k+1} = \Phi_k x_k + \Gamma_k \varepsilon_k, \tag{9}$$

where the parameters to be estimated are presented by the state vector $\mathbf{x}_k$ at time instant k, and Φ_k is the state transition matrix. The state transition matrix relates the state at time instant k to the state at time instant $k+1$. $\Gamma_k \varepsilon_k$ models the uncertainty of the motion model. The process noise ε_k is assumed to be Gaussian distributed with an expected value $E\{\varepsilon_k\} = 0$ and the covariance matrix $Q_k = E\{\varepsilon_k \varepsilon_k^T\}$. Γ_k is the process noise transition matrix.

The state vector x_k consists of the position (x_k, y_k, z_k), velocities $(\dot{x_k}, \dot{y_k}, \dot{z_k})$ and accelerations $(\ddot{x_k}, \ddot{x_k}, \ddot{x_k})$ of the device at time instant k. It is defined as follows

$$\mathbf{x}_k = [x_k, y_k, z_k, \dot{x_k}, \dot{y_k}, \dot{z_k}, \ddot{x_k}, \ddot{y_k}, \ddot{z_k}]^T.$$

In the beginning, the elements of the state vector are set to zero. The time step between two successive images is normalized to 1. We approximate the variances of the process noise from the maximum accelerations allowed.

6.4.2 Measurements

The measurement model is needed to relate the state to the 2-D image motion and accelerometer observations. In our case, the model is defined as

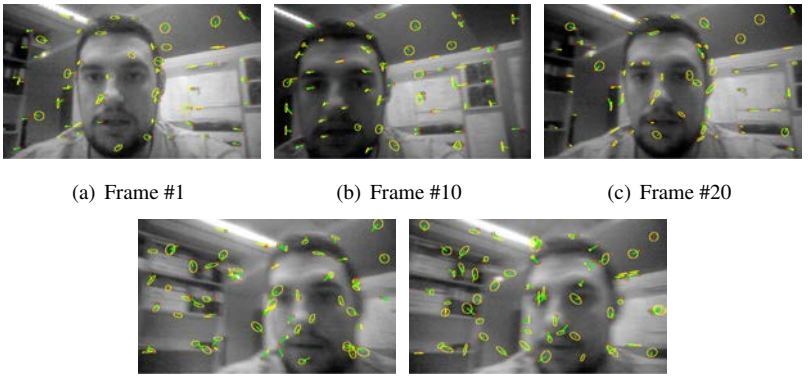
$$\mathbf{w}_k = \mathbf{H}\mathbf{x}_k + \eta_k, \tag{10}$$

where $\mathbf{H}$ is the observation matrix. The measurement noise η_k models uncertainty in the motion measurements and it is assumed to be Gaussian distributed with an expected value $E\{\eta_k\} = 0$ and the covariance matrix $R_k = E\{\eta_k \eta_k^T\}$. The noise covariance can be adjusted based on lighting conditions. For example, in dark lighting conditions the uncertainty of image motion estimation is greater.

In actual measurements, the ego-motion of the device is estimated from 2-D image motion measured between two successive frames. The utilized approach employs a sparse set of feature blocks first selected from one image and then the displacements are determined (Hannuksela *et al.* 2007b) (Sangi *et al.* 2007). The confidence analysis of block matching is of special interest since the results of this process can be utilized in further analysis.

First, the previous frame is split into 16 subregions, and one 8 by 8 pixel block is selected from each region based on analysis of spatial gradients. Displacement the of

selected blocks is estimated using the zero mean sum of squared differences (ZSSD) criterion which is applied over some range of candidate displacements (e.g. 16 pixels). Refinement to subpixel precision is done in the neighbourhood of the displacement, minimizing the ZSSD measure using fitting of the second order polynomials. The ZSSD values are also used for analyzing uncertainty information related to the local displacement estimate.



(a) Frame #1 (b) Frame #10 (c) Frame #20

(d) Frame #30 (e) Frame #40

Fig 40. Example frames from the image sequence. Estimates of feature block displacements (lines) and associated error covariances (ellipses). (Bordallo López *et al.* 2011b) ©SPIE.

In this setup, a four-parameter affine model is sufficient for approximating motion between frames as it can represent 2-D motion consisting of x-y translation (θ_1, θ_2), rotation around the z-axis (ϕ), and scaling s. Scaling s is related to the translation in the z-direction, ΔZ, as $(s-1) = \Delta Z/Z$, where Z is the scene depth.

In the presented example of the system, images are captured at fixed rate of 15 fps and the motion is estimated at the same rate. Therefore, the accelerometer data is acquired also at a rate of 15 fps. This leads to an easy implementation of the measurement fusion. Figure 41 shows example data for x-,y-, and z-acceleration for the same sequence as that presented in Figure 40.

In order to obtain smoother output as a result of motion estimation, Kalman filtering is applied for implementing sensor fusion. The Kalman filter algorithm estimates the motion recursively, repeating two stages: prediction and correction. At the first stage, the state at the next time instant is predicted, based on the previous state estimate and the dynamical model. In the correction stage, the predicted state is adjusted by using the measurements of the image motion and device acceleration.

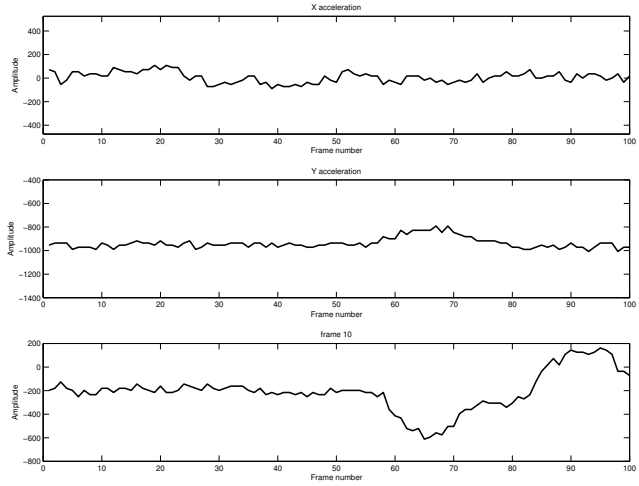


Fig 41. Accelerometer measurements for the example sequence. (Bordallo López *et al.* 2011b) ©SPIE.

The presented framework fuses image analysis with data from motion sensors, improving the user interactivity and reliability of camera-based applications, especially when the environmental conditions are not optimal for approaches using camera data alone. The fusion of the data also decreases the number of operations that are needed for an image-based motion estimation, improving the computational efficiency of the system.

Other than the accelerometer data, the framework proposed allows the integration of data from other types of sensors that describe the device's motion, such as gyroscopes or magnetometers. Other sensors present in the device, such as ambient-light or proximity sensors, can be used to adapt the fusion process, evaluating the conditions and adjusting a proper balance for the contribution of each sensor to the final results.

6.5 Improving multi-frame reconstruction using sensor processing

The previous section describes how to fuse the data obtained from accelerometers and image analysis to enhance the user interactivity and robustness of camera-based applications. The framework allows the balancing of several motion input sensors, improving the device movement estimates, especially when the environmental conditions are not optimal for approaches using camera data alone. This scheme can be utilized

for improving vision-based interactive applications. For example, a camera-based tracking system can be utilized in real-time for browsing large images or documents such as maps on small screens with single hand operation. Based on video analysis, the solution enables the display to be controlled by the motion of the user's hand. The motion sensor obtains the device orientation and can be used as complementary information for motion-based browsing. The system decides the motion input by fusing the measurements from the camera and accelerometers depending on the light conditions and image quality.

The sensor fusion method can be integrated in the multi-frame reconstruction method described in chapter 3. The best use case consists of a multi-frame reconstruction application that acts as a real-time panorama builder that composes a mosaic image from individual video frames using the motion information obtained by both the camera and the device accelerometers. The image registration stage relies on the sensor fusion method to offer an accurate description of the camera movement with increased robustness against poor light conditions. This is practically done with a multimodal motion estimation approach.

To improve the robustness of the system, the ambient-light sensor (or the average luminance obtained from the camera) present on the device determines the lighting conditions of the environment. In case the illumination is insufficient, the night-mode is turned on and the motion estimation model assigns an increased value of trust to the accelerometer measurements, decreasing the value of the features extracted from the camera frames.

In order to determine the quality of each frame, and to add robustness to the selection system, the accelerometer measurements are integrated with the blur in a scoring system. A simple measure of the involuntary tilting or shaking is done by calculating the average of the last motion vectors provided by the accelerometers and subtracting the result from the current vector. The result of this operation is then thresholded to determine if a frame is too blurry and should be discarded. The final selection of the images to be blended is done based on the score of the frame quality, calculated with the values of rotation, change of scale, involuntary tilting and the motion detection process **(Bordallo López *et al.* 2011b)**.

6.6 Improving face-tracking based UIs with sensor integration

As described in chapter 3, a typical problem of face-tracking user interfaces is the lack of robust solutions that are able to handle the case of a tracking loss. A possible solution to this is the integration of data provided by the mobile device's motion sensors into the tracking system, providing for an alternative when the tracking is not possible due to occlusions, fast movement, bad image quality or the user being out of the field of view.

The combination of accelerometers, magnetoscopes and gyroscopes on mobile devices is able to offer the position and ego-motion of the device. This information can be used to correct the input image, reducing the face rotation and improving the face-tracking success rate. For example, Scheuermann *et al.* (2011) report up to 50% improved detection rate of a Viola & Jones (2001) based face tracker when the phone angle with respect to the perpendicular is bigger than 30 degrees. Another possibility consists of using the motion sensors present in the devices to improve the prediction of the face position in the tracking algorithm (Han *et al.* 2012).

The same ego-motion information can be used to track the possible movement of the user face respective to the device, and increase the tracker robustness extending the tracking to a beyond field-of-view solution. An example of this technique can be found in the work of Joshi *et al.* (2012), which describes a camera-gyro fusion system that allows the browsing of large images without the use of the touch screen.

6.7 Discussion

The main contribution of this chapter is the analysis of the relationship between sensing, intrinsically related to latency, and computing, with an impact in the energy consumption on the system. This is achieved through the implementation of several methods to reduce the perceived latencies utilizing sensor integration. In addition the proposal of a sensing architecture provides steps towards a future *always-on* sensing platform.

The described scenarios represent a more general class of applications that employ multimodal sensory information, and may provide for ingredients for novel device designs. Motion estimation, obtained from the fusion of camera data and multiple complementary sensors can be identified as a potential platform level service to be offered via the multimedia API. As it plays key roles in the demonstrated use case applications, it is most likely to be employed in many others.

Sensor-assisted vision-based applications could provide for fast response using the analysis of the context and the prediction of the user intentions. However, the reduction of application and UI start-up latencies can only be tackled with the use of continuous sensing. Undoubtedly, this requires specific architectures that allow the camera and sensors to be always-on. In this context, the future inclusion of specific sensor processors in semi-independent subsystems is paramount.

7 Reconfigurable computing for future vision-capable devices

"The ability to customize the architecture to match the computation and the data flow of the application has demonstrated significant performance benefits compared to general purpose architectures. Computer vision application characteristics have significant overlap with the advantages of reconfigurable architectures. "

- Bondalapati & Prasanna (2002)

The previous chapters exemplified the computing and sensing challenges of vision-based interactive applications and user interfaces and the relationship with the limitations imposed by the embedded nature of mobile devices. The importance of using utilization of all the available platform resources to hide application latencies and maximizing the computational throughput has been shown.

However, current computing and sensing platforms still present some limitations in the scalability for higher resolutions and more complex algorithms, while still maintaining energy-efficiency. Future high-resolution cameras and high-performance applications are likely to require more specific solutions such as dedicated image processors or reconfigurable hardware architectures.

This chapter presents concepts laying on platform level adaptability, exploring the acceleration of vision-based interactive applications through the utilization of three reconfigurable architectures. Based on the analysis of interactive applications and user interfaces, several computationally expensive image processing kernels are implemented using three different reconfigurable architectures. In this context, a processor with a reconfigurable accelerator is proposed as a low-power high-efficiency alternative, or complement to the current ARM processors and NEON units. A hybrid reconfigurable SIMD/MIMD platform is proposed to complement mobile GPUs. Lastly, the inclusion of the flexible Transport-Triggered Architecture-based processors is proposed as a low-power complement to current DSP-based solutions. All three architectures are evaluated and compared with their current counterparts, analyzing their advantages and weaknesses in terms of performance and energy-efficiency when implementing highly interactive vision-based applications.

7.1 Experimental setup

For experimental purposes, to provide a comparison with the selected reconfigurable architectures, we benchmark different computer vision kernels across several current platforms. In this context, we have measured and estimated the performance on two different devices based on the Texas Instruments OMAP3 family, OMAP3430 and OMAP3530. The platforms, a Beagleboard revision C and a Nokia N900 mobile phone, include an ARM Cortex-A8 CPU, a PowerVR SGX530 GPU and a TMS320C64x+ DSP. The OMAP3530 SoC can be set to use at least six different operating points, with frequencies of the main processor ranging from 125 to 720MHz and DSP frequencies up to 520MHz. The chosen operating point features a 600MHz ARM core, with a 430MHz DSP and a 110MHz GPU. For the selected operating point, the single-core ARM processor presents a maximum power consumption of 550mW. The utilization of the NEON coprocessor increases the consumption to 670mW. The mobile GPU consumes about 93mW alone and about 110mW including the overheads of the memory readings. Lastly, the DSP core consumes about 248mW **(Bordallo López *et al.* 2011c)** (Texas-Instruments 2011).

7.2 Reconfigurable architectures

A reconfigurable processor is a processor with erasable hardware that can rewire itself dynamically. This allows the chip to adapt effectively to the programming tasks demanded by the particular software they are interfacing with at any given time. Ideally, a reconfigurable processor can transform itself to run applications across different fields with the highest possible performance.

As discussed in Chapter 4, the scalability of the performance of general purpose processors has been recently declining. Even with transistor densities improving according to Moore's law, the failure of Dennard Scaling (Esmaeilzadeh *et al.* 2011) and the lack of proportional improvements in battery technology will prevent future devices from utilizing the whole die area at the same time. Alternative processors such as the GPU could be used as energy-efficient architecture alternatives. Thus, their use is likely to rapidly increase. However, the current architectures included in mobile devices have noticeable drawbacks either in future scalability or lack of flexibility. In this context, reconfigurable computing has the chance of becoming a future mainstream alternative as a part of the future scalable mobile architectures (Chung *et al.* 2013).

Over the years, numerous reconfigurable architectures have been proposed to fill the gap between the performance of ASICs and the flexibility of General Purpose Processors. Computer Vision algorithms and applications are inherently comprised of very variable of tasks that range from low-level pixel processing to high-level inference of abstract representations (Nieto *et al.* 2011). Reconfigurable computing, oriented in performance, but with specific flexibility in mind, adapts extremely well to this paradigm (Bondalapati & Prasanna 2002).

Several processors aim to meet the processing requirements of camera pipelines without compromising the costs. This is the case of the CRISP stream processor (Coarse-Grained Reconfigurable Image Stream Processor) (Chen & Chien 2008) which outperforms modern DSPs in these kinds of tasks by a factor up to 80.

Other processors focus on the inherent parallelism of image data to enhance the performance of computation-intensive tasks by including SIMD units. The MorphoSys processor (Singh *et al.* 2000) adds a reconfigurable SIMD coprocessor based on a 2-dimensional mesh with enhanced connectivity to a RISC core utilized for control tasks. To exploit task parallelism, other architectures are designed with the focus on the execution of different tasks at the same time (Lanuzza *et al.* 2007) (Uzun *et al.* 2003).

Low-level image processing, inherently data parallel, usually consumes most of the computation time. However, subsequent tasks are also time-consuming, and custom accelerators that allow task parallelism are often a requirement. Hybrid architectures permit facing both processing stages, reducing hardware requirements and taking advantage of the interaction between these stages to improve performance, instead of considering them independently. Embedding FPGAs in modern SoCs composing an heterogeneous system provides for flexibility and high performance (von Sydow *et al.* 2006).

Exploiting both data and task parallelism, heterogeneous reconfigurable architectures such as the HERA processor (Wang & Ziavras 2004) have been developed. Usually composed of two complementary units and a data sharing network, they present some limitations in dataflow control. Some of these limitations can be overcome by the inclusion of a RISC core that executes sequential parts of the algorithm and takes care of control flow (Prengler & Adi 2009).

Since many architectures still require a general purpose or domain specific processor, other reconfigurable architectures focus on assisting a more general counterpart such as a DSP to perform specific tasks in a faster manner (Hung *et al.* 2003).

Utilizing an example architecture of each type, the rest of this chapter analyzes three different styles of reconfigurable architectures, a reconfigurable accelerator for a RISC processor (EnCore), a task/data parallel reconfigurable architecture (Hybrid) and reconfigurable application-specific processors to assist the general processor (TTA).

7.3 EnCore processor with a Configurable Flow Accelerator

The EnCore processor (Almer *et al.* 2009) (PASTA group, University of Edinburgh. 2014) is a configurable 32-bit single-issue RISC core which implements the ARCompact instruction set (Arc International. 2014). The processor can be integrated on a System on Chip, together with an extension interface for reconfigurable accelerators. The specific reconfigurable accelerator of the EnCore architecture is called the Configurable Flow Accelerator (CFA). It defines a small Instruction Set Architecture (ISA) which allows the customization of Application-Specific Instruction-set Processors (ASIP) through the use of user-defined Instruction Set Extensions (ISE).

Figure 42 shows a simplified schematic of the EnCore Castle datapath. Although not shown, the current implementation has a 5-stage pipeline. The Fetch block manages instruction supply. There are two banks of registers. The first, a general-purpose-processor register bank (GPP) is employed for the standard ALU of the CPU. The second register bank stores data for the CFA extension datapath.

In a similar manner as SIMD coprocessors in ARM processors, the EnCore CFA defines an Instruction Set Extension which includes specific operations that can be used to accelerate algorithmic computations. The particularity of the CFA is that it enables the definition of custom instructions that specifically adapt to the algorithm in hand. The instructions include additional arithmetic operations or a combination of them, facilitating the speed-up of the most critical parts of the application.

The inclusion of the CFA entails only a limited increase in hardware resources and power consumption, but usually implies a large increase in performance. This is achieved by making use of several single-function ALUs that allow spatial and temporal parallelism through resource sharing and pipelining. In addition, the CFA is fully programmable, supporting up to 64 reconfigurable extension instructions.

Figure 43 presents a simplified scheme of the CFA unit. A set of ALUs and multiplexers allow the data alignment through shuffling. The CFA is highly configurable and its datapath runs multi-cycle operations. The CFA has a 3-stage pipeline and is able to handle 4 independent arithmetic operations according to the configuration of the

particular ISE under execution. The CFA register bank supplies a vector of 4 elements to the CFA, storing up to 10 of them.

The programmability of the CFA is exploited by the definition of new specific instructions that adapt to the desired algorithm or application. The process of analyzing the applications to identify candidate instructions, can be done in a manual or automatic manner, resulting in a series of templates that adapt to an existing CFA or the generation of a new one. In this context, the EnCore processor employs a design flow for automated construction of ISEs (Almer *et al.* 2009). Figure 44 shows how a custom ISE is mapped in the CFA unit.

Using an adapted compiler to identify and exploit the more suitable ISEs, the resulting CFA mapping can reuse the results across different applications. The resulting ISEs are larger and more complex than the standard RISC instructions. It has to be noted that in order to adapt to the available CFA hardware resources, not all the ISEs identified by the compiler are necessarily matched to the CFA. In practice, there is a trade-off between specific ISEs with low latency and high resource usage and reusable shared instructions with higher instruction latencies. Thus, data allocation becomes critical in the performance maximization process (Zuluaga & Topham 2008).

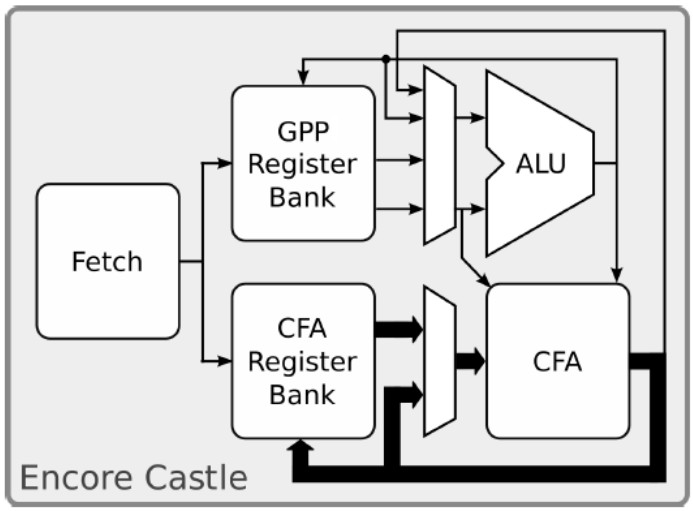


Fig 42. EnCore Processor simplified organization scheme.

7.3.1 Improving vision-applications using an Encore processor

To provide for an example of the possible improvements obtained by using the EnCore processor to execute vision-based interactive applications, the execution of several operations has been measured. The evaluation of the several image processing kernels has been done using two setups, the EnCore processor alone and the Encore processor utilizing a CFA. The setup for the EnCore processor consists on an EnCore Castle chip that has been used to obtain the measurements in terms of performance and energy efficiency. The Castle test-chip, a second iteration of the architecture, is fabricated with a generic 90nm CMOS process, and occupies only 2.25 mm^2, including the CFA and two 32KB caches. Embedded within a SoC providing a generic 32-bit memory interface, the processor features an operation clock-rate of 600MHz, with a maximum power consumption of 70mW on typical conditions. This evaluation of the EnCore processor is partially presented in previous work (Nieto 2012). The results are compared with a mobile CPU (ARM Cortex-A8), with and without the use of a NEON unit. Table 11 presents a summary of the experiments.

The experiments show how the performance of the EnCore processor is comparable to the ARM. However, EnCore, designed with energy-efficiency in mind, consumes much less power. The energy consumption of the EnCore processor represents from 8 to 33% of the ARM consumption, depending on the image processing kernel. However, it has to be noted, that the comparison with newer ARM models (e.g. Cortex A15) could reduce the gap to about the half.

Depending on the operation, the use of the NEON coprocessor increases the performance of the ARM core up to 50% while increasing the total power consumption

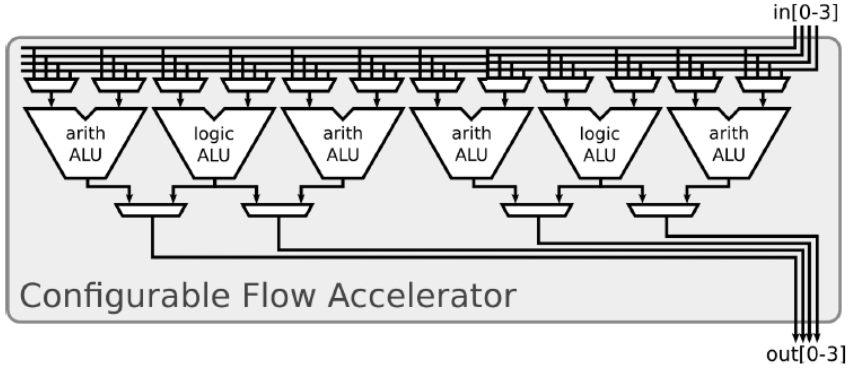


Fig 43. A simplified scheme of the Configurable Flow Accelerator (CFA).

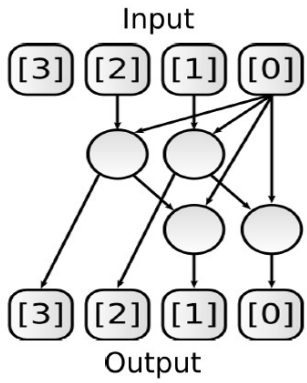


Fig 44. An example of a custom instruction of the Configurable Flow Accelerator. Four input registers and four independent arithmetic operations result in four output registers.

only 20%. The potential of a reconfigurable accelerator can be seen in the comparison of the NEON unit with the CFA of the EnCore processor. The performance of the EnCore processor increases up to 4 times for certain kernels, such as grayscale conversion or alpha blending. The difference is more noticeable in simpler kernels, where the needed arithmetic operations can be mapped directly into a single CFA instruction.

Computationally expensive kernels can benefit from a reconfigurable co-processor. The integrated nature of the CFA unit can be included in the tool-chain in a transparent

Table 11. Cycles per pixel needed by several algorithms in the ARM and EnCore processors including accelerators.

	CPP				nJPP			
Operation	ARM	NEON	EnCo	CFA	ARM	NEON	EnCo	CFA
Grayscale conv.	216,4	156	240	66	198	174,2	28,0	7,7
Image displac.	78,4	56	50	47	71,5	62,5	5,8	5,5
Alpha Blending	141	100	86	20,0	129	111,7	10,0	2,3
Blur detection	72,8	52	84	19,5	19,5	58,0	9,8	2,3
Convolution(3x3)	422,8	302	199	58	66,0	337,2	23,2	6,8
Histogram	21,4	21,4	29,0	20,1	19,25	23,5	3,4	2,3
Image Rotation	546	390	608	234	500	435,5	70,9	27,3
Image Scaling	384	250	390	143	352	279,2	45,5	16,7

manner. However, for memory intensive operations, with bottlenecks mainly dependent on fast data access, the performance gains are expected to be smaller.

The low-power design of the EnCore/CFA configuration also implies a very important gain in energy consumption. The reconfigurable setup outperforms the ARM/NEON combination, consuming only 5% of the energy.

The EnCore processor with its CFA proves to be a very good alternative to reduce the power consumption of mobile microprocessors. In this context, an asymmetric configuration of one or two ARM cores with several EnCore processors in a multicore architecture could be a viable option for future SoCs.

7.4 SIMD/MIMD dynamically-reconfigurable architecture

As seen in chapters 2 and 3, interactive vision based applications integrate complex computer vision algorithms that include a wide range of operations, data dependencies and program flows. Current GPU devices, although extremely performance-efficient in certain tasks, lack the flexibility of an unrestrained program flow control that can adapt to different types of parallelism when faced with looping and branching. In this context, a reconfigurable architecture that is able to reorganize its processing elements is a suitable candidate to complement current mobile GPUs.

A hybrid SIMD/MIMD dynamically-reconfigurable architecture is essentially an image coprocessor designed to take advantage of the different types of parallelism (data parallelism and task parallelism) present on each algorithm by adding a flexible datapath to the processor. Keeping certain similarities with GPUs, the hybrid platform is essentially a many-core architecture able to process many operations concurrently. However, the addition of the flexible data path allows the architecture to reconfigure during the program flow to select the best characteristics for SIMD and MIMD (Multiple Instruction Multiple Data) computing paradigms.

The hybrid architecture features general purpose capabilities, dynamic and at-runtime reconfiguration that can select the SIMD or MIMD modes as needed. The architecture is completely modular and scalable, adaptable according to the requirements of the algorithm or the target platform. In addition, it aims to reduce the set-up time and ease algorithm migration by automatically managing tasks such as data I/O or synchronization between the computing units. The architectural details can be found in the work of Nieto *et al.* (2012a).

Figure 45 depicts the main elements of the Image Coprocessor. It is composed of three main elements: two I/O Processors, the Programmable Input Processor (PIP) and the Programmable Output Processor (POP), and a set of Processing Elements (PEs). Depending on the configuration of the coprocessor, the set of PEs can execute both SIMD and MIMD computing paradigms. In the SIMD mode, all PEs execute the same instruction, exploiting the spatial (data) parallelism. In the MIMD mode, each PE executes a small kernel of the whole algorithm, making it possible to take advantage of the temporal (task) parallelism. Two different networks enable data sharing between the PEs.

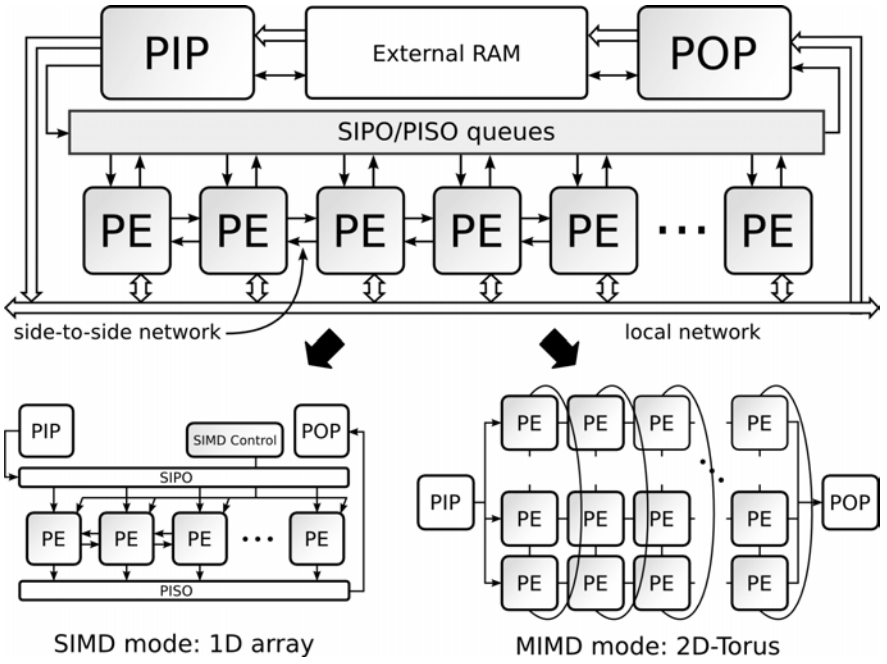


Fig 45. Schematic view of the Hybrid Image Coprocessor and the operation modes. (Bordallo López *et al.* 2014) ©Springer.

In the SIMD mode, adjacent PEs can exchange data synchronously using the *side-to-side network*, while in the MIMD mode, the different kernels executed on the different PEs are chained, employing the *local network*. This mode uses the Stream Queues to enable automatic synchronization, therefore no additional operations are needed. The different modules of the architecture are enabled, depending on the operation mode, and this selection depends on the algorithm characteristics and how the tasks are scheduled.

7.4.1 Accelerating vision-based applications with a Hybrid reconfigurable architecture

The hybrid SIMD/MIMD architecture is currently prototyped on an FPGA for evaluation purposes. The target device is a Xilinx Virtex-6 X240T, included on the Xilinx ML605 Base Board (Xilinx Inc. 2014). An AXI4-based MicroBlaze SoC with Multi-Port Memory Controller and 10/100-Ethernet units was implemented to support the Image Coprocessor. It was configured with 128-PEs of 32-bit each. Their ALUs only support integer and fixed-point arithmetic in order to save FPGA resources. Due to the FPGA characteristics, turning on the prototype consumes a static power of 1.97W. Clocked up to 150MHz, the peak performance is 19.6GOP/s and the maximum power consumption is 7.197W. More details of the hardware prototype are available in the article by Nieto *et al.* (2012a).

To provide an example of the possible benefits of employing a Hybrid SIMD/MIMD platform for accelerating vision-based interactive applications, this section evaluates its use in several computer vision kernels utilized in many interactive camera applications and vision-based UIs. The performance is compared with an ARM Cortex-A8 processor and a PowerVR530 GPU, both included on the OMAP3430 SoC. The results, partially presented in previous work (Nieto 2012) **(Bordallo López *et al.* 2014)** (Nieto *et al.* 2012b), show that the Hybrid SIMD/MIMD platform can outperform mobile CPUs and GPUs in scenarios requiring a flexible data path and parallel computations. Table 12 summarizes the performance of the platform compared with a mobile CPU and a mobile GPU.

The measurements show that the Hybrid architecture, designed with emphasis in performance, outperforms the ARM processor in speed and energy efficiency for all the implemented image kernels. When compared with a mobile GPU, the flexibility of the Hybrid platform offers a considerable advantage in operations that require a more complicated program flow, such as feature extraction, 2D-convolution or LBP computation. Its flexibility is better exploited with long image pipelines that can take more advantage of its task-parallel capabilities through the SIMD configuration.

However, when data access patterns become irregular, the performance of the Hybrid platform is hindered. For example, for pixel-wise operations typically present in graphics processing such as image rotation and scaling, the well optimized GPU still outperforms the Hybrid platform.

Table 12. Cycles per pixel needed by several algorithms in the mobile CPU, mobile GPU and Hybrid platforms.

	CPP			nJPP		
Operation	ARM	mGPU	Hybrid	ARM	mGPU	Hybrid
Grayscale conversion	156	13,4	2,1	174,2	11,3	98,0
Image displacement	56	13,6	1,3	62,5	11,5	60,7
Alpha Blending	100	13,6	1,0	111,7	11,5	46,7
Feature Extraction	549	75,5	0,7	613,0	63,8	32,7
Blur detection	52	100,7	1,0	58,0	85,1	46,7
LBP extraction	37	18	0,2	41,3	15,2	6,8
2D-convolution(3x3)	302	160	1,0	337,2	135,3	46,7
Histogram	1,1	-	2,4	1,22	-	112
Image Rotation	390	13,6	12,0	435,5	11,5	560
Image Scaling	250	20	136,7	279,1	16,9	6379

The speedups obtained by the hybrid platform imply a smaller energy consumption for the less parallelizable kernels, such as the image histogram or the feature matching even when implemented on an FPGA. The implementation in silicon could drop the energy consumption by at least one order of magnitude (Kuon & Rose 2007) (Chinnery & Keutzer 2005).

The results suggest that a Hybrid SIMD-MIMD platform is a good alternative to be used in conjunction with a GPU, providing for a flexible architecture that is able to exploit different types of parallelism, supporting different program flows.

7.5 Transport-triggered architecture

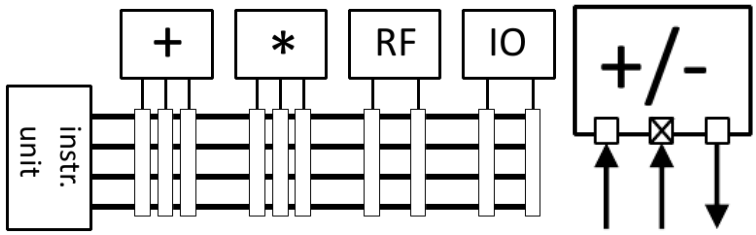
Current mobile Image Signal Processor (ISP) architectures are based in a combination of programmable Digital Signal Processors with monolithic and inflexible hardware *codecs*. Future vision-capable mobile platforms are expected to provide for energy-efficient solutions with enough flexibility and programmability to adapt to several scenarios. In this context, the inclusion of reconfigurable architectures in future devices, designed for computing and sensing tasks is a suitable solution.

Transport-Triggered Architecture (TTA) is a processor technology that is fundamentally different from conventional processor designs (Corporaal 1997). TTA resembles the VLIW processor architecture and exploits instruction-level parallelism, executing

multiple instructions simultaneously in the same clock cycle. While in mainstream embedded and signal processors computations are triggered by processor instructions that are accompanied with operands, in TTA processors there is only one instruction: move data. Computational operations are triggered as side-effects of data moves. TTA resembles the VLIW processor architecture and exploits instruction-level parallelism executing multiple instructions simultaneously in the same clock cycle

TTAs fetch and execute several instructions in parallel every clock cycle. This makes TTA processors well-suited for computationally intensive signal processing-style computations that offer abundant instruction-level parallelism.

An example TTA processor is depicted in Figure 46. In TTA design, there is no theoretical limit to the number of buses (and respectively, number of instructions executed in parallel), however, the maximum operating frequency goes down as the number of buses increases.



(a) Toy TTA (b) Function Unit

Fig 46. A toy TTA and one of the function units. (Bordallo López *et al.* 2014) ©Springer.

Also, the maturity of the TTA design tools that include a compiler of standard C code, makes the platform especially attractive for high performance applications with moderate development times, and easy to integrate with other chip designs.

7.5.1 Improving intensive operations with TTA processors

To provide for an example of the possible benefits of employing TTA processors for accelerating vision-based interactive applications, this section evaluates the use of two different programmable application-specific instruction processor (ASIP). The first one implemented by Boutellier *et al.* (2012) is capable of performing LBP feature extraction for HDTV resolution video at a modest clock frequency of 304MHz in real time **(Bordallo López *et al.* 2014)**. The second one, presented by Pitkänen *et al.* (2006)

is designed to accelerate 1024-point fast Fourier transforms with minimum latencies, employing a maximum frequency of 400MHz.

The custom TTA processors, have been designed with the open source TCE tool-set (Esko *et al.* 2010) that provides a complete co-design flow all the way to ASIC synthesis. The programmability of the processor enables changing its operation solely by software updates. To verify the functionality and to measure the power consumption to test the suitability for mobile devices, the processors can be synthesized on an FPGA board.

In the case of the LBP processor, for evaluation purposes, the FPGA used for measurements and testing was Altera EP4CE115F29C7 Cyclone IV. The FFT processor was synthesized using 130nm CMOS standard sell ASIC technology.

The resulting processor, a TTA-based ASIP, can be integrated in face detection and tracking system such as the one presented in Chapter 3. The system, designed to minimize the dissipated power while keeping the programmability requires about 11 Cycles per Pixel (*CPP*) for the non-interpolated LBP, while the interpolated LBP requires 20 *CPP*. This implies a figure of energy consumption equivalent to only 1.1 pJ/pixel, which proves that the approach is extremely power efficient. This means that, even on the FPGA prototype, the real-time HD720 processing at 30 fps. can be achieved while keeping the power consumption below 30mW. It is expected that the synthesis of the processor in silicon could mean a possible increase in the power efficiency of about one order of magnitude.

To provide a comparison, the LBP implementation is compared with highly optimized implementations on a DSP (Humenberger *et al.* 2010) (Patyk *et al.* 2013) and optimized ARM and NEON implementations **(Bordallo López *et al.* 2014) (Bordallo López *et al.* 2012b)**. The DSP core, explicitly designed for signal processing tasks, offers a performance about four times faster than the ARM and NEON implementation. The DSP code is a carefully optimized code and makes use of DSP intrinsics.

For the LBP processor, the experiments suggest a *CPP* count of 6,7 and 11,8 for LBP and interpolated LBP, respectively. These numbers show that the DSP is actually faster than the reconfigurable TTA processor, which make it still a very suitable candidate for high resolution and high performance applications. However, when distributing the performance over power, the TTA processor is about 3,5 times more efficient than the DSP, making it especially suitable for continuous sensing tasks. The energy efficiency of the TTA processor could be improved further with the synthesis of the processor in silicon (Kuon & Rose 2007) (Chinnery & Keutzer 2005).

On the FFT processor, the TTA-architecture outperforms the DSP in terms of *CPP*. In addition, implementation in silicon makes also the processor about 3,5 times more energy efficient. Table 13 shows a summary of the experiments.

Table 13. Cycles per pixel needed by the two TTA processors compared with a DSP and an ARM processor.(1) FPGA. (2) 130nm technology.

		CPP			nJPP	
Operation	ARM	DSP	TTA	ARM	DSP	TTA
LBP	37,1	6,7	11,0	41	3,9	1,1(1)
iLBP	76,8	11,8	20,0	86	6,9	2,0(1)
FFT	160	6	5,0	146	3,5	1,1(2)

The analysis of the results show that the TTA processors outperform the ARM processor in both CPP and JPP metrics. Compared with the more specific DSP, designed to take advantage of instruction parallelism, the performance is comparable. However, the TTA processors prove to be more energy efficient, even when implemented on an FPGA. Future implementations of the LBP processor in silicon are expected to reach even better energy-efficiency.

Although not a replacement for current DSPs, the inclusion of several reconfigurable TTA processors in a mobile architecture enables the computation of several specific tasks with small energy consumption, while still providing enough flexibility.

7.6 Discussion

The main contribution of this chapter is the analysis of the characteristics of reconfigurable computing platforms in the context of mobile vision-based interactivity. The comparison of different architectures and the characterization of their advantages and shortcomings is achieved through the implementation of several compute-intensive image processing kernels. The analysis of the trade-off between the flexibility of general purpose processors and the high performance of dedicated hardware circuits provides advances towards future devices that include platform level adaptability concepts.

The use of reconfigurable processors in future mobile platforms is a very attractive opportunity. However, there are still major challenges that have to be addressed. The main target of future research is the integration of reconfigurable computing into existing

hardware and software systems. In this context, the investment in the development of efficient tools that help the exploitation of such devices is imperative. The identification of novel emerging applications and their possible bottlenecks and constrains in terms of size and energy consumption is paramount for the definition of the boundaries between software and hardware, and the apparition of new paradigms of reconfigurable computing. In this context, the future of reconfigurable computing will be determined by the same trends that affect the development of these systems today. System integration, dynamic reconfiguration and high-level compilation are still major areas that require development (Tessier & Burleson 2001).

8 Conclusions and future research

"We are stuck with technology when what we really want is just stuff that works."

- Douglas Adams (2002)

Mobile hand-held devices are attractive platforms for the development of new interaction methods. Their small size, multiple sensors and cameras call for the creation of new applications and user interfaces. Integrating vision-based interactivity on these devices introduces new challenges and constraints.

Researchers and developers with a role in the field of mobile computer vision need to consider the relationships between interactivity and the platform as a whole, and start designing their systems accordingly. Vision-based algorithms and methods should be analyzed using a holistic process that considers all stages of mobile development, from hardware design to application implementation.

In this context, the main contribution of this thesis is establishing a relationship between interactivity and energy-efficiency, tying the computing and sensing needs of interactive vision-based applications together with the requirements imposed by the battery-powered characteristic of the devices. The solutions contained in this thesis laid on the creation of novel interactive applications and interfaces, the full exploitation of current computing devices, the proposal of new specific sensing architectures and advances towards platform adaptability.

The relationship between interactivity and improved quality can be seen in the implementation of an interactive multi-frame reconstruction application. Since the user will likely collaborate to obtain the best results if provided with enough feedback, the application shows the principles of user-feedback utilization on a real scenario, proving the usefulness of introducing interactive stages in **camera-based applications**.

The implementation of a virtual 3-D display that integrates a camera as a crucial point of a **user interface** has shown the sensing needs of the approach. In complete vision-based user interfaces, which can be utilized to create novel interaction methods and applications, the consideration of the system timing and accuracy proves to be of vital importance. Based on this implementation, new interaction methods deriving from this kind of interface have been presented.

The constraints imposed by the nature of the **mobile architecture** put in relevance the intertwining of all the interactivity qualities. The computational performance characterized by throughput and latency needs to be maximized, but carefully balancing it with the battery consumption, composing an energy-efficient system. The introduction of a set of heterogeneous processors and their exploitation using the principles of asymmetric multiprocessing showed that utilizing all the available resources of the device in the most suitable manner is the way to face the challenge.

With the most suitable tasks distributed on the most appropriate processors, the optimization of the implementations to be adapted to the **computational platform** at hand is of vital importance. The implementation and description of expensive algorithmic kernels on mobile Graphics Processing Units puts in relevance the importance of the exploitation of the supported APIs and the identification of the limitations and opportunities that they offer.

However, since interactivity is intrinsically related with latency and robustness, the consideration of a **sensing platform** that integrates the information of the available cameras and sensors is paramount. This is shown in the implemented methods for the reduction of the perceived latencies. The proposed solutions for sensing platform reorganization should lead to the design of an *always-on* system that includes a dedicated architecture for continuous sensor processing.

The design of a specific processor or platform still poses a challenge that faces the trade-off between the flexibility of application processors and the performance of dedicated hardware circuits. In this context, the analysis of reconfigurable computing platforms has shown the advances towards a much required **platform level adaptability**.

However, despite the latest progress of mobile design, several challenges still remain. The investigation of robust methods that work in variable conditions is still a field to explore. In this context the exploitation of complementary methods such as infra-red or thermal imaging could provide for the required reliability of the systems in difficult lighting or weather conditions. In turn, this can drive the creation of novel and meaningful ways of utilizing vision-based UIs for interactive purposes to complement or substitute the current user interfaces.

To tackle the computing needs of these novel methods, future research should focus on the standardization of the means to use all the computing resources, providing different tools to address all levels of application design, from low-level arithmetic optimization to automatic application partitioning. Also important is the future integration of novel

reconfigurable devices in commercial systems and the creation of suitable developing tools, including high-level compilers and APIs.

For the particular case of vision-based interactivity, the integration of mobile GPUs in the camera pipeline is the next crucial step. In this context, any future standards for GPGPU computing should consider simple ways of accessing the camera data.

With this in mind, several questions are still unanswered. Can continuous sensing, GPU processing and reconfigurable architectures fundamentally change devices? What is the impact of an *always-on advanced vision-based interface* in a device that considers the techniques described earlier?

Future designs are expected to converge to a full range of devices with multiple sizes and shapes, including wearable devices. The interactive requirements of all of them are expected to have certain common characteristics that can be tackled with the principles shown in this thesis. The relationship established between interactivity and energy-efficiency is even more relevant in devices with reduced size that cannot use traditional interaction methods. The principles based on the distribution of the tasks among the most suitable processors, considering interactivity as a crucial part of the system, should play the main role in the developing of the interfaces of future devices.

Recent advances suggest that we have a brilliant future ahead. However, we are still at the beginning of a path full of challenges and opportunities.

References

Abolfazli S, Sanaei Z, Gani A, Xia F & Yang LT (2013) Rich mobile applications:genesis, taxonomy, and open issues. Journal of Network and Computer Applications .

Adams A, Gelfand N & Pulli K (2008) Viewfinder alignment. Proc. Eurographics 2008, 597–606.

Adams A, Talvala EV, Park SH, Jacobs DE, Ajdin B, Gelfand N, Dolson J, Vaquero D, Baek J, Tico M, Lensch HPA, Matusik W, Pulli K, Horowitz M & Levoy M (2010) The frankencamera: An experimental platform for computational photography. ACM Trans. Graph. 29(4): 29:1–29:12.

Adams D (2002) The salmon of doubt. William Heinemann Ltd.

Ahonen T, Hadid A & Pietikäinen M (2006) Face description with local binary patterns: Application to face recognition. IEEE Transactions on Pattern Analysis and Machine Intelligence 28(12): 2037–2041.

Akenine-Möller T & Johnsson B (2012) Performance per what? Journal of Computer Graphics Techniques (JCGT) 1(1): 37–41.

Akenine-Möller T & Strom J (2008) Graphics processing units for handhelds. IEEE 96(5): 779 –789.

Almer O, Bennett R, Böhm I, Murray A, Qu X, Zuluaga M, Franke B & Topham N (2009) An end-to-end design flow for automated instruction set extension and complex instruction selection based on gcc. Proc. First International Workshop on GCC Research Opportunities (GROW'09), 49–60.

Amdahl GM (1967) Validity of the single processor approach to achieving large scale computing capabilities. Proc. April 18-20, 1967, spring joint computer conference, ACM, 483–485.

An Jh & Hong KS (2011) Finger gesture-based mobile user interface using a rear-facing camera. Proc. Consumer Electronics (ICCE), 2011 IEEE International Conference on, IEEE, 303–304.

Apple (2014) Apple iphone.

Apple Inc (2013) M7 coprocessor.

Arc International (2014) http://www.arc.com.

Asanovic K, Bodik R, Catanzaro BC, Gebis JJ, Husbands P, Keutzer K, Patterson DA, Plishker WL, Shalf J, Williams SW *et al.* (2006) The landscape of parallel computing research: A view from berkeley. Technical report, Berkeley University.

Balfour J, Dally WJ, Black-Schaffer D, Parikh V & Park J (2008) An energy-efficient processor architecture for embedded systems. Computer Architecture Letters 7(1): 29–32.

Berjon D, Cuevas C, Moran F & Garcia N (2013) Gpu-based implementation of an optimized nonparametric background modeling for real-time moving object detection. Consumer Electronics, IEEE Transactions on 59(2): 361–369.

Bilcu R, Burian A, Knuutila A & Vehvilainen M (2008) High dynamic range imaging on mobile devices. Proc. Electronics, Circuits and Systems, 2008. ICECS 2008. 15th IEEE International Conference on, 1312 –1315.

Blanco AAFAV (2013) Paralldroid: A framework for parallelism in android tm. Low Energy Application Parallelism Conference (LEAP) .

Bleser G & Stricker D (2009) Advanced tracking through efficient image processing and visual–inertial sensor fusion. Computers & Graphics 33(1): 59–72.

Blume H, Livonius Jv, Rotenberg L, Noll TG, Bothe H & Brakensiek J (2008) Openmp-based parallelization on an mpcore multiprocessor platform–a performance and power analysis.

Journal of Systems Architecture 54(11): 1019–1029.
Bondalapati K & Prasanna VK (2002) Reconfigurable computing systems. IEEE 90(7): 1201–1217.
Bordallo López M, , Hannuksela J & Silvén O (2011a) Mobile feature-cloud panorama construction for image recognition application. Proc. Mobiphoto, International Workshop on Camera Phone Sensing, Penghu, Taiwan, 1–6.
Bordallo López M, Boutellier J & Silvén O (2007) Implementing mosaic stitching on mobile phones. Proc. Finnish Signal Processing Symposium.
Bordallo López M, Hannuksela J, Silvén JO & Vehviläinen M (2011b) Multimodal sensing-based camera applications. Proc. SPIE- The International Society for Optical Engineering, SPIE, P. O. BOX 10 Bellingham WA 98227-0010 USA, 7881.
Bordallo López M, Hannuksela J, Silvén O & Fan L (2012a) Head-tracking virtual 3-d display for mobile devices. Proc. Computer Vision and Pattern Recognition Workshops (CVPRW), 2012 IEEE Computer Society Conference on, IEEE, 27–34.
Bordallo López M, Hannuksela J, Silvén O & Vehviläinen M (2009) Graphics hardware accelerated panorama builder for mobile phones. Proc. Proceeding of SPIE Electronic Imaging 2009, 7256.
Bordallo López M, Hannuksela J, Silven O & Vehvilainen M (2012b) Interactive multi-frame reconstruction for mobile devices. Multimedia Tools and Applications 1–21.
Bordallo López M, Niemelä K & Silvén O (2012c) Gpgpu-based surface inspection from structured white light. Proc. IS&T/SPIE Electronic Imaging, International Society for Optics and Photonics, 829510–829510.
Bordallo López M, Nieto A, Boutellier J, Hannuksela J & Silvén O (2014) Evaluation of lbp computing in multiple architectures. Journal of Real-Time Image Processing 1–34.
Bordallo López M, Nykänen H, Hannuksela J, Silvén O & Vehviläinen M (2011c) Accelerating image recognition on mobile devices using gpgpu. Proc. SPIE Electronic Imaging 2011, 7872.
Borkar S (1999) Design challenges of technology scaling. Micro, IEEE 19(4): 23–29.
Boutellier J, Bordallo López M, Silvén O, Tico M & Vehviläinen M (2007) Creating panoramas on mobile phones. Proc. SPIE Electronic Imaging 2007, 6498.
Boutellier J, Lundbom I, Janhunen J, Ylimäinen J & Hannuksela J (2012) Application-specific instruction processor for extracting local binary patterns. Proc. Conference on Design and Architectures for Signal and Image Processing.
Brookwood N (2010) Amd fusion family of apus: enabling a superior, immersive pc experience. Insight 64(1): 1–8.
Capin T, Pulli K & Akenine-Möller T (2008) The state of the art in mobile graphics research. IEEE Computer Graphics and Applications 1: 74–84.
Carroll A & Heiser G (2010) An analysis of power consumption in a smartphone. Proc. 2010 USENIX conference on USENIX annual technical conference, 21–21.
CEVA Inc (2014) Mm3000 ip platform. http://www.ceva-dsp.com/CEVA-MM3000.
Che S, Boyer M, Meng J, Tarjan D, Sheaffer JW & Skadron K (2008) A performance study of general-purpose applications on graphics processors using CUDA. J. Parallel Distrib. Comput. 68(10): 1370–1380.
Chen JC & Chien SY (2008) Crisp: coarse-grained reconfigurable image stream processor for digital still cameras and camcorders. Circuits and Systems for Video Technology, IEEE Transactions on 18(9): 1223–1236.

Cheng KT & Wang YC (2011) Using mobile gpu for general-purpose computing; a case study of face recognition on smartphones. Proc. VLSI Design, Automation and Test (VLSI-DAT), 2011 International Symposium on, 1 –4.

Cheng KTT, Yang X & Wang YC (2013) Performance optimization of vision apps on mobile application processor. Proc. Systems, Signals and Image Processing (IWSSIP), 2013 20th International Conference on, IEEE, 187–191.

Chinnery DG & Keutzer K (2005) Closing the power gap between asic and custom: an asic perspective. Proc. 42nd annual Design Automation Conference, ACM, 275–280.

Chung D, Kim S, Bae J & Lee S (2009) Photographic expert-like capturing by analyzing scenes with representative image set. Proc. Proc. SPIE, 7252.

Chung E, Burger D, Butts M, Gray J, Thacker C, Vissers K & Wawrzynek J (2013) Reconfigurable computing in the era of post-silicon scaling. Proc. Field-Programmable Custom Computing Machines (FCCM), 2013 IEEE 21st Annual International Symposium on, IEEE, xvi–xvi.

Clipp B, Raguram R, Frahm JM, Welch G & Pollefeys M (2008) A mobile 3d city reconstruction system. Proc. Proc. IEEE VR workshop on Cityscapes, 67.

Consolvo S, McDonald DW, Toscos T, Chen MY, Froehlich J, Harrison B, Klasnja P, LaMarca A, LeGrand L, Libby R *et al.* (2008) Activity sensing in the wild: a field trial of ubifit garden. Proc. SIGCHI Conference on Human Factors in Computing Systems, ACM, 1797–1806.

Cornelis N & Van Gool L (2008) Fast scale invariant feature detection and matching on programmable graphics hardware. Proc. Computer Vision and Pattern Recognition Workshops, 2008. CVPRW '08. IEEE Computer Society Conference on, 1–8.

Corporaal H (1997) Microprocessor Architectures : From VLIW to TTA. John Wiley & Sons.

Crowley J, Berard F, Coutaz J *et al.* (1995) Finger tracking as an input device for augmented reality. Proc. International Workshop on Gesture and Face Recognition, 1415.

Cuervo E, Balasubramanian A, Cho Dk, Wolman A, Saroiu S, Chandra R & Bahl P (2010) Maui: making smartphones last longer with code offload. Proc. 8th international conference on Mobile systems, applications, and services, ACM, 49–62.

Dabrowski J & Munson E (2001) Is 100 milliseconds too fast? Proc. Conference on Human Factors in Computing Systems, 317–318.

Dennard RH, Gaensslen FH, Rideout VL, Bassous E & LeBlanc AR (1974) Design of ion-implanted mosfet's with very small physical dimensions. Solid-State Circuits, IEEE Journal of 9(5): 256–268.

DiVerdi S & Höllerer T (2007) Groundcam: A tracking modality for mobile mixed reality. Proc. IEEE Virtual Reality, 75–82.

Doerffel D & Sharkh SA (2006) A critical review of using the peukert equation for determining the remaining capacity of lead-acid and lithium-ion batteries. Journal of Power Sources 155(2): 395–400.

El-Mahdy A & Elsersy R (2014) A large-scale mobile facial recognition system using embedded gpus. Proc. Proceedings of the High Performance Computing Symposium, Society for Computer Simulation International, 23.

Elhassan I (2005) Fast texture downloads and readbacks using pixl buffer objects in opengl. In: nVidia Technical Brief. nVidia.

Ensor A & Hall S (2011) Gpu-based image analysis on mobile devices. arXiv preprint arXiv:1112.3110 .

Epson Inc (2014) Epson microcontrollers. http://global.epson.com/products/semicon/products/mcu/.

Erol B, Antúnez E & Hull JJ (2008) Hotpaper: multimedia interaction with paper using mobile phones. Proc. 16th ACM international conference on Multimedia, ACM, 399–408.

Esko O, Jääskeläinen P, Huerta P, de La Lama CS, Takala J & Martinez JI (2010) Customized exposed datapath soft-core design flow with compiler support. Proc. 20th International Conference on Field Programmable Logic and Applications, Milano, Italy, 217–222.

Esmaeilzadeh H, Blem E, St Amant R, Sankaralingam K & Burger D (2011) Dark silicon and the end of multicore scaling. Proc. Computer Architecture (ISCA), 2011 38th Annual International Symposium on, IEEE, 365–376.

Esmaeilzadeh H, Blem E, St Amant R, Sankaralingam K & Burger D (2012) Power limitations and dark silicon challenge the future of multicore. ACM Transactions on Computer Systems (TOCS) 30(3): 11.

Fabritius S, Grigore V, Maung T, Loukusa V & Mikkonen T (2003) Towards energy aware system design. Technical report, Nokia Research Center.

Ferreira D, Dey AK & Kostakos V (2011) Understanding human-smartphone concerns: a study of battery life. In: Pervasive Computing, 19–33. Springer.

Ferri C, Viescas A, Moreshet T, Bahar R & Herlihy M (2008) Energy efficient synchronization techniques for embedded architectures. Proc. 18th ACM Great Lakes symposium on VLSI, ACM, 435–440.

Fitzmaurice GW (1993) Situated information spaces and spatially aware palmtop computers. Communications of the ACM 36(7): 39–49.

Francone J & Nigay L (2011) Using the user's point of view for interaction on mobile devices. Proc. 23rd French Speaking Conference on Human-Computer Interaction, ACM, New York, NY, USA, 4:1–4:8.

Freund Y & Schapire RE (1995) A decision-theoretic generalization of on-line learning and an application to boosting. Proc. Second European Conference on Computational Learning Theory, 23–37.

Fung J & Mann S (2008) Using graphics devices in reverse: Gpu-based image processing and computer vision. Proc. Multimedia and Expo, 2008 IEEE International Conference on, 9 –12.

Gelfand N, Adams A, Park SH & Pulli K (2010) Multi-exposure imaging on mobile devices. Proc. international conference on Multimedia, ACM, New York, NY, USA, 823–826.

Gellersen HW, Schmidt A & Beigl M (2002) Multi-sensor context-awareness in mobile devices and smart artifacts. Mobile Networks and Applications 7(5): 341–351.

Goodacre J (2009) The evolution of mobile processing architectures. Proc. ARM Holdings.

Google Inc (2014) Google camera. https://play.google.com/store/apps/details?id=com.google.android.GoogleCamera.

Greenhalgh P (2011) Big.little processing with arm cortex-a15 & cortex-a7. ARM White Paper .

Grochowski E, Ronen R, Shen J & Wang P (2004) Best of both latency and throughput. Proc. Computer Design: VLSI in Computers and Processors, 2004. ICCD 2004. Proceedings. IEEE International Conference on, IEEE, 236–243.

Gustafson JL (1988) Reevaluating amdahl's law. Communications of the ACM 31(5): 532–533.

Ha SJ, Lee SH, Cho NI, Kim SK & Son B (2008) Embedded panoramic mosaic system using auto-shot interface. IEEE Transactions on Consumer Electronics 54(1): 16–24.

Hachet M, Pouderoux J & Guitton P (2005) A camera-based interface for interaction with mobile handheld computers. Proc. I3D'05 - ACM SIGGRAPH 2005 Symposium on Interactive 3D Graphics and Games, ACM Press, 65–71.

Hadid A, Pietikäinen M & Ahonen T (2004) A discriminative feature space for detecting and recognizing faces. Proc. Proc. IEEE Conference on Computer Vision and Pattern Recognition, Washington, D.C.

Hakanen J (2014) Accelerating image processing pipeline on mobile devices using gpu. Ph.D. thesis, Tampere University of Technology.

Han S & Philipose M (2013) The case for onloading continuous high-datarate perception to the phone. Proc. 14th USENIX conference on Hot Topics in Operating Systems, USENIX Association, 5–5.

Han S, Yang S, Kim J & Gerla M (2012) Eyeguardian: a framework of eye tracking and blink detection for mobile device users. Proc. Twelfth Workshop on Mobile Computing Systems & Applications, ACM, New York, NY, USA, 6:1–6:6.

Hannuksela J (2008) Camera based motion estimation and recognition for human-computer interaction. Ph.D. thesis, Dissertation, Acta Univ Oul C 313, 89 p.

Hannuksela J, Huttunen S, Sangi P & Heikkilä J (2007a) Motion-based finger tracking for user interaction with mobile phones. Proc. 4th European Conference on Visual Media Production, London, UK.

Hannuksela J, Sangi P & Heikkilä J (2007b) Vision-based motion estimation for interaction with mobile devices. Computer Vision and Image Understanding: Special Issue on Vision for Human-Computer Interaction 108(1–2): 188–195.

Hannuksela J, Sangi P, Heikkila J, Liu X & Doermann D (2007c) Document image mosaicing with mobile phones. Proc. Image Analysis and Processing, 2007. ICIAP 2007. 14th International Conference on, IEEE, 575–582.

Hannuksela J, Sangi P, Heikkilä J, Liu X & Doermann D (2007d) Document image mosaicing with mobile phones. Proc. 14th International Conference on Image Analysis and Processing, 575–580.

Hannuksela J, Silvén O, Ronkäinen S, Alenius S & Vehviläinen M (2010) Camera assisted multimodal user interaction. Proc. SPIE Electronic Imaging 2010, 754203.

Haro A, Mori K, Capin T & Wilkinson S (2005) Mobile camera-based user interaction. Proc. IEEE International Conference on Computer Vision, Workshop on Human-Computer Interaction, Beijing, China, 79–89.

Harris C & Stephens M (1988) A combined corner and edge detector. Proc. Alvey vision conference, Manchester, UK, 15: 50.

Heikkinen MV & Nurminen JK (2010) Consumer attitudes towards energy consumption of mobile phones and services. Proc. Vehicular Technology Conference Fall (VTC 2010-Fall), 2010 IEEE 72nd, IEEE, 1–5.

Henderson CR (1975) Best linear unbiased estimation and prediction under a selection model. Biometrics 423–447.

Herrera D, Kannala J & Heikkilä J (2011) Accurate and practical calibration of a depth and color camera pair. CAIP, Computer analysis of images and patterns 1(6855): 437–445.

Heymann S, Muller K, Smolic A, Frohlich B & Wiegand T (2007) Sift implementation and optimization for general-purpose gpu. International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision .

Hinckley K, Pierce JS, Sinclair M & Horvitz E (2000) Sensing techniques for mobile interaction. Proc. 13th annual ACM symposium on User Interface Software and Technology, 91–100.

Hofmann R, Seichter H & Reitmayr G (2012a) A gpgpu accelerated descriptor for mobile devices. Proc. Mixed and Augmented Reality (ISMAR), 2012 IEEE International Symposium on,

IEEE, 289–290.
Hofmann R *et al.* (2012b) Extraction of natural feature descriptors on mobile gpus. Ph.D. thesis, Faculty of Informatics, Universitat Koblenz-Landau.
Horowitz M, Alon E, Patil D, Naffziger S, Kumar R & Bernstein K (2005) Scaling, power, and the future of cmos. Proc. Electron Devices Meeting, 2005. IEDM Technical Digest. IEEE International, IEEE, 7–pp.
Humenberger M, Zinner C & Kubinger W (2009) Performance evaluation of a census-based stereo matching algorithm on embedded and multi-core hardware. Proc. Proceedings of 6th International Symposium on Image and Signal Processing and Analysis, 2009. ISPA 2009, 388 –393.
Humenberger M, Zinner C, Weber M, Kubinger W & Vincze M (2010) A fast stereo matching algorithm suitable for embedded real-time systems. Computer Vision and Image Understanding 114(11): 1180–1202.
Hung CY, Estevez LW & Rabadi WA (2003) Multiplexer reconfigurable image processing peripheral having for loop control. US Patent 6,530,010.
Hürst W & van Wezel C (2013) Gesture-based interaction via finger tracking for mobile augmented reality. Multimedia Tools and Applications 62(1): 233–258.
Huttunen S, Rahtu E, Kunttu I, Gren J & Heikkilä J (2011) Real-time detection of landscape scenes. Proc. Image Analysis, SCIA 2011 Proceedings, Lecture Notes in Computer Science, Springer Berlin/Heidelberg, 338–347.
Hwang J, Jung J & Kim GJ (2006) Hand-held virtual reality: A feasibility study. Proc. ACM Virtual Reality Software and Technology, 356–363.
Imagination-Unlimited (2014) Instaframe photo collage. https://play.google.com/store/apps/details?id=com.ImaginationUnlimited.instaframe.
Inc A (2014) Amazon fire phone. http://www.amazon.com/Fire_Phone_13MP-Camera_32GB/dp/B00EOE0WKQ.
Intel Inc (2014) Silicon hive vliw processor. http://intel.com/.
Intsig-Information (2014) https://play.google.com/store/apps/details?id=com.intsig.camscanner.
Jääskeläinen P, de La Lama C, Huerta P & Takala J (2010) Opencl-based design methodology for application-specific processors. Proc. Embedded Computer Systems (SAMOS), 2010 International Conference on, 223 –230.
Jiang B, Neumann U & You S (2004) A robust hybrid tracking system for outdoor augmented reality. Virtual Reality Conference, IEEE 3.
Joselli M, Passos EB, Junior JRS, Zamith M, Clua E, Soluri E & Tecnologias N (2012) A flocking boids simulation and optimization structure for mobile multicore architectures.
Joselli M, Silva Jr JR, Clua E & Soluri E (2013) Mobilewars: A mobile gpgpu game. In: Entertainment Computing–ICEC 2013, 75–86. Springer.
Joshi N, Kar A & Cohen M (2012) Looking at you: fused gyro and face tracking for viewing large imagery on mobile devices. Proc. 2012 ACM annual conference on Human Factors in Computing Systems, ACM, 2211–2220.
Kalman RE (1960) A new approach to linear filtering and prediction problems. Transactions of the ASME-Journal of Basic Engineering 82: 35–45.
Kalva H, Colic A, Garcia A & Furht B (2011) Parallel programming for multimedia applications. Multimedia Tools Appl. 51(2): 801–818.
Kannala J & Brandt S (2006) A generic camera model and calibration method for conventional, wide-angle, and fish-eye lenses. Pattern Analysis and Machine Intelligence, IEEE Transactions

on 28(8): 1335 –1340.
Kawahara Y, Kurasawa H & Morikawa H (2007) Recognizing user context using mobile handsets with acceleration sensors. Proc. Portable Information Devices, 2007. PORTABLE07. IEEE International Conference on, IEEE, 1–5.
Kayombya GR (2010) Sift feature extraction on a smartphone gpu using opengl es2. 0. Ph.D. thesis, Massachusetts Institute of Technology.
Kim NS, Austin T, Baauw D, Mudge T, Flautner K, Hu JS, Irwin MJ, Kandemir M & Narayanan V (2003) Leakage current: Moore's law meets static power. Computer 36(12): 68–75.
Kim S & Su WY (1993) Recursive high-resolution reconstruction of blurred multiframe images. Image Processing, IEEE Transactions on 2(4): 534 –539.
Klein G & Murray D (2009) Parallel tracking and mapping on a camera phone. Proc. Mixed and Augmented Reality, 2009. ISMAR 2009. 8th IEEE International Symposium on, IEEE, 83–86.
Klein GSW & Drummond T (2002) Tightly integrated sensor fusion for robust visual tracking. Proc. Proc. British Machine Vision Conference, 787–796.
Konrad M (2014) Parallel Computing for Digital Signal Processing on Mobile Device GPUs. Ph.D. thesis, Faculty of Business Sciences, HTW Berlin, Univerisity of Applied Sciences.
Korpipää P & Mäntyjärvi J (2003) An ontology for mobile device sensor-based context awareness. In: Modeling and Using Context, 451–458. Springer.
Kratz S & Ballagas R (2007) Gesture recognition using motion estimation on mobile phones. Proc. 3rd Intl. Workshop on Pervasive Mobile Interaction Devices at Pervasive 2007.
Kristof P, Yu H, Li Z & Tian X (2012) Performance study of SIMD programming models on intel multicore processors. Proc. Parallel and Distributed Processing Symposium Workshops PhD Forum (IPDPSW), 2012 IEEE 26th International, 2423 –2432.
Kuhmann W, Boucsein W, Schaefer F & Alexander J (1987) Experimental investigation of psychophysiological stress-reactions induced by different response times in human-computer interaction. Ergonomics 30: 933–943.
Kumar R, Tullsen DM, Jouppi NP & Ranganathan P (2005) Heterogeneous chip multiprocessors. Computer 38(11): 32–38.
Kuon I & Rose J (2007) Measuring the gap between fpgas and asics. Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on 26(2): 203–215.
Labrie M & Hebert P (2007) Efficient camera motion and 3d recovery using an inertial sensor. Proc. Computer and Robot Vision, 2007. CRV'07. Fourth Canadian Conference on, IEEE, 55–62.
Lahdenoja O, Laiho M, Maunu J & Paasio A (2007) A massively parallel face recognition system. EURASIP Journal on Embedded Systems 2007(1): 31–31.
Lane ND, Miluzzo E, Lu H, Peebles D, Choudhury T & Campbell AT (2010) A survey of mobile phone sensing. Communications Magazine, IEEE 48(9): 140–150.
Lanuzza M, Perri S, Corsonello P & Margala M (2007) A new reconfigurable coarse-grain architecture for multimedia applications. Proc. Adaptive Hardware and Systems, 2007. AHS 2007. Second NASA/ESA Conference on, IEEE, 119–126.
Lee J (2008) Hacking the nintendo wii remote. Pervasive Computing, IEEE 7(3): 39 –45.
Leppänen J & Eronen A (2011) Context extraction. US Patent App. 14/127,366.
Leskelä J, Nikula J & Salmela M (2009) Opencl embedded profile prototype in mobile device. Proc. Signal Processing Systems, 2009. SiPS 2009. IEEE Workshop on, 279 –284.

Li I, Peek EM, Wünsche B & Lutteroth C (2012a) Enhancing 3d applications using stereoscopic 3d and motion parallax. Proc. AUIC, 59–68.

Li JJ, Kuan CB, Wu TY & Lee JK (2012b) Enabling an opencl compiler for embedded multicore dsp systems. 2012 41st International Conference on Parallel Processing Workshops 0: 545–552.

Liang Y, Zhou X, Yu Z & Guo B (2013) Energy-efficient motion related activity recognition on mobile devices for pervasive healthcare. Mobile Networks and Applications 1–15.

LiKamWa R, Priyantha B, Philipose M, Zhong L & Bahl P (2013) Energy characterization and optimization of image sensing toward continuous mobile vision. Proc. Mobisys.

Lipowezky U *et al.* (2010) Indoor-outdoor detector for mobile phone cameras using gentle boosting. Proc. Computer Vision and Pattern Recognition Workshops (CVPRW), 2010 IEEE Computer Society Conference on, IEEE, 31–38.

Liu X, Doermann D & Li H (2005) Fast camera motion estimation for hand-held devices and applications. Proc. 4th International Conference on Mobile and Ubiquitous Multimedia, 103–108.

Lowe D (2004) Distinctive image feature from scale-invariant keypoints. International Journal of Computer Vision 60(2): 91–110.

Lumsden J & Brewster S (2003) A paradigm shift: alternative interaction techniques for use with mobile & wearable devices. Proc. 2003 conference of the Centre for Advanced Studies on Collaborative research, IBM Press, 197–210.

Luxenburger A, Zimmer H, Gwosdek P & Weickert J (2012) Fast pde-based image analysis in your pocket. In: Scale Space and Variational Methods in Computer Vision, 544–555. Springer.

Maghazeh A, Bordoloi UD, Eles P & Peng Z (2013) General purpose computing on low-power embedded gpus: Has it come of age? Proc. Proceedings of SAMOS XIII.

McCaffey J (2012) Exploring mobile vs. desktop opengl performance. In: Cozzi P & Riccio C (eds) OpenGL Insights, 337–351. CRC Press. http://www.openglinsights.com/.

Miettinen AP & Nurminen JK (2010) Energy efficiency of mobile clients in cloud computing. Proc. 2nd USENIX conference on Hot topics in cloud computing, USENIX Association, 4–4.

Miluzzo E, Lane ND, Fodor K, Peterson R, Lu H, Musolesi M, Eisenman SB, Zheng X & Campbell AT (2008) Sensing meets mobile social networks: the design, implementation and evaluation of the cenceme application. Proc. 6th ACM conference on Embedded network sensor systems, ACM, 337–350.

Möhring M, Lessig C & Bimber O (2004) Optical tracking and video see-through ar on consumer cell phones. Proc. Workshop on Virtual and Augmented Reality of the GI-Fachgruppe AR/VR, 193–204.

Mun M, Reddy S, Shilton K, Yau N, Burke J, Estrin D, Hansen M, Howard E, West R & Boda P (2009) Peir, the personal environmental impact report, as a platform for participatory sensing systems research. Proc. 7th international conference on Mobile systems, applications, and services, ACM, 55–68.

Nah JH, Kang YS, Lee KJ, Lee SJ, Han TD & Yang SB (2010) Mobirt: an implementation of opengl es-based cpu-gpu hybrid ray tracer for mobile devices. Proc. ACM SIGGRAPH ASIA 2010 Sketches, ACM, 50.

Nazhandali L, Zhai B, Olson J, Reeves A, Minuth M, Helfand R, Pant S, Austin T & Blaauw D (2005) Energy optimization of subthreshold-voltage sensor network processors. Proc. 32nd annual international symposium on Computer Architecture, IEEE Computer Society,

Washington, DC, USA, 197–207.
Neuvo Y (2004) Cellular phones as embedded systems. Proc. Solid-State Circuits Conference, 2004. Digest of Technical Papers. ISSCC. 2004 IEEE International, IEEE, 32–37.
Nieto A (2012) Dynamically reconfigurable architecture for embedded Computer Vision systems. Ph.D. thesis, Depaeramento de Electrónica e Computación, Universidad de Santiago de Compostela, Spain.
Nieto A, Vilariño D & Brea V (2012a) SIMD/MIMD dynamically-reconfigurable architecture for high-performance embedded vision systems. Proc. 23rd IEEE International Conference on Application-specific Systems, Architectures and Processors (ASAP 2012).
Nieto A, Vilariño D & V B (2011) Towards the optimal hardware architecture for Computer Vision. InTech.
Nieto A, Vilariño D & V B (2012b) Feature detection and matching on an simd/mimd hybrid embedded processor. Proc. Computer Vision and Pattern Recognition Workshops (CVPRW), 2012 IEEE Computer Society Conference on, 21 –26.
Niskanen M, Turtinen M & Hannuksela J (2012) Detecting and tracking objects in digital images. united states patent 8103058.
Nokia Inc (2014a) Nokia camera. http://www.windowsphone.com/en-us/store/app/nokia-camera/.
Nokia Inc (2014b) Nokia panorama. http://n9-apps.com/nokia-panorama.
nVidia Inc (2014) Compute unified device architecture. https://developer.nvidia.com/cuda-zone.
Ojala T, Pietikäinen M & Harwood D (1996) A comparative study of texture measures with classification based on featured distributions. Pattern Recognition 29(1): 51 – 59.
Olson T (2013) Benchmarking floating-point precision in mobile gpus.
Owens JD, Luebke D, Govindaraju N, Harris M, Krüger J, Lefohn AE & Purcell TJ (2007) A survey of general-purpose computation on graphics hardware. Computer Graphics Forum 26(1).
Park J, Choi J, Seo BK & Park JI (2013) Fast stereo image rectification using mobile gpu. Proc. The Third International Conference on Digital Information Processing and Communications (ICDIPC2013), The Society of Digital Information and Wireless Communication, 485–488.
PASTA group, University of Edinburgh (2014) http://groups.inf.ed.ac.uk/pasta/hw_encore.html.
Pathak A, Hu YC, Zhang M, Bahl P & Wang YM (2011) Fine-grained power modeling for smartphones using system call tracing. Proc. Sixth conference on Computer systems, ACM, 153–168.
Patyk T, Guevorkian D, Pitkanen T, Jaaskelainen P & Takala J (2013) Low-power application-specific fft processor for lte applications. Proc. Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS XIII), 2013 International Conference on, IEEE, 28–32.
Pears N, Olivier P & Jackson D (2008) Display registration for device interaction. Proc. 3rd International Conference on Computer Vision Theory and Applications, 446–451.
Pentland A (2000) Looking at people: Sensing for ubiquitous and wearable computing. Pattern Analysis and Machine Intelligence, IEEE Transactions on 22(1): 107–119.
Peukert W (1897) Über die abhängigkeit der kapazität von der entladestromstärke bei bleiakkumulatoren. Elektrotechnische Zeitschrift 20: 20–21.
Photojojo (2012) Fisheye, macro, wide angle and telephoto phone lenses, url:http://photojojo.com/store/awesomeness/cell-phone-lenses.
Pickard WF & Abbott D (2012) Addressing the intermittency challenge: Massive energy storage in a sustainable future. Proceedings of the IEEE 100(2): 317.

Pitkänen T, Mäkinen R, Heikkinen J, Partanen T & Takala J (2006) Low-power, high-performance tta processor for 1024-point fast fourier transform. In: Embedded Computer Systems: Architectures, Modeling, and Simulation, 227–236. Springer.

Pons-Moll G, Baak A, Helten T, Muller M, Seidel HP & Rosenhahn B (2010) Multisensor-fusion for 3d full-body human motion capture. Proc. Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on, IEEE, 663–670.

Prengler A & Adi K (2009) A reconfigurable simd-mimd processor architecture for embedded vision processing applications. Technical report, SAE Technical Paper.

Priyantha B, Lymberopoulos D & Liu J (2010a) Enabling energy efficient continuous sensing on mobile phones with littlerock. Proc. 9th ACM/IEEE International Conference on Information Processing in Sensor Networks, ACM, 420–421.

Priyantha N, Lymberopoulos D & Liu J (2010b) Eers: Energy efficient responsive sleeping on mobile phones. Proceedings of PhoneSense 2010 1–5.

Pulli K, Baksheev A, Kornyakov K & Eruhimov V (2012) Real-time computer vision with opencv. Commun. ACM 55(6): 61–69.

Pulli K, Chen WC, Gelfand N, Grzeszczuk R, Tico M, Vedantham R, Wang X & Xiong Y (2009a) Mobile visual computing. Proc. Ubiquitous Virtual Reality, 2009. ISUVR '09. International Symposium on, 3 –6.

Pulli K, Chen WC, Gelfand N, Grzeszczuk R, Tico M, Vedantham R, Wang X & Xiong Y (2009b) Mobile visual computing. Proc. Ubiquitous Virtual Reality, 2009. ISUVR'09. International Symposium on, IEEE, 3–6.

Qualcomm (2013) Computer vision with fastcv.

Rachuri KK, Musolesi M & Mascolo C (2010) Energy-accuracy trade-offs in querying sensor data for continuous sensing mobile systems. Proc. Proc. of Mobile Context-Awareness Workshop, 10.

Rajovic N, Rico A, Vipond J, Gelado I, Puzovic N & Ramirez A (2013) Experiences with mobile processors for energy efficient hpc. Proc. Conference on Design, Automation and Test in Europe, EDA Consortium, 464–468.

Rakhmatov D & Vrudhula S (2003) Energy management for battery-powered embedded systems. ACM Trans. Embed. Comput. Syst. 2(3): 277–324.

Ramachandran M, Veeraraghavan A & Chellappa R (2011) A fast bilinear structure from motion algorithm using a video sequence and inertial sensors. Pattern Analysis and Machine Intelligence, IEEE Transactions on 33(1): 186–193.

Randell C & Muller H (2000) Context awareness by analyzing accelerometer data. Proc. ISWC '00: 4th IEEE International Symposium on Wearable Computers, IEEE Computer Society, 175.

Rautaray SS & Agrawal A (2012) Vision based hand gesture recognition for human computer interaction: a survey. Artificial Intelligence Review 1–54.

Ready JM & Taylor CN (2007) Gpu acceleration of real-time feature based algorithms. Proc. IEEE Workshop on Motion and Video Computing, IEEE Computer Society, Washington, DC, USA, 8.

Rekimoto J (1995) A vision-based head tracker for fish tank virtual reality-vr without head gear. Proc. Virtual Reality Annual International Symposium, 1995. Proceedings., 94 –100.

Rekimoto J (1996) Tilting operations for small screen interfaces. Proc. 9th annual ACM symposium on User interface software and technology, ACM Press, 167–168.

Rister B, Wang G, Wu M & Cavallaro JR (2013) A fast and efficient sift detector using the mobile gpu. Proc. International Conference on Acoustics, Speech, and Signal Processing (ICASSP).

Rodríguez-Vázquez A, Domínguez-Castro R, Jiménez-Garrido F, Morillas S, García A, Utrera C, Pardo MD, Listan J & Romay R (2010) A cmos vision system on-chip with multi-core, cellular sensory-processing front-end. In: Cellular nanoscale sensory wave computing, 129–146. Springer.

Rohs M (2004) Real-world interaction with camera-phones. Proc. 2nd International Symposium on Ubiquitous Computing Systems, 39–48.

Ronkainen S (2010) Camera based motion estimation and recognition for human-computer interaction. Ph.D. thesis, Dissertation, Acta Univ Oul C 355, 114 p.

Rusell S (2013) Mobile gpu floating point accuracy variances.

Sangi P, Hannuksela J & Heikkilä J (2007) Global motion estimation using block matching with uncertainty analysis. Proc. 15th European Signal Processing Conference, 1823–1827.

Satyanarayanan M (2005) Avoiding dead batteries. Pervasive Computing, IEEE 4(1): 2–3.

Scheuermann B, Ehlers A, Riazy H, Baumann F & Rosenhahn B (2011) Ego-motion compensated face detection on a mobile device. Proc. Proceeding of Seventh Embedded Computer Vision Workshop 2011,.

Seo BK, Park J & Park JI (2011) 3-d visual tracking for mobile augmented reality applications. Proc. Multimedia and Expo (ICME), 2011 IEEE International Conference on, 1 –4.

Shye A, Scholbrock B & Memik G (2009) Into the wild: studying real user activity patterns to guide power optimizations for mobile architectures. Proc. 42nd Annual IEEE/ACM International Symposium on Microarchitecture, ACM, 168–178.

Siirtola P & Röning J (2012) Recognizing human activities user-independently on smartphones based on accelerometer data. International Journal of Interactive Multimedia & Artificial Intelligence 1(5).

Silvén O & Rintaluoma T (2007) Energy efficiency of video decoder implementations. Proc. F. Fitzek and F. Reichert (eds.) Mobile Phone Programming and its Applications to Wireless Networking, Springer, 421–439.

Singh H, Lee MH, Lu G, Kurdahi FJ, Bagherzadeh N & Chaves Filho EM (2000) Morphosys: an integrated reconfigurable system for data-parallel and computation-intensive applications. Computers, IEEE Transactions on 49(5): 465–481.

Singhal N, Park IK & Cho S (2010) Implementation and optimization of image processing algorithms on handheld gpu. Proc. Image Processing (ICIP), 2010 17th IEEE International Conference on, 4481 –4484.

Singhal N, Yoo JW, Choi HY & Park IK (2012) Implementation and optimization of image processing algorithms on embedded gpu. IEICE TRANSACTIONS on Information and Systems 95(5): 1475–1484.

Sinha SN, Frahm JM, Pollefeys M & Genc Y (2006) Gpu-based video feature tracking and matching. Proc. Workshop on Edge Computing Using New Commodity Architectures.

Stavness I, Lam B & Fels S (2010) pcubee: a perspective-corrected handheld cubic display. Proc. ACM CHI '10: 28th international conference on Human factors in computing systems, 1381–1390.

Sutter H (2005) The free lunch is over: A fundamental turn toward concurrency in software. Dr. DobbâĂŹs Journal 30(3): 202–210.

Swiss-Codemonkeys (2014) Warp. https://play.google.com/store/apps/details?id=com.appspot.swisscodemonkeys.warp.

Tacoty-CN (2014) Beauty camera. https://play.google.com/store/apps/details?id=com.menue.sh.beautycamera.

Tessier R & Burleson W (2001) Reconfigurable computing for digital signal processing: A survey. Journal of VLSI signal processing systems for signal, image and video technology 28(1-2): 7–27.

Texas-Instruments (2011) Omap3530 power estimation spreadsheet. Technical report, Texas Instruments.

Thiagarajan A, Ravindranath L, LaCurts K, Madden S, Balakrishnan H, Toledo S & Eriksson J (2009) Vtrack: accurate, energy-aware road traffic delay estimation using mobile phones. Proc. 7th ACM Conference on Embedded Networked Sensor Systems, ACM, 85–98.

Tian Y, Yap KH & He Y (2011) Vehicle license plate super-resolution using soft learning prior. Multimedia Tools and Applications 1–17. 10.1007/s11042-011-0821-2.

Tolia N, Andersen DG & Satyanarayanan M (2006) Quantifying interactive user experience on thin clients. Computer 39(3): 46–52.

Turk M & Robertson G (2000) Perceptual user interfaces (introduction). Commun. ACM 43(3): 32–34.

Uzun I, Amira A & Bensaali F (2003) A reconfigurable coprocessor for high-resolution image filtering in real time. Proc. 10th IEEE International Conference onElectronics, Circuits and Systems, 2003. ICECS 2003., IEEE, 1: 192–195.

Vandewalle P, Süusstrunk S & Vetterli M (2006) A frequency domain approach to registration of aliased images with application to super-resolution. EURASIP Journal on Applied Signal Processing (special issue on Super-resolution) 24: 1–14.

Viola P & Jones M (2001) Rapid object detection using a boosted cascade of simple features. Proc. Computer Vision and Pattern Recognition, 2001. CVPR 2001. 2001 IEEE Computer Society Conference on, 1: I–511 – I–518 vol.1.

von Sydow T, Neumann B, Blume H & Noll TG (2006) Quantitative analysis of embedded fpga-architectures for arithmetic. Proc. Application-specific Systems, Architectures and Processors, 2006. ASAP'06. International Conference on, IEEE, 125–131.

Vuduc R & Czechowski K (2011) What gpu computing means for high-end systems. Micro, IEEE 31(4): 74–78.

Vuori T, Alakarhu J, Salmelin E & Partinen A (2013) Nokia pureview oversampling technology. Proc. IS&T/SPIE Electronic Imaging, International Society for Optics and Photonics, 86671C–86671C.

Wagner D, Mulloni A, Langlotz T & Schmalstieg D (2010) Real-time panoramic mapping and tracking on mobile phones. Proc. Virtual Reality Conference (VR), IEEE.

Wang G, Rister B & Cavallaro JR (2013a) Workload analysis and efficient opencl-based implementation of sift algorithm on a smartphone. Proc. IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP).

Wang G, Xiong Y, Yun J & Cavallaro JR (2013b) Accelerating computer vision algorithms using opencl framework on the mobile gpu-a case study. Proc. IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP).

Wang G, Xiong Y, Yun J & Cavallaro JR (2014) Computer vision accelerators for mobile systems based on opencl gpgpu co-processing. Journal of Signal Processing Systems 1–17.

Wang X & Ziavras SG (2004) Hera: A reconfigurable and mixed-mode parallel computing engine on platform fpgas. Proc. 16th International Conference on Parallel and Distributed Computing and Systems (PDCS), 374–379.

Wang Y & Donyanavard K B Cheng (2010) Energy-aware real-time face recognition system on mobile cpu-gpu platform. Proc. International Workshop on Computer Vision on GPU.
Wang YC & Cheng KTT (2012) Energy and performance characterization of mobile heterogeneous computing. Proc. Signal Processing Systems (SiPS), 2012 IEEE Workshop on, IEEE, 312–317.
Wang YC, Pang S & Cheng KT (2010) A gpu-accelerated face annotation system for smartphones. Proc. International conference on Multimedia, ACM, 1667–1668.
Winkler S, Rangaswamy K & Zhou Z (2007) Intuitive map navigation on mobile devices. In: Stephanidis C (ed) 4th International Conference on Universal Access in Human-Computer Interaction, Part II, HCI International 2007, LNCS 4555, 605–614. Springer, Beijing, China.
Wolf W (2004) The future of multiprocessor systems-on-chips. Proc. Design Automation Conference, 2004. Proceedings. 41st, IEEE, 681–685.
Xilinx Inc (2014) http://www.xilinx.com.
Xiong Y & Pulli K (2010) Fast panorama stitching for high-quality panoramic images on mobile phones. IEEE Transactions on Consumer Electronics 56(2): 298–306.
Yan Z, Subbaraju V, Chakraborty D, Misra A & Aberer K (2012) Energy-efficient continuous activity recognition on mobile phones: An activity-adaptive approach. Proc. Wearable Computers, 2012 16th International Symposium on, IEEE, 17–24.
Yang C, Wu Y & Liao S (2012) O2render: An opencl-to-renderscript translator for porting across various gpus or cpus. Proc. Embedded Systems for Real-time Multimedia (ESTIMedia), 2012 IEEE 10th Symposium on, 67–74.
Yang X & Cheng KTT (2012) Accelerating surf detector on mobile devices. Proc. 20th ACM international conference on Multimedia, ACM, 569–578.
You Y & Mattila VV (2013) Visualizing web mash-ups for in-situ vision-based mobile ar applications. Proc. 21st ACM international conference on Multimedia, ACM, 413–414.
Zhang L, Tiwana B, Qian Z, Wang Z, Dick RP, Mao ZM & Yang L (2010) Accurate online power estimation and automatic battery behavior based power model generation for smartphones. Proc. Eighth IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis, ACM, 105–114.
Zhou F, Duh HBL & Billinghurst M (2008) Trends in augmented reality tracking, interaction and display: A review of ten years of ismar. Proc. 7th IEEE/ACM International Symposium on Mixed and Augmented Reality, IEEE Computer Society, 193–202.
Zolynski G, Braun T & Berns K (2008) Local binary pattern based texture analysis in real time using a graphics processing unit. Proc. VDI wissenforum GmbH - Proceedings of Robotik.
Zuluaga M & Topham N (2008) Resource sharing in custom instruction set extensions. Proc. Application Specific Processors, 2008. SASP 2008. Symposium on, IEEE, 7–13.

Book orders:
Granum: Virtual book store
http://granum.uta.fi/granum/

UNIVERSITY OF OULU P.O. Box 8000 FI-90014 UNIVERSITY OF OULU FINLAND

ACTA UNIVERSITATIS OULUENSIS

ISBN 978-952-62-0671-4 (Paperback)
ISBN 978-952-62-0672-1 (PDF)
ISSN 0355-3213 (Print)
ISSN 1796-2226 (Online)

UNIVERSITY of OULU
OULUN YLIOPISTO