UNIVERSITY OF VAASA

FACULTY OF TECHNOLOGY

AUTOMATION TECHNOLOGY

Kari Koivuporras

ENHANCING A NEUROSURGICAL IMAGING SYSTEM WITH A PC-BASED VIDEO PROCESSING SOLUTION

Master's thesis for the degree of Master of Science in Technology submitted for inspection, Vaasa, 15 March 2011.

Supervisor Jarmo Alander

Instructor Petri Välisuo

PREFACE

This thesis is a contribution to CLIN-NIRce, a subproject of the FIELD NIRce project, concentrating on clinical applications. The research in FIELD NIRce is focused on developing portable, self-contained solutions for near-infrared spectroscopy in the field, as opposed to a laboratory environment. The research in FIELD NIRce is funded by the European Union (EU) via the Botnia-Atlantica programme and several public funders in Finland and Sweden.

The purpose of this thesis is to explore and implement new features for a particular medical imaging system, built upon a neurosurgical operating microscope at Töölö Hospital, Helsinki. The collaborative efforts on this topic were initiated in August 2008 by the automation research group working at the Department of Electrical Engineering and Energy Technology in University of Vaasa and a group of neurosurgeons at the Department of Neurosurgery of Helsinki University Central Hospital (HUCS).

The work included in this thesis was done between April 2010 and March 2011. The majority of the work was done at the University of Vaasa and at home in Vaasa and Raasepori. The project also included several meetings at Töölö Hospital for gathering feedback from the surgeons about the implemented features and to get ideas for new ones. This work was funded by the FIELD NIRce project.

I would like to thank my supervisor Professor Jarmo Alander and my instructor M.Sc. Petri Välisuo for the invaluable guidance and support during the project. I would also like to thank the neurosurgeons M.D. Ph.D. Martin Lehečka and M.D. Ph.D. Aki Laakso for the smooth and productive collaboration. A special thanks for making this work possible goes to Professor Paul Geladi, the head of the FIELD NIRce project, and the head of the Töölö Hospital neurosurgical team, Professor Juha Hernesniemi.

Vaasa, 30 March 2011

Kari Koivuporras

TABLE OF CONTENTS **page**

PREFACE 1

LIST OF FIGURES 5

LIST OF TABLES 6

SYMBOLS AND ABBREVIATIONS 7

TIIVISTELMÄ 11

ABSTRACT 11

1. INTRODUCTION 13
   1.1. Technical background and project guidelines 14
   1.2. Related work 15
   1.3. Thesis structure 16

2. BACKGROUND AND WORKING ENVIRONMENT 18
   2.1. Operating room environment 18
      2.1.1. Surgery staff 18
      2.1.2. Neurosurgical operating microscopes 19
      2.1.3. Instrumentation in the target operating room 20
   2.2. Introduction to indocyanine green angiography 22
      2.2.1. Instrumentation and principle of angiography 22
      2.2.2. Clinical properties and usability 24
   2.3. Microscope-integrated indocyanine green videoangiography 25

3. EFFICIENT VIDEO PROCESSING ON PERSONAL COMPUTERS 27

3.1. Presentation of digital image and video data 27
3.1.1. Video and image compression 28
3.1.2. Video formats, containers and codecs 30
3.2. Essential features for high-throughput video processing systems 31
3.2.1. Parallel processing 31
3.2.1.1. Increasing instruction level parallelism 32
3.2.1.2. Increasing data level parallelism 33
3.2.2. Memory subsystem 34
3.2.2.1. Main memory architectures 34
3.2.2.2. Buses and I/O 36
3.2.3. Software design for performance 36
3.2.3.1. Memory management 37
3.2.3.2. Software optimization 39
3.3. PC as a video processing platform 40
3.3.1. Hardware platform 40
3.3.1.1. System architecture 41
3.3.1.2. Computational capabilities 43
3.3.2. Software platform 44
3.3.2.1. Operating systems 45
3.3.2.2. Video processing software 45
3.3.2.3. Rendering models 49

4. ARCHITECTURE OF THE IMAGING STATION 51
4.1. Hardware configuration 51
4.2. Software platform 52
4.2.1. GNU/Linux 53
4.2.2. X window system 55
4.2.3. GNOME desktop environment 57
4.2.4. Application development libraries 58
4.2.4.1. GTK+ 58
4.2.4.2. GStreamer 59

5. RESULTS 61
5.1. Selection of features 61
5.2. Implementation of the video processing application 62
5.2.1. User interface and features 62
5.2.2. Video processing pipelines 64
5.2.3. Performance 67
5.3. Image post-processing experiments 67
5.3.1. Blending angiograms with anatomic images 68
5.3.1.1. Performance tests 68
5.3.1.2. Clinical usefulness 71
5.3.2. Pseudo coloring angiograms 72

6. CONCLUSIONS 74

BIBLIOGRAPHY 76

APPENDICES 87
APPENDIX 1. Transcript of a meeting at Töölö Hospital, 12.1.2009 87
APPENDIX 2. Transcript of a meeting at Töölö Hospital, 22.2.2010 89
APPENDIX 3. Transcript of a meeting at Töölö Hospital, 12.5.2010 90
APPENDIX 4. Transcript of a meeting at Töölö Hospital, 6.9.2010 91
APPENDIX 5. Transcript of a meeting at Töölö Hospital, 15.2.2011 92

LIST OF FIGURES **page**

Figure 1. The OPMI Pentero operating microscope at Töölö Hospital. 21
Figure 2. Typical excitation and emission curves for a fluorescent dye. 22
Figure 3. The principle of fluorescence imaging and indocyanine green angiography. 23
Figure 4. A still frame captured from an ICG videoangiogram. 26
Figure 5. An example of a four-buffer multibuffering scheme utilizing DMA. 38
Figure 6. A traditional PC system built around a motherboard-integrated northbridge. 42
Figure 7. The layout of a modern high-end PC system. 43
Figure 8. The connections between the PC imaging station and operating microscope imaging system. 52
Figure 9. The server-client model used by the X window system. 56
Figure 10. The hierarchy of software libraries used in the traditional, non-accelerated GUI programming for the X window system. 57
Figure 11. The user interface of iStation showing the playback mode in side-by-side view. 63
Figure 12. A data flow diagram of the iStation playback pipelines. 65
Figure 13. A data flow diagram of the iStation notification pipelines. 66
Figure 14. A data flow diagram of the iStation recording pipelines. 66
Figure 15. The GStreamer pipeline used for alpha blending two video streams. 69
Figure 16. A screen shot of the OpenCV demo application used in the feasibility studies on alpha blending. 72
Figure 17. A pseudo coloring example using the "hot metal" palette from Amide. 73
Figure 18. A pseudo coloring example using the "inverse red" palette from Amide. 73

LIST OF TABLES **page**

Table 1. Some AV containers and the video codec standards they support. 30
Table 2. A comparison of some GPU accelerated video processing APIs. 47
Table 3. A collection of frameworks that can be used for executing image processing code on the GPU. 48
Table 4. A reference hardware specification for the imaging station. 53
Table 5. The setup of the PC used for the feasibility tests on alpha blending. 70
Table 6. The results of the alpha blending performance tests on the OpenCV application. 70
Table 7. The results of the alpha blending performance tests on GStreamer. 71

SYMBOLS AND ABBREVIATIONS

Abbreviations

2D	Two-Dimensional
3D	Three-Dimensional
6DoF	Six Degrees of Freedom
ALU	Arithmetic Logic Unit
AMD	Advanced Micro Devices
API	Application Programming Interface
AV	Audio and Visual
AVC	Advanced Video Coding
AVI	Audio Video Interleave
AVX	Advanced Vector Extensions
CCD	Charge Coupled Device
Cg	C for Graphics
CISC	Complex Instruction Set Computer
CMOS	Complementary Metal Oxide Semiconductor
CMP	Chip Multiprocessor
CPU	Central Processing Unit
CT	Computed Tomography
CUDA	Compute Unified Device Architecture
DCT	Discrete Cosine Transform
DDR	Double Data Rate
DLP	Data Level Parallelism
DMA	Direct Memory Access
DRAM	Dynamic Random Access Memory
DRI	Direct Rendering Interface
DSA	Digital Subtraction Angiography
DSP	Digital Signal Processor
FI	Fluorescence Imaging
FOSS	Free and Open-Source Software
FPGA	Field Programmable Gate Array
fps	Frames per second
FSB	Front Side Bus
GCC	GNU Compiler Collection

GDK	GIMP Drawing Kit
GDI	Graphics Device Interface
GIMP	GNU Image Manipulation Program
GLSL	OpenGL Shading Language
GLX	OpenGL Extension to the X Window System
GNOME	GNU Network Object Model Environment
GNU	GNU is Not Unix
GPGPU	General-Purpose Graphics Processing Unit
GPP	General-Purpose Processor
GPU	Graphics Processing Unit
GpuCV	Graphics Processing Unit Computer Vision
GTK+	GIMP Toolkit
GUI	Graphical User Interface
GVFS	GNOME Virtual File System
HD	High Definition
HDTV	High Definition Television
HDD	Hard Disk Drive
HLL	High Level Language
HLSL	High Level Shading Language
I/O	Input/Output
IC	Integrated Circuit
ICG-VA	Indocyanine Green Video Angiography
ICG	Indocyanine Green
ICGA	Indocyanine Green Angiography
iDCT	Inverse Discrete Cosine Transform
IEC	International Electrotechnical Commission
IEEE	Institute of Electrical and Electronics Engineers
ILP	Instruction Level Parallelism
IPP	Integrated Performance Primitives
ISA	Instruction Set Architecture
ISO	International Organization for Standardization
ITU-T	Telecommunication Standardization Sector of the International Telecommunication Union
JPEG	Joint Photography Experts Group
KMA	Kernel Memory Allocator

LCD	Liquid Crystal Display
M-JPEG	Motion JPEG
MC	Motion Compensation
MMIO	Memory-Mapped I/O
MMX	MultiMedia eXtension (unofficial)
MMU	Memory Management Unit
MPEG	Motion Picture Experts Group
MRI	Magnetic Resonance Imaging
NIR	Near-Infrared
NPP	NVIDIA Performance Primitives
OpenCL	Open Computing Language
OpenCV	Open Computer Vision
OpenGL	Open Graphics Library
OS	Operating System
PC	Personal Computer
PCB	Printed Circuit Board
PCI	Peripheral Component Interconnect
PCIe	PCI express
PMIO	Port-Mapped I/O
PnP	Plug-and-Play
POSIX	Portable Operating System Interface for Unix
QoS	Quality of Service
RAM	Random Access Memory
RISC	Reduced Instruction Set Computer
RTOS	Real-Time Operating System
SD	Standard Definition
SDTV	Standard Definition Television
SDK	Software Development Kit
SIMD	Single Instruction Multiple Data
SoC	System on Chip
SRAM	Static Random Access Memory
SSE	Streaming SIMD Extensions
V4L	Video for Linux
VA API	Video Acceleration API
VDPAU	Video Decode and Presentation API for Unix

VLD	Variable-Length Decoding
VLIW	Very Long Instruction Word
Xv	X-video extension
XvBA	X-video Bitstream Acceleration
XvMC	X-video Motion Compensation

VAASAN YLIOPISTO
Teknillinen tiedekunta
Tekijä: Kari Koivuporras
Tutkielman nimi: Neurokirurgisen kuvantamisjärjestelmän parantaminen PC-pohjaisella videonkäsittelyratkaisulla

Valvojan nimi: Jarmo Alander
Ohjaajan nimi: Petri Välisuo
Tutkinto: Diplomi-insinööri
Yksikkö: Sähkö- ja energiatekniikan yksikkö
Koulutusohjelma: Sähkö- ja energiatekniikan koulutusohjelma
Suunta: Automaatiotekniikka
Opintojen aloitusvuosi: 2003
Tutkielman valmistumisvuosi: 2011 **Sivumäärä:** 92

TIIVISTELMÄ:
Tässä työssä esitellään PC-pohjaisen videonkäsittelysovelluksen prototyyppi, joka on suunniteltu toimimaan tietyn neurokirurgisen kuvantamislaitteen, OPMI® Pentero™ leikkausmikroskoopin, yhteydessä Helsingin keskussairaalan Töölön yksikössä. Työn aihetta motivoi usean kliinisesti tärkeän ominaisuuden puuttuminen alkuperäisestä leikkausmikroskoopin tarjoamasta kuvantamisjärjestelmästä.

Kuvantamisjärjestelmän rooli on tukea tosiaikaista diagnostiikkaa leikkauksen aikana. Leikkausmikroskoopissa on kaksi sisäistä videokameraa: toinen tavalliseen videokuvaukseen ja toinen lähi-infrapuna-alueen angiografiaan, jossa kuva saadaan fluoresenssikuvantamisella käyttäen indosyaniinivihreää loisteaineena. Mikroskoopin kulloisenkin aktiivisen kameran kuva luetaan PC:lle komposiittisignaalina laitteen takapaneelista. Mikroskooppiin on lisäksi liitetty korkean resoluution värivideokamera, jonka kuva luetaan sen käyttämästä videokeskittimestä niin ikään komposiittisignaalina.

PC valittiin videonkäsittelyalustaksi, koska se tarjoaa hyvät mahdollisuudet sekä kokeelliseen ohjelmistokehitykseen että tehokkaaseen videokäsittelyyn. Työssä tehtiin perusteellinen katsaus PC alustan ominaisuuksiin ja tehokkaan videonkäsittelyn menetelmiin, ja näitä tuloksia käytettiin kuvantamisaseman suunnittelun pohjana. Projektin aikana käyttökelpoisiksi osoittautuneet ominaisuudet yhdistettiin GNU/Linux-jakelu Ubuntulla suoritettavaan sovellukseen. Ominaisuuksien kliininen hyödyllisyys varmistettiin keskusteluissa kuvantamisjärjestelmää käyttävien aivokirurgien kanssa.

Alkuperäisen järjestelmän suurimmat puutteet korjattiin työn aikana. Kehitetyn sovelluksen tärkeimmät toiminnot ovat: videovirran toisto, samanaikainen toisto ja tallennus sekä tallenteiden toisto käyttäen korkeintaan kahta videovirtaa. Ohjelman toistomoodi tarjoaa kattavat ohjaimet mediantoiston hallintaan, hienojakoisen ruutu-ruudulta kelauksen ja intuitiivisen, nopeasti reagoivan, käyttöliittymän. Ohjelma tarjoaa videovirroille vertailu- ja yksittäistoistomoodit. Näistä jälkimmäinen tarjoaa paremman erottelukyvyn yhdelle videolle ja ensimmäinen mahdollistaa esimerkiksi anatomisen ja angiografisen materiaalin sekä yksittäisen videon ennen–jälkeen-tyyppisen vertailun.

AVAINSANAT: videonkäsittely, PC, neurokirurgia, operaatiomikroskooppi

UNIVERSITY OF VAASA
Faculty of Technology
Author: Kari Koivuporras
Topic of the Thesis: Enhancing a Neurosurgical Imaging System with a PC-based Video Processing Solution
Supervisor: Jarmo Alander
Instructor: Petri Välisuo
Degree: Master of Science in Technology
Department: Department of Electrical Engineering and Energy Technology
Degree Programme: Degree Programme in Electrical and Energy Engineering
Major of Subject: Automation
Year of Entering the University: 2003
Year of Completing the Thesis: 2011 **Pages:** 92

ABSTRACT:
This work presents a PC-based prototype video processing application developed to be used with a specific neurosurgical imaging device, the OPMI® Pentero™ operating microscope, in the Department of Neurosurgery of Helsinki University Central Hospital at Töölö, Helsinki. The motivation for implementing the software was the lack of some clinically important features in the imaging system provided by the microscope.

The imaging system is used as an online diagnostic aid during surgery. The microscope has two internal video cameras; one for regular white light imaging and one for near-infrared fluorescence imaging, used for indocyanine green videoangiography. The footage of the microscope's current imaging mode is accessed via the composite auxiliary output of the device. The microscope also has an external high resolution white light video camera, accessed via a composite output of a separate video hub.

The PC was chosen as the video processing platform for its unparalleled combination of prototyping and high-throughput video processing capabilities. A thorough analysis of the platform and efficient video processing methods was conducted in the thesis and the results were used in the design of the imaging station. The features found feasible during the project were incorporated into a video processing application running on a GNU/Linux distribution Ubuntu. The clinical usefulness of the implemented features was ensured beforehand by consulting the neurosurgeons using the original system.

The most significant shortcomings of the original imaging system were mended in this work. The key features of the developed application include: live streaming, simultaneous streaming and recording, and playing back of upto two video streams. The playback mode provides full media player controls, with a frame-by-frame precision rewinding, in an intuitive and responsive interface. A single view and a side-by-side comparison mode are provided for the streams. The former gives more detail, while the latter can be used, for example, for before-after and anatomic-angiographic comparisons.

KEYWORDS: video processing, PC, neurosurgery, operating microscope

1. INTRODUCTION

Neurosurgical operating microscopes are devices designed to provide better visualization of the operated area. By employing good lighting, magnification and other optical effects, microsurgical operations can be performed with more precision by the operating surgeon, who looks at the scene through oculars. Many modern operating microscopes are also often equipped with integrated video cameras, allowing the utilization of digital video processing. (Uluç, Kujoth & Başkaya 2009; Yaşargil 1984: 209–213.)

The operating microscope (OPMI® Pentero™, Carl Zeiss Co, Oberkochen, Germany) used in the target operating room has two integrated video cameras offering a 720×576 output resolution. One of the cameras is used for imaging in visible and the other in Near-Infrared (NIR) wavelengths. The white light camera is intended for capturing anatomical footage, while the NIR band is reserved for a fluorescence-based blood vessel imaging (*angiography*) scheme, tailored for the Indocyanine Green (ICG) fluorescent dye. This particular method of intraoperative angiography, marketed by Carl Zeiss Meditec as INFRARED 800™, is called Indocyanine Green Videoangiography (ICG-VA). (Dashti, Laakso, Niemelä, Porras & Hernesniemi 2009; Carl Zeiss Meditec 2009; Carl Zeiss Meditec 2008.) In addition to the internal cameras, the imaging system includes a High Definition (HD) color video camera, which uses the optics of the microscope and images at 1920×1080 precision.

Although advanced, the current imaging system lacks several functionalities considered clinically important by the operating neurosurgeons. The goal of this thesis is to provide implementations for these missing features. At the moment the capabilities of the system are live display of the HD and either the white light or ICG footage, plus recording from the microscope-integrated cameras. Based on the interviews of the operating neurosurgeons, the limitations of the current system include: no online pause or rewind possibility for the ICG video, no way to compare angiographic imagery before and after surgical operations, no way to simultaneously visualize anatomic and angiographic material, and no video post-processing.

The added features will be incorporated into a PC-based video processing application used in the operating room as a supplementary tool during surgery. As the application is an extension to an online imaging system, a reactive and smooth operation is important.

This makes efficient video processing a key design goal. On the other hand the design needs to support fast prototyping and deploying of new features.

1.1. Technical background and project guidelines

Video processing is a demanding application area, where dedicated hardware solutions, like Field Programmable Gate Arrays (FPGA), Digital Signal Processors (DSP), media processors and System-on-Chips (SoC), are commonly used and in some cases required to achieve acceptable performance levels. These platforms typically use a time-consuming, low-level software development model, which is ideal for pushing the last bit of performance out of the hardware but not quick prototyping. (Ge 2008; Nair 2008 ; van der Wolf & Henriksson 2008; Kuroda & Nishitani 2002.)

One advantage of the PC platform is a faster software development cycle compared to dedicated hardware solutions. This is due to the advanced programming environment and use of high-level languages and application programming libraries used in PCs (Kehtarnavaz & Gamadia 2006: 33). The PC platform also offers extensive support for building complex Graphical User Interfaces (GUI), an essential feature for implementing modern interactive application. The weak point of PCs in efficient video processing has traditionally been the lack of parallel processing power.

In the past, many demanding PC video processing applications have had to rely on expansion cards utilizing dedicated hardware technologies in order to achieve sufficient performance (Kuroda & Nishitani 2002). Today, the increase of media processing capabilities and core count of the Central Processing Units (CPU) as well as the rapid development of the consumer-level Graphics Processing Units (GPU) have made many computationally demanding applications possible, even on standard PC hardware (Kehtarnavaz & Gamadia 2006: 39–41). Accelerating various video processing tasks in modern GPUs has also been made much easier by programmability enhancements and addition of dedicated video processing engines (Owens, Luebke, Govindaraju, Harris, Kruger, Lefohn & Purcell 2007; Pieters, Van Rijsselbergen, De Neve & Van de Walle 2007).

This work will thoroughly review the key methods for efficient video processing and the potential provided by the hardware and software platform of the PC. This is necessary to be able to make informed choices when building the hardware configuration for the

imaging station PC and choosing the necessary software frameworks for the application. The feasibility of each proposed features will be tested before freezing the final feature set, and thus hardware requirements, of the application. This way the potential performance issues can be addressed in time by reviewing the hardware or software architecture. Most of the development work will be done using alternative live stream sources like webcams and recorded video material from the operating microscope, as the actual hardware is rarely available for on-site testing.

On the software side, this project will rely on open-source components whenever possible. The main reasons for this are the ability to freely adjust existing software to the needs of the project, freedom of licensing fees and better control of the whole software stack. For these reasons a Linux-based operating system is an obvious choice. Ubuntu will be the Linux distribution used in the project because it is a well-documented, popular and is based on solely free and open source[1] software (Canonical 2010).

1.2. Related work

As ICG-VA is a relatively new method, there is still a lot of engineering work to be done to fully harness its potential (Alander, Pätilä, Spillmann, Laakso, Kaartinen, Huotari & Tuchin 2011). Only a few operating microscope manufacturers provide integrated ICG imaging capability and the feature set of these systems is often limited to viewing live footage and recording for post-operation viewing (Carl Zeiss Meditec 2008; Leica microsystems 2008).

Even though the feature set of microscope-integrated imaging systems is quite limited for the time being, the situation might change in the future, given that these systems are still quite young. Carl Zeiss, for one, recently introduced a separately sold blood flow dynamics visualization software, FLOW® 800[2], for the INFRARED 800. FLOW 800 is an example of extracting additional information from the ICG angiograms. It uses the video sequences produced by INFRARED 800 to generate visual presentations of the temporal changes in blood flow. FLOW 800 also allows intraoperative side-by-side comparison of fluorescence imaging sequences to visualize blood flow changes – a feature highly usable for before and after imagery. (Carl Zeiss Meditec 2010.)

[1] Some restricted video codecs and proprietary display drivers may still be used in the project
[2] FLOW 800 is not currently purchased for the unit at HUCS

In the literature there are a number of publications concerning the clinical feasibility and practicality of the ICG-VA method (Dashti, Laakso, Niemelä, Porras, Celik, Navratil, Romani & Hernesniemi 2010; Fischer, Stadie & Oertel 2010; Dashti et al. 2009; de Oliveira, Beck, Seifert, Teixeira & Raabe 2008; Imizu, Kato, Sangli, Oguri & Sano 2008; Takagi, Kikuta, Nozaki, Sawamura & Hashimoto 2007). On the other hand, publications handling post-processing of ICG angiograms are scarce and concentrated on applications for ophthalmology (Rosen, Hathaway, Rogers, Pedro, Garcia, Dobre & Podoleanu 2009; Kohno, Miki, Shiraki, Kano, Matsushita, Hayashi & De Laey 1999). Most post-processing methods also deal with still images instead of live videos.

The literature on PC-based video processing systems is quite extensive. A few interesting categories on CPU-based processing systems include: various real-time PC video processing systems (Lee & Chang 2008 ;He & Zhang 2006; Di Stefano, Marchionni & Mattoccia 2004) and optimizing CPU-based processing with instruction set extensions (Taylor 2007; Landré & Truchetet 2007; Ranganathan, Adve & Jouppi 1999). The main interest in recent years, indicated by a large number of publications, has been utilizing the GPU for video processing tasks. Some of the most popular categories include: GPU-implementations for common processing algorithms (NVIDIA 2010b; Gómez-Luna, González-Linares, Benavides & Guil 2009; Allusse, Horain, Agarwal & Saipriyadarshan 2008; Babenko & Shah 2008), performance benefits for using the GPU over the main CPU (Bui & Brockman 2009; Gong, Langille & Gong 2005; Sugita, Naemura & Harashima 2003), architecture overviews (Owens, Houston, Luebke, Green, Stone & Phillips 2008; Owens et al. 2007) and evaluations of GPU processing frameworks (Luebke, Harris, Krüger, Purcell, Govindaraju, Buck, Woolley & Lefohn 2004; Owens et al. 2007; Yang, Zhu & Pu 2008).

1.3. Thesis structure

This thesis will begin by presenting the background of the working environment and the relevant medical methodology in Chapter 2. This includes an introduction to the ICG contrast agent, a description of the operating room and the work flow in it, as well as a closer look at the ICG-VA method. Chapter 3 introduces the reader to the basics of digital video, lays out some general concepts for achieving efficient video processing, and evaluates the PC as a video processing platform. Chapter 4 describes the hardware

and software platforms chosen for the imaging station and in the next the results of the project are presented. In Chapter 5, the results of the project are given, including the description of the features of the imaging application and how they were implemented as well as the documentation of other prototyping work done during the project. Finally, the conclusions chapter summarizes the whole project in a concise format , assesses how well the goals were met, and contemplates with future perspectives.

2. BACKGROUND AND WORKING ENVIRONMENT

Neurosurgery is a specialized field of medicine with a unique instrumentation and work-flow. Neurosurgeons perform microsurgical operations in a delicate environment, often from a small hole made to the base of the skull. Therefore special attention needs to be paid to achieve good visibility of the operated area and a relaxed, yet firm, working position of the surgeon. To accommodate these needs a variety of special instrumentation has been developed – the neurosurgical operating microscope perhaps being the most important. (Yaşargil 1984: 210–214.)

This chapter presents the essential background on neurosurgical instrumentation and work-flow as well as the ICG-VA method in order to create a solid understanding of what is being developed in this work and why. The chapter will begin by giving an overview of the instrumentation and working set in the HUCS neurosurgical operating room. The next section introduces the properties of ICG contrast agent and the technical implementation of angiography based on it.

2.1. Operating room environment

The aim of this section is to shortly describe the relevant instrumentation and staff in intracranial surgical operation. Since this work is only concerned by operating microscope integrated imaging solutions, other instrumentation like surgical tools are left out here. While the description is mostly kept on a general level, the equipment specific to the target operating room are explained when necessary. Literature references are given whenever possible; the rest of the descriptions and remarks, for example, on hardware capabilities and operating room organization are based on interviews of the Töölö Hospital surgeons.

2.1.1. Surgery staff

The staff of a typical neurosurgery consists of an operating and assisting surgeons, anesthesiologist and his assistant, and the scrub nurse and a few circulating nurses. All of these people must have basic knowledge of the the work-flow and equipment in the operating room so that valuable time is not lost during surgery. (Yaşargil 1984: 213–214.)

The roles and number of representatives of each group may differ between surgical teams, but the operating surgeon is usually the one who uses the operating microscope

and does all the neurosurgical operations (Yaşargil 1984: 213–214). It is also important to note that according to HUCS neurosurgeon Aki Laakso, there is always at least one member of staff who has the opportunity to use supplementary software during surgery.

2.1.2. Neurosurgical operating microscopes

The operating microscope is primarily a visualization aid for the operating surgeon, who looks at the operated area through the microscope's optics. The main purpose of the operating microscope is to provide the best possible viewing conditions. Important features in pursuing this include good scene illumination, stereoscopic vision and magnification. (Yaşargil 1984: 209–213.)

Ideally, the operating surgeon never needs to use his hands for any other purpose than the surgical operations or remove his eyes from the eyepieces. An ideal operating microscope also maintains an optimal illumination of the field, has a suitably narrow interpupillary distance in order to see into narrow gaps, and incorporates auto-focus with an automatic determination of the desired focal plane. Since the introduction of operating microscopes to neurosurgery in the late 50's, the mechanical and optical development has tried to meet these requirements with partial success. (Uluç et al. 2009.)

Recent operating microscopes use counter-weighted arms to allow smooth adjustment of the imaging unit with very little force. Another improvement, first implemented by M.G. Yaşargil in the early 70's, was the use of microscope joint attached electromagnetic brakes, controlled by a mouth-piece on the imaging unit. The mouth-piece is attached so that it can be used while looking at the operated area through the oculars. In recent microscopes the mouth-piece allows all movements of the imaging unit to be controlled by the surgeon's head. (Yaşargil 1984: 209–213; Uluç et al. 2009.)

Latest operating microscopes also incorporate several accessories including specialized integrated imaging systems like fluorescence imaging and the possibility to insert additional imagery, for instance, from Computed Tomography (CT) or Magnetic Resonance Imaging (MRI), to the eyepieces. These operating microscopes often incorporate digital video cameras and a video processing system, allowing recording and post-processing of the produced material. (Uluç et al. 2009.)

2.1.3. Instrumentation in the target operating room

OPMI Pentero, launched in 2004, is a fairly new operating microscope optimized for neurosurgery. Its standard imaging capabilities include autofocus and color video capture at 720×576 pixel Standard Definition Television (SDTV) quality. The captured video can be shown live and recorded for later analysis. The microscope can also be equipped with two optional intraoperative fluorescence imaging modes, BLUE 400 and INFRARED 800, of which the latter is used in the target device for ICG angiography. Later a blood flow analysis software, which uses the ICG angiograms produced by the INFRARED 800, became available as another optional feature. (Wiederspahn 2004; Carl Zeiss Meditec 2009.)

OPMI Pentero consists of a stance on wheels, a computerized main unit with a touch-screen interface, and a Six Degrees of Freedom (6DoF) arm. The main unit drives a GUI in the touch-screen and houses an internal Hard Disk Drive (HDD) for white light (anatomic), BLUE 400, and INFRARED 800 recordings (Wiederspahn 2004; Carl Zeiss Meditec 2009). The imaging unit of the HUCS OPMI Pentero is also equipped with a High Definition (HD) color video camera (Image 1 HD H3-M, Carl Storz GmbH & Co, Tuttingen, Germany) providing Full HD video at 1920×1080 resolution. The hardware setup is visualized in Figure 1.

OPMI Pentero offers a single video output channel in multiple formats for attaching to external displays. These outlets always provide the video stream of the currently active imaging mode of the microscope, eliminating the possibility to access white light and INFRARED 800 video simultaneously. However, internally OPMI Pentero is able to record both channels to the built-in HDD.

The cameras for INFRARED 800 and white light appear to use different optical paths; one is taken from the left and the other from the right eye tube. The HD camera attaches to the right eye tube via the side optics slot on the imaging unit. Unfortunately, the HD camera slot remains stationary when the imaging unit is rotated. This incurs a rotational misalignment to the output images of the HD camera and the internal cameras, when the imaging unit is removed from the default position. The field of view and aspect ratio of the HD camera are also different to those of the internal cameras.

In addition to the operating microscope, the operating room contains several displays with varying sizes. All displays and the outputs of the operating microscope can be

Figure 1. The OPMI Pentero operating microscope being presented by surgeon Martin Lehečka at Töölö Hospital, HUCS 22.2.2010. The larger image visualizes the main parts of the microscope and smaller is a close-up of the imaging unit (images courtesy of Jarmo Alander, 2010).

connected to the audio and video bus of the operating room. Thus, the video from the microscope can be output to any number of displays in the room and also to a big screen in the hall of the neurosurgical department. A separate videohub (Image 1 HUB™ HD, Karl Storz GmbH & Co, Tuttingen, Germany) is used by the HD camera. The hub provides multiple output channels in both SDTV and HDTV quality. At the moment the hub is used to show the HD camera image in a large LCD TV monitor.

2.2. Introduction to indocyanine green angiography

ICG belongs to the family of cyanine dyes and it was originally developed for NIR imaging by Kodak Research Laboratories in 1955 (Björnsson, Murphy & Chadwick 1982). Use of ICG in clinical applications was approved in 1956 and the first applications for angiography appeared about ten years later (Kogure & Choromokos 1969).

2.2.1. Instrumentation and principle of angiography

Indocyanine Green Angiography (ICGA) is essentially a Fluorescence Imaging (FI) method. In FI, the fluorescing substance is exposed to a light source known to contain excitation wavelengths. This triggers a near immediate emission at slightly longer wavelengths, captured by a camera or some other imaging device (Reichman 2000: 2–7). The excitation and emission curves for a typical fluorescent dye are depicted in Figure 2. For high image quality, the excitation and emission wavelengths are usually separated into two non-overlapping bands with optical filters (Frangioni 2003).

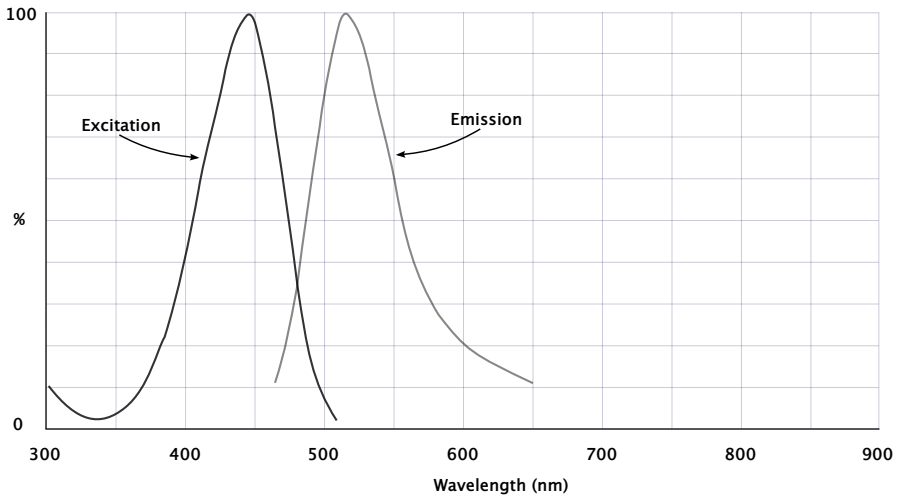

Figure 2. Typical excitation and emission curves for a fluorescent dye. Due to energy loss in fluorescence, the emission band shifts towards the longer wavelengths (Reichman 2000: 4).

In ICGA, a bolus of contrast agent is injected into the patients blood stream and the desired area is illuminated with excitation wavelengths, while simultaneously capturing the emission with a NIR sensitive camera (Raabe, Beck, Gerlach, Zimmermann & Seifert 2003). A clarifying schematic of the procedure is provided in Figure 3.

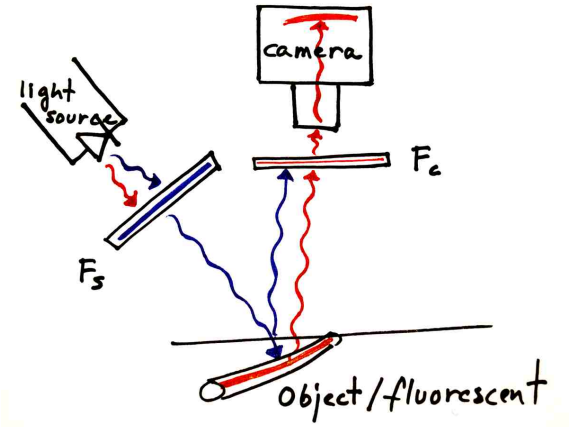

Figure 3. The principle of fluorescence imaging and indocyanine green angiography. The optical filter F_s only allows the excitation wavelengths (blue) to reach the vein (object), while F_c prevents everything outside the emission wavelengths (red) from entering the camera (image courtesy of Jarmo Alander, 2009).

Two optical bandpass filters marked F_s and F_c in Figure 3 can be used to adjust the spectral distribution of light reaching the imaged area and the camera respectively. F_c is used to ensure that only the light on the emission band enters the camera. This raises the image quality and enables NIR imaging under regular room lighting. Concurrent visible light imaging is also possible in this kind of setup. F_s, in turn, is used to shape the excitation wavelength distribution to match the purpose. This is often desired because the qualities of existing light sources are seldom satisfying from the start. Filtering also allows a wider range of light sources to be used.

The emission peak of ICG lies around 800 nm, which is in the NIR region, well beyond the visible range of human vision. However, regular silicon sensor based cameras – including those using Charge Coupled Device (CCD) and Complementary Metal Oxide Semiconductor (CMOS) sensors – can be used for ICGA, as their sensitivity stretches deep enough into the NIR region. This requires, though, that any NIR blocking filters, commonly found in normal color cameras, are removed. In addition to this, excitation and emission filters might be needed for a complete imaging solution. Still, a minimum technical implementation of ICGA is both inexpensive and simple, requiring only a few optical filters and a regular video camera. (Alander et al. 2011: 26.)

2.2.2. Clinical properties and usability

The fluorescence emission spectra of ICG is somewhat dependent on the physical and chemical environment like dye concentration, ambient temperature and the chemical properties of the solution. During its use, ICG has been proven to have some important clinical features including:

- non-toxicity,
- non-ionizing,
- short lifetime in blood circulation allowing fast re-injection,
- binding to lipoproteins in blood isolating the dye in circulation,
- operates within reach of common silicon sensors (cheap imaging devices),
- tissue has low NIR autofluorescence giving a low noise background. (Kulyabina & Kochubey 2006; Björnsson et al. 1982; Cherrick, Stein, Leevy & Davidson 1960.)

The surgical applications of ICG have emerged quite recently, despite its long history in clinical applications. Traditionally, ICG has been used for determining cardiac output, liver blood flow and hepatic function, and chroroidal angiography (Cherrick et al. 1960; Desmettre, Devoisselle & Mordon 2000). In recent years, there has been an increase in new applications using ICG, mainly related to surgery. Thus, there is still a lot of engineering and research challenges related to the use of ICG in these new fields. Some challenging points related to use of ICG include:

- invisibility to the naked eye (need of a NIR imaging device),
- need for auxiliary devices like illumination control,
- few chemical derivatives exist for more specific imaging setups,
- injection solution contains some sodium iodide (risk of allergic reaction),
- instability in solution when exposed to light,
- non-linear fluorescence quantum yield vs. concentration. (Kulyabina & Kochubey 2006; Landsman, Kwant, Mook & Zijlstra 1976.)

2.3. Microscope-integrated indocyanine green videoangiography

The first noteworthy application of ICG imaging in neurosurgery was the technique of microscope-integrated ICG-VA first described in a journal article by Raabe and colleagues in 2003 (Raabe et al. 2003). In ICG-VA a video camera as well as a filter and excitation illumination setups are integrated into a neurosurgical operating microscope. Several feasibility studies after the first experiments have shown considerable benefits from adding ICG-VA into routine imaging methods (Raabe, Beck & Seifert 2005a; Raabe, Nakaji, Beck, Kim, Hsu, Kamerman, Seifert & Spetzler 2005b; Takagi et al. 2007; Kumar & Friedman 2009). The neurosurgical team of HUCS have used ICG-VA since 2005 in assessment of cerebrovascular blood flow during aneurysm surgery and reported increases in the quality of treatment as a result of using this method (Dashti et al. 2009; Dashti et al. 2010).

When compared to other popular angiography methods that can be used during surgery like intraoperative Digital Subtraction Angiography (DSA), microvascular Doppler and fluoresceine imaging, ICG-VA has certain unparalleled benefits. An ICG-VA study takes only a few minutes to perform and requires minimal preparations. Due to the rapid extraction of ICG from the body, multiple ICG-VA studies can be done during the same surgery. From patient's point of view the procedure is minimally invasive. There is also no need for large imaging apparatus which take time to position and require the patient immobility for good quality images. Side effects from ICG are rare and mild. Also the daily dosage is of less concern than with radiology-based imaging methods. All these make ICG-VA a safe, quick and simple method for assessment of blood flow immediately after corrective surgical operations. (Dashti et al. 2010; Fischer et al. 2010; Dashti et al. 2009; Takagi et al. 2007; Frangioni 2003.) A still frame in Figure 4, captured from an ICG-VA study conducted with the OPMI Pentero unit at Töölö Hospital, exemplifies the high contrast attainable with the ICG-VA method when using proper instrumentation.

Aneurysm surgery[1], which is a common operation in Töölö Hospital, is a good example of a procedure, where ICG-VA has been successful used. In the surgery, the blood flow into an aneurysm sack is occluded with a metallic clip. In these operations ICG-VA helps by allowing immediate assessment of blood circulation after clip setting. This is considered especially important, since corrective operations resulting from an unsuccessful

[1] Aneurysms are dilatations in arteries having an increasing risk of rupture over time (Nienstedt 2006: 31)

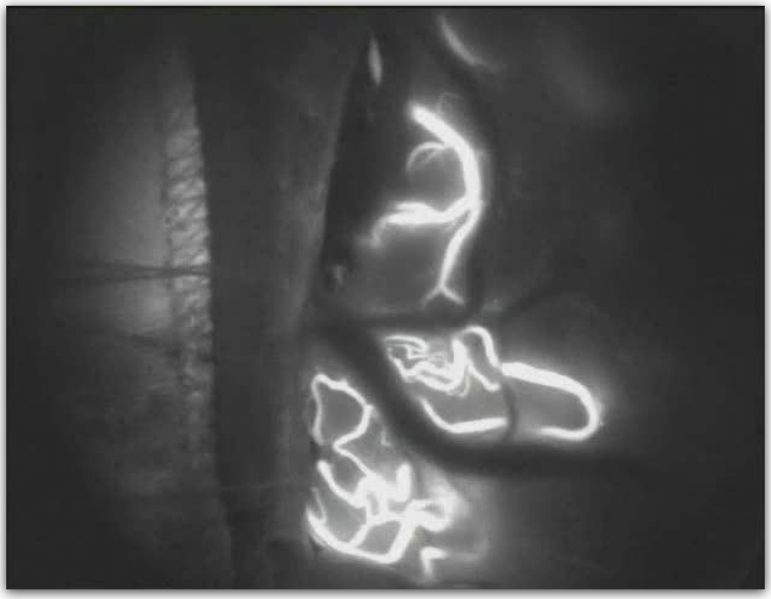

Figure 4. A still frame captured from an ICG videoangiogram. The video was recorded during an operation using the OPMI Pentero unit at Töölö Hospital (image courtesy of Martin Lehečka, 2009).

aneurysm clipping are often too late, when discovered in post-operative imaging (Dashti et al. 2009). Possible filling of the dilatation, as well as unintended occlusion of surrounding blood vessels, are usually clearly visible in the videoangiogram within a few minutes from corrective measures.

3. EFFICIENT VIDEO PROCESSING ON PERSONAL COMPUTERS

Video processing, be it post-processing or tasks related to playback and storage, requires a lot of computation and memory resources. Because these tasks typically suit well for parallelization, they are more efficiently handled by dedicated processing engines than general-purpose processors. Thorough insight on efficient video processing is vital in order to implement a responsive video processing application, especially with today's high resolution videos. This chapter focuses on the hardware and software architectures that may aid in delivering fluent playback and post-processing on the PC platform.

The first section is dedicated to establishing the basics, that is, how computers store image and video data and how that data is transformed and transported between systems. The next section reviews the most common methods for raising video processing throughput on a general level. The last section in this chapter is dedicated to reviewing the strong and weak points of the PC platform in efficient video processing.

3.1. Presentation of digital image and video data

Digital images consist of a two-dimensional array of pixels and can be presented in three different forms: symbolic image description, raster image and compressed image. Raster images store each pixel's color directly. The other two need additional processing to produce a raster image, which is the only suitable form for most displays. (Poynton 2003: 31–42.)

There are four commonly used raster image formats based on color coding: bilevel, grayscale, truecolor and indexed color. Bilevel images only have black and white colors and thus take one bit per pixel. In grayscale (*monochrome*) images, pixels are usually coded with 8 bits or more and can only present different shades of grays as the name implies. To present photographic-quality color images, the truecolor format is a minimum requirement. Truecolor images use 8 bits per each primary color channel – Red, Green, and Blue (RGB) – resulting in about 16.7 million distinct colors. Indexed color images use a subset of the truecolor color set using less bits per channel and typically 8 bits per pixel. (Poynton 2003: 31–42.)

In modern 32-bit computer architectures, the graphics hardware often uses 32 bits per pixel in truecolor mode because of the speed advantage of aligning memory according to

the word length of the architecture. In such formats the color coding itself is identical to the truecolor specification, but the use of the extra 8-bit channel varies. In modern PCs the extra channel is often used in image composition to convey translucency information. The channel is commonly called the alpha[1] channel and the format RGBA. (Wikipedia contributors 2010c.)

Videos are essentially a sequence of images and thus share the same color coding and presentation principles. The illusion of motion in videos is created simply by displaying a sequence of still images with a sufficiently high rate. Usually 24 frames per second (fps) is enough, although the properties of the environment and the vision of the spectator have a small impact. To save bandwidth, a color coding technique called chroma subsampling is more commonly used in video encoding than that of still images. In chroma subsampling the color information (chroma) is coded with less precision than luminance (luma) without a significant hit in the perceived color accuracy. This is possible because the human visual system has a lower acuity for color differences than luminance. In practical implementations, chroma subsampling is implemented as a set of luma-chroma color spaces, where the amount of color information degradation and compression varies. (Poynton 2003: 51–64.)

3.1.1. Video and image compression

Directly storing or transmitting digital video requires a large data capacity – for example, playing back an uncompressed 1920×1080 Full HD video at 24 fps and in the RGB space, requires transferring roughly 150 megabytes per second. This is why compression is required for economical use of video data (Poynton 2003: 51–64). This section introduces the principles of compression and some of the most common image and video compression schemes, like Joint Photography Experts Group (JPEG), Motion Picture Experts Group (MPEG) and Motion JPEG (M–JPEG).

Generally, compression means reducing the number of bits required to store or convey data. This can be done in either lossy or lossless manner. In lossless compression, the data is exactly recoverable by a decompression algorithm, whereas in lossy compression, some data is sacrificed for a higher compression ratio. Compression, in general, exploits

[1] Alpha refers to alpha blending, which is a method for producing partially transparent images (Porter & Duff 1984).

the statistical properties of data. Additional compression is often possible by utilizing the perceptual properties of the human vision, allowing particular parts of data to be discarded without a significant perceptible difference. (Poynton 2003: 117–126.)

Many compression methods for grayscale and truecolor images include lossless Discrete Cosine Transform (DCT) and lossy methods based on perceptually important information. One of the most common lossy still image compression methods is JPEG. Compressing color images with JPEG usually involves using DCT and a transformation from the RGB colorspace to a chroma subsampled one. (Poynton 2003: 117–126.)

Motion-JPEG (M-JPEG) uses JPEG algorithm to compress motion video. The compression is done by simply applying JPEG compression on each frame individually. This means that temporal coherence of frames is not exploited in compression, which lowers the compression ratio. On the other hand this compression method enables M-JPEG video sequences to be easily edited, because each frame can be processed separately. (Poynton 2003: 117–126.)

Further compression is usually possible by taking advantage of the fact that successive frames in a video are highly correlated; if this was not the case spectators would have a hard time following the events in videos. The MPEG codec family for example uses this scheme of compression. Compared to M-JPEG, the compression ratio can be increased by a factor of 5 to 10 by exploiting the temporal redundancy.

MPEG uses a limited number of self-contained key frames and only stores pixel differences between successive key frames. Motion estimation algorithms are also used to minimize differences between successive frames. The encoder divides the frame into blocks, calculates motion vectors for each block, and uses these vectors to calculate the differences. The decoder is able to reconstruct the frames based on the differences and the motion vectors. This is only the principle of how MPEG coding works (including MPEG–1 and MPEG–2). Yet, it is easy to see why processing MPEG streams is more complicated than that of M-JPEG streams. Before any image processing can be applied to the frames in MPEG streams, they need to be reconstructed by a complex algorithm. Hence, the encoding and decoding latency is often significant with MPEG and other methods using temporal coding, which may limit applicability in real-time applications. (Poynton 2003: 117–126.)

3.1.2. Video formats, containers and codecs

A video format describes the presentation of the contained data. It defines the color model, a progressive or interlaced scanning mode, and the possible compression scheme. Common video formats are usually internationally agreed standards, laid out by organizations like the Telecommunication Standardization Sector of the International Telecommunication Union (ITU-T), International Electrotechnical Commission (IEC), International Organization for Standardization (ISO) and Institute of Electrical and Electronics Engineers (IEEE).

A codec is a software or hardware implementation of the encoding and decoding algorithms for a particular format. Codecs are implemented by many parties, some of them proprietary, some free or open source. (Fallon, de Lattre, Bilien, Daoud, Gautier & Stenac 2002–2004: 3.) Currently, one of the most commonly used video codec standards is the ITU-T H.264/MPEG–4, which is technically identical to ISO/IEC MPEG–4 Advanced Video Coding (AVC).

When a video stream is stored into a file, a container layer is added. Most containers support multiple formats of Audio and Visual (AV) streams, described in the header and stored in the payload section of the file. The header section provides information about the payload section like the stream count, bitrates and format details. (Fallon et al. 2002–2004: 3.) Some popular containers and the video codec standards they support are presented in Table 1.

Table 1. Some AV containers and the video codec standards they support (Wikipedia contributors 2010a).

Container	**RealVideo**	**Theora**	**AVC/MPEG–4**	**H.264/MPEG–4**	**VC–1/WMV**
QuickTime	?	✓	✓	✓	✓
AVI	✗	✓	✓	Partial	✓
Matroska	✓	✓	✓	✓	✓
MP4	✗	✗	✓	✓	✓
MXF	✗	?	✓	✓	✓

3.2. Essential features for high-throughput video processing systems

Processing of digital video requires both a significant amount of computation and memory resources. Both of these requirements relate to the high amount of data digital images involve. To face these challenges any system needs a hardware architecture that exploits parallel processing blocks. Secondly a timely processing of the data is useless if one cannot move the operands and results fast enough to and from the processing unit; therefore a fast memory link is needed. (Kehtarnavaz & Gamadia 2006: 34–35.)

In addition to the work of post-processing algorithms altering the content of videos, significant computational work is needed by the processing related to video presentation transformations. When compression, decompression, color space transformations and the like are taken into account, the gravity of the computational requirements of the whole process is exposed. In order to cope with these challenges, any processing system needs a substantially parallel hardware architecture combined with software frameworks that are able to effectively utilize the capabilities of that hardware (Kehtarnavaz & Gamadia 2006: 1–3, 55).

3.2.1. Parallel processing

The principal operation of all processors is similar; they are controlled via programs consisting of a sequence of processor-dependent instructions. All instructions form an instruction set that gives a palette of simple tasks a processor can perform. There are two fundamentally different processor architectures based on their instruction characteristics: Reduced Instruction Set Computers (RISC) and Complex Instruction Set Computers (CISC). The RISC approach favors a small set of simple and fixed-length instructions whereas CISC does the opposite having a larger set of dedicated instructions for more complex tasks. The instruction set also has a big impact in how parallelization strategies can be utilized. (Granlund 2004: 139.)

Computationally, parallelism can be divided into two categories: Data Level Parallelism (DLP) and Instruction Level Parallelism (ILP). These two concepts can be considered complementary in a way that by increasing DLP, the ratio of operands is increased in the instruction stream, while increases in ILP contribute to a higher ratio of instructions. In practice increasing DLP often means applying the same instruction to a large data set

at the same time. Increasing ILP in turn means executing more instructions simultaneously. (Kehtarnavaz & Gamadia 2006: 1–5.)

On a general level, image and video processing operations can be categorized into low, intermediate, and high level operations. In this division, low level operations process image data pixel by pixel or within a pixel neighborhood and produce modified images as output. Intermediate level operations extract features from pixel data to produce a reduced data set like segmentation or contour information. The output of intermediate level processing is then exploited by high level processing, which interprets the abstract data. While DLP is crucial in low level video and image processing operations, its importance decreases as the amount of data is reduced towards higher processing levels, where ILP in turn can be exploited. (Kehtarnavaz & Gamadia 2006: 1–5.)

3.2.1.1. Increasing instruction level parallelism

Processors consist of specialized blocks called functional units each doing their own task, instructed by a control unit. The processing of an instruction takes several smaller steps or stages like fetching the instruction, doing arithmetics and storing the results. Many traditional processors fetch and execute instructions sequentially one after another. This is not very efficient, however, because the majority of functional units are idle most of the time. An effective and widely used solution to better utilize the hardware is to use instruction pipelining. Instead of waiting for the whole instruction to be finished executing, a new instruction is loaded as soon as the previous finishes the first pipeline stage. If the pipeline works ideally, one instruction reaches the execution stage on every clock cycle. This leads to an ILP gain proportional to pipeline depth. Unfortunately interrupts, branching, varying micro-operation execution times and other complications cause delay and thus decrease the performance gain. (Lapsley, Bier, Lee & Shoham 1996: 99–109.)

The chosen instruction set type also effects how pipelining can be implemented. Traditionally, RISC instructions have been more suitable for pipelining, as they are fixed-length and have less variance in execution time. CISC instructions are often more difficult for pipelining, because of their complexity. Nonetheless, pipelining has been applied to CISC processors as well; for example, in some CPU architectures used in PCs, CISC instructions are first decoded into RISC-like micro-operations and then scheduled for a pipeline (Stallings 2009: 506–539, 600). In general, designing a high-throughput pipeline

is a complicated task for any architecture and trade-off situations are common (Lapsley et al. 1996: 99–109).

Instruction parallelism can be further increased by taking multiple independent pipelines into use. This is implemented in the superscalar architecture. In superscalar pipelining, dependencies between instructions, intended to be executed in parallel, are resolved dynamically during execution. The difficulty with superscalar pipelining is to determine what instruction should be fetched into what pipeline next to achieve the best overall throughput. (Stallings 2009: 506–539.)

Another technology for increasing ILP is the Very Long Instruction Word (VLIW) architecture, where multiple instructions and operands are packed into a single instruction word. In contrast to superscalar architecture, VLIW resolves instruction dependencies and performs the scheduling statically during compile time. This approach transfers the complexity of producing an optimal scheduling from silicon to compiler design. (Talla, John, Lapinskii & Evans 2000; Granlund 2004: 150–151.)

3.2.1.2. Increasing data level parallelism

Single Instruction Multiple Data (SIMD) is one of the most important solutions in increasing DLP. It embodies simultaneous execution of the same instruction on many data operands. A form of SIMD is packed data processing, also known as subword parallelism, which is available in many processor architectures including the CPUs used in desktop PCs, DSPs, and media processors (Kehtarnavaz & Gamadia 2006: 34). In packed data processing a single instruction can load an array of operands into specific SIMD register regions and execute arithmetic operations on them (Talla et al. 2000). This kind of vector-arithmetics is especially useful in low-level video and image processing operations (Kehtarnavaz & Gamadia 2006: 34).

One of the biggest challenges in using SIMD computation is the difficulty of implementing automatic data vectorization, which means producing suitably aligned data ahead of time, for loading the SIMD registers. Thus, the performance increase for using the SIMD calculation hardware can be considerably lower than the theoretical maximum. (Talla et al. 2000; Talla, John & Burger 2003.)

3.2.2. Memory subsystem

Most computer systems use a hierarchical memory subsystem consisting of different kinds of memory chips having varying latency, capacity, and cost. The memories are connected by internal and external buses. Although some common principles apply across platforms, the organization of the memory hierarchy varies greatly between systems: DSPs and other embedded solutions, for example, use different memory types and bus layouts than desktop computers.

The fastest memory types, like registers and caches, can usually be found on the processor die. These are referred to as on-chip memory. Although there is a performance incentive to make these memories large, they must be kept reasonably sized because of die size and power consumption issues. The second fastest level is usually called the main or internal memory, which on the PC is often Dynamic Random Access Memory (DRAM). The two aforementioned memory levels are directly connected to the memory bus of the processor. The slowest and cheapest memory types, classifiable as external memory or secondary storage, are usually accessible through an external bus and possess the greatest capacity. Hard-drives and optical medias are examples of such memory. (Stallings 2009: 69–81, 96–103.)

The slowest link between different levels of the memory hierarchy determines the maximum performance when a data stream is transferred through the system. In PCs, the performance of the main memory is critical, because it serves as an I/O interface for the data: the input stream is copied and stored in main memory, then processed and finally rendered on screen or output elsewhere. (Hennessy, Patterson, Goldberg & Asanovic 2003: 428.)

3.2.2.1. Main memory architectures

Main memory architectures can be divided into two groups based on the number of parallel memory buses. In the von Neumann architecture, a single set of control, address, and data buses is used to access memory, whereas the Harvard architecture includes two or more sets of buses. Obviously, the latter provides higher performance, because multiple memory accesses can be made simultaneously, when only one per clock cycle is possible in von Neumann designs. Harvard architecture is common in high-power em-

bedded processors and DSPs, while von Neumann is used for example in desktop CPUs. (Lapsley et al. 1996: 49–55.)

A common method for increasing memory throughput is to use caching between the processor and the main memory. A cache is a small and fast block of memory, usually residing on the processor die, that holds copies of recently used instructions and data for quicker access by the processor. When the processors needs to access memory, the cache is first checked, and only in case of a cache miss, the data is actually fetched from the slower memory and copied into the cache. Caches also take advantage of the locality of memory accesses. It has been shown, that memory locations near the previously fetched are very likely to be referenced soon, because of the sequential nature of program code over short periods of time. So to further improve performance, cashes copy a block of memory, called a cache line, instead of just the desired data. The simplest cache implementations use a single cache level, but more can be added. (Stallings 2009: 103–121.)

Optimizing cache designs is an on-going and complex research problem. One fundamental difficulty in cache design is choosing an optimal cache line width, which is affected by the proportions of sequential, conditional, and branching instructions. Another difficulty, related to multi-level caches, is coherence between cache levels and main memory in shared memory multi-processor architectures. The problem is that processors or cores see different versions of the data at the same time, due to intermediate memory hierarchies causing delay in the propagation of changes. When caches are used, it is difficult to give guarantees for memory access times, because the occurrence of cache misses cannot be accurately predicted (Stallings 2009: 103–121). This is also one of the reasons, why multilevel caching may not be desirable in hard real-time systems. Caches also require a relatively large die area and have a high power consumption (Puaut & Pais 2007; Avissar, Barua & Stewart 2001). Instead of regular cashes, some embedded solutions favor scratchpad memories, which are basically software controlled single-level caches. Scratchpad memories are usually on-chip Static Random Access Memory (SRAM), mapped at a predefined address range in the processor's address space (Puaut & Pais 2007; Avissar et al. 2001).

3.2.2.2. Buses and I/O

Buses are used to provide a shared access to the resources and devices on the system. Buses with different protocols and speed ratings can be connected with bridges. The latency inflicted by busses and bridges is often significant and poses additional bottlenecks into the system. (Stallings 2009: 67–88; AMD 2002.)

Methods for managing data transfers between different parts of a computer system depend on the communicating parties and nature of data. For example, software driven I/O (polling) and interrupt driven I/O are the two principal methods used to keep the CPU up to date with the status of an I/O device. In polling the CPU synchronously inquires a device about its status. A more efficient solution is to set up the I/O devices to raise interrupts to notify the processor of an event. (Williams 2006: 46–48; Thompson & Thompson 2003: 21.)

For bidirectional data exchange between the processor and an I/O device there are two methods called Memory-Mapped (MMIO) and Port-Mapped I/O (PMIO). In memory-mapping the private address space of a device replaces an area of the main address space of the processor. This allows the processor to use the I/O memory as any main memory area. In PMIO the address space of the device is kept private and the processor needs to use special instructions to access the device. (Williams 2006: 46–48; Thompson & Thompson 2003: 21; Murdocca & Heuring 2000: 319–326.)

Furthermore, the data transfer between main memory and I/O devices can be handled using a dedicated circuitry, called Direct Memory Access (DMA). DMA enables devices to exchange data with memory or with each other without processor's active involvement. The transfer is conducted by a DMA controller and the processor only initiates the transfer and possibly gets notification of its completion. The benefit of this is that the main processing unit is free to do other work while data is being transfered. (Williams 2006: 46–48; Thompson & Thompson 2003: 21; Lapsley et al. 1996: 64–65.)

3.2.3. Software design for performance

A high performance hardware is useless without software solutions utilizing the potential of the hardware. In other words, software design determines to what degree the performance of the hardware is actually realized. Principles in this section apply to all parts

of the software stack and are meant as general guidelines for handling video data and implementing video processing algorithms. Designing software for performance is not a simple task though. Successful software design involves areas like identifying and exploiting parallelism in the processing algorithms, intelligent memory management and utilizing software optimization methods. Thorough knowledge about the underlying hardware is also vital in order to effectively utilize the capabilities of each platform. (Kehtarnavaz & Gamadia 2006: 55–79.)

The first step in a software project is choosing a programming language that best fills the performance, portability and development time requirements of the application. As a rule of thumb, more performance can be extracted from a processor, if a low-level language rather than a high-level is used. Although modern compilers are able to produce fairly optimized code, they usually cannot match an experienced programmer's insight. In low-level languages, like variants of assembly languages, the programmer has better control over the hardware; therefore higher performance is usually achievable, albeit with the cost of a longer development time (Hyde 1996). On the other hand, portability is often a key issue, making completely non-portable assembly applications unattractive. For these reasons, an ideal solution for a video processing application that demands good performance, would be a language that provides:

- enough high-level characteristics to allow good programming productivity,
- enough low-level characteristics to allow good optimization possibilities and control,
- and portability between platforms.

The languages which best meet these requirements are often considered to be C and C++. This notion is supported by the fact that many operating systems and other software with high-performance requirements are often implemented in C and C++.

3.2.3.1. Memory management

Memory management relates to how the image data is stored and transferred between various memories and processors. Optimizing this is especially important in image and video processing, due to the vast amounts of information being transferred. Many processing platforms, including PCs, are also memory-limited instead of compute-limited.

Memory management can be improved for example by reducing cache-miss rates, using image partitioning schemes, and utilizing DMA whenever possible. (Kehtarnavaz & Gamadia 2006: 61–64.)

Ideally, entire images could be read into the fast on-chip memories along with other vital data. However, this is often not possible, which has led to the development of various partitioning schemes. Partitioning means fetching and operating on fixed-sized portions of frames at a time. For example in the row-stripe partitioning, a few lines or rows of an image are prefetched into an on-chip buffer for faster processing. Critical instructions can also be prefetched and kept in the faster internal memory. However this level of internal memory customization is not supported by all hardware architectures. (Kehtarnavaz & Gamadia 2006: 61–64.)

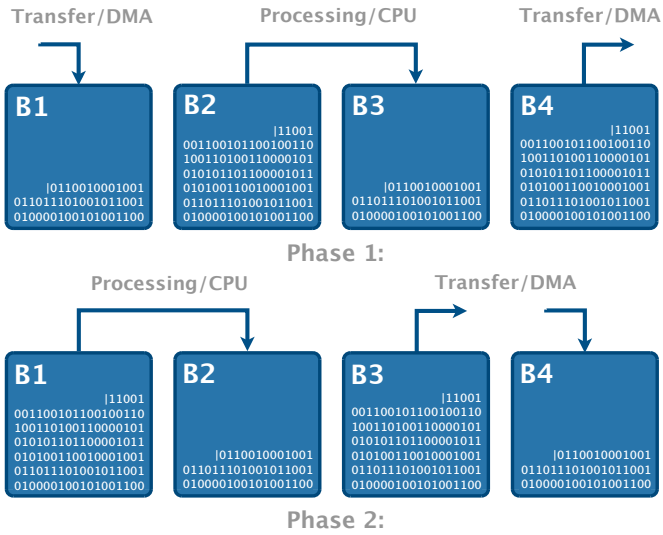

Figure 5. An example of a four-buffer (B1-B4) multibuffering scheme utilizing DMA. In the scheme the CPU processes data using two buffers in fast memory, while DMA is used to load new data from and write out results to slower memory. At the end of each phase the buffer roles are swapped.

If the system includes a DMA controller, multibuffering strategies can be used for concurrent movement and processing of data. These strategies employ a few buffers, allocated within the on-chip memory whenever possible, and a swapping logic. As an example, a four-buffer (B1-B4) swapping logic is depicted in Figure 5. In the first phase, buffer B1 is being filled with new data by a DMA transfer and the data in B2 is being processed with the results written into B3. Simultaneously, the data in B4, which contains the re-

sults from the previous cycle, is being written out by another DMA transfer. Then buffer roles are shifted to the left as shown in the lower row of Figure 5. This essentially results in a system where data transfer and processing have been parallelized. Multibuffering schemes can further be optimized by using data alignment that matches the width of the data bus. (Kehtarnavaz & Gamadia 2006: 61–64.)

In addition to optimizing the efficiency of data transfer to and from external memory into on-chip memory or main memory, one can also take measures to minimize the transferred data. One way to do this is to take the spatial and temporal locality of pixels into account. This can be done, for example, by using pixel-based instead of image-based processing. In image-based processing each operation is applied separately to the whole image. This means that a slow load and store of the whole image between the processor and the main memory is done upon every processing operation. In pixel-based processing, a set of pixels is loaded into the faster on-chip memory once and all the operations are applied before storing the result in a slower memory, saving a lot of memory bandwidth. (Kehtarnavaz & Gamadia 2006: 61–64.)

3.2.3.2. Software optimization

Code optimization is knowing and using the design patterns that lead to performance increases as well as voiding those that undermine it. For example excessive branching and conditionals incur a performance penalty. Sometimes subtle changes, like using a different instruction, altering execution order or the registers used, may cut processing time significantly. Loops are usually good places to start optimization, because the execution time reduction achieved inside a loop is multiplied by the loop cycle time. This is especially important in image and video processing where nested loops are common. (Kehtarnavaz & Gamadia 2006: 66–71;Chang, Mahlke & Hwu 1991.)

Because each hardware platform and instruction set is different, the location of bottlenecks may vary even with the same source code. The process of finding those portions of code posing bottlenecks is called software profiling. Profiling can be done by timing different portions of code or individual functions to see which of them takes the most CPU time. Many existing profiling tools can be help in this task. Once the critical sections have been identified the code is refactored until the result is within acceptable limits. (Kehtarnavaz & Gamadia 2006: 66–71; Chang et al. 1991.)

When using High-Level Languages (HLL), the machine code created by the compiler cannot be dictated to the smallest detail in source code. Luckily modern compilers are equipped with automatic optimizers that can be configured to emphasize different attributes like speed or size. If the performance is still not acceptable, the next step is to replace the most critical parts by inline assembly or use processor specific intrinsics. Intrinsics are special HLL functions that the compiler translates into specific assembly sections giving the programmer similar benefits to writing assembly by hand. (Kehtarnavaz & Gamadia 2006: 66–71; Wieber Jr & Zopetti 2008.)

Pushing the last bit of performance out of the hardware with software optimization can take a lot of time. Therefore, it is worthwhile to use the existing optimized libraries, which are available on many platforms. A few examples of such libraries include the Intel® Integrated Performance Primitives (IPP) for Intel desktop processors and DSPlib and Imglib for Texas Instruments' DSPs. (Kehtarnavaz & Gamadia 2006: 66–71.)

3.3. PC as a video processing platform

Unlike many systems, PC was not designed to do any specific task with a maximum efficiency; on the contrary, it was designed to host all kinds of expansion hardware and run a lot of productivity software concurrently. Hence, the maximization of the overall throughout is a priority on the PC platform in general. This is clearly visible in the evolution of both PC hardware and operating systems.

However, in recent years new application areas, like high definition video playback and real-time 3D games, have started to drive the capabilities of the PC platform into a new direction. Due to some inherent disadvantages, the PC platform is still not considered well-suited for hard real-time applications with strict latency deadlines, but many developments in both hardware and software have enabled new possibilities for efficient online video processing.

3.3.1. Hardware platform

For a time, the hardware side of the PC platform did not see significant architectural re-designs, but in recent years the situation has changed. The architecture on system level, that is, how the discrete chips make a system, is changing into a leaner and more

integrated environment. This section presents the most important features and trends of the modern PC assembly, related to using it for efficient video processing.

3.3.1.1. System architecture

All the separate hardware units in a PC such as the CPU, DRAM banks and mass storage devices are attached to a single Printed Circuit Board (PCB) called the motherboard. On logical level, hardware units attach to different buses which in turn are connected by the chipset. The chipset is a set of Integrated Circuits (IC) for interfacing different busses that also incorporate special hardware features like DMA controllers and Plug-and-Play (PnP). In a traditional design the chipset is separated into north- and southbridges so that bandwidth-hungry devices such as the processor, main memory and graphics hardware were connected to the northbridge while the rest, including mainly I/O devices, attached to the southbridge. (Mathivanan 2003.) A generic example of a classic northbridge-southbridge motherboard layout is shown in Figure 6.

Online video processing is a memory intensive task where a stream is usually read from an I/O device, transfered to the main memory, processed by the CPU, written back to the main memory and transfered to the framebuffer memory in the GPU for rendering (Revel, Cowan, McNamee, Pu & Walpole 1997). This causes a lot of traffic on the system bus, also know as the Front Side Bus (FSB), which quickly poses a bottleneck when images become larger. Another issue is that the DRAM technology has not been able to keep up with the increases in CPU performance. The disparity between the speeds of these two components has only become worse with time, albeit the resulting performance hit is alleviated to some degree by caching. On the other hand, the use of multi-level caches results in unpredictable memory access latencies, which make development of hard real-time applications difficult. (Peng, Peir, Prakash, Staelin, Chen & Koppelman 2008; Burger, Goodman & Kägi 1996).

These shortcomings are being addressed in newer architectures with improvements like new and faster buses, multiple parallel DRAM channels, and better overall system organization. There is a clear trend of increasing the level of integration for better performance and energy efficiency. In modern designs, the whole functionality of the old northbridge including the memory controller, the graphics card interface, and the system bus is integrated into the CPU die (Wikström 2010). Intel's 32nm Westmere microarchi-

tecture (see Figure 7) launched in early 2010 was the first design, where a graphics chip was integrated inside the CPU package, although on a separate die. All of these changes reduce power consumption and latency due to shorter distances and reduced interface logic. (Williams 2010; Kayi, El-Ghazawi & Newby 2009)

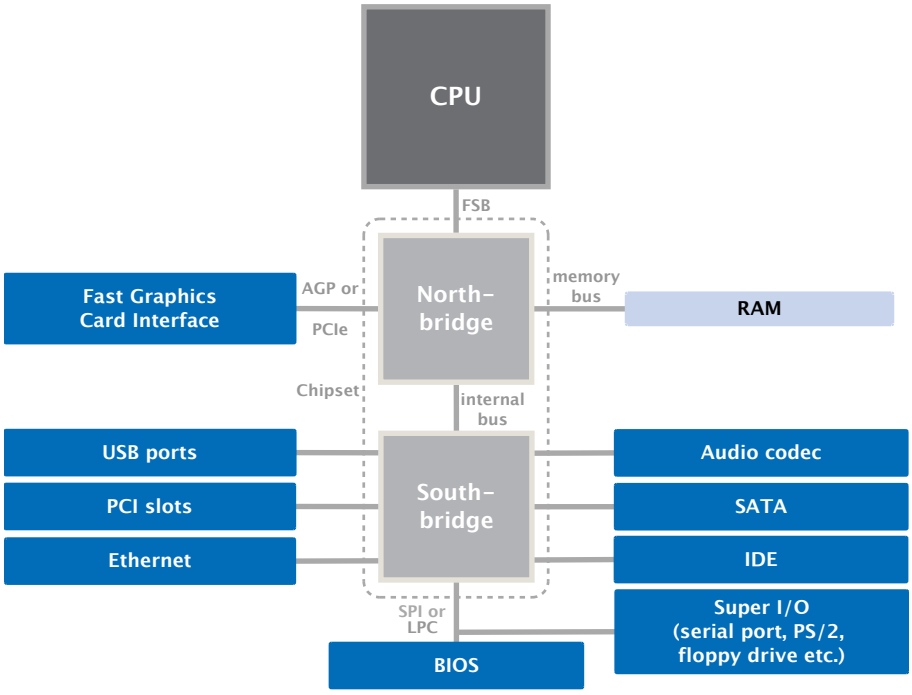

Figure 6. A traditional PC system built around a motherboard-integrated northbridge. In this kind of design the bottleneck is often the system bus (FSB) (Wikipedia contributors 2010b).

Traditionally all high-end PC systems having good graphics performance, have had to rely on separate graphics cards attached to an external bus. Although this architecture does not show signs of passing away, there has been significant progress in the efforts to combine the potential of CPU's serial and the GPU's parallel computation powers into a single unit. For example, the upcoming Sandy Bridge microarchitecture from Intel features a basic GPU with a dedicated media processing engine integrated onto the CPU die (Shimpi 2010). AMD is also shortly releasing its first Fusion™ family Accelerated Processing Units (APU), which aim to provide fully featured CPU die integrated graphics supporting the latest GPU innovations. In the long run these architectures are expected to provide better performance and lower power consumption, compared to yesterday's designs based on separate chips (Brookwood 2010).

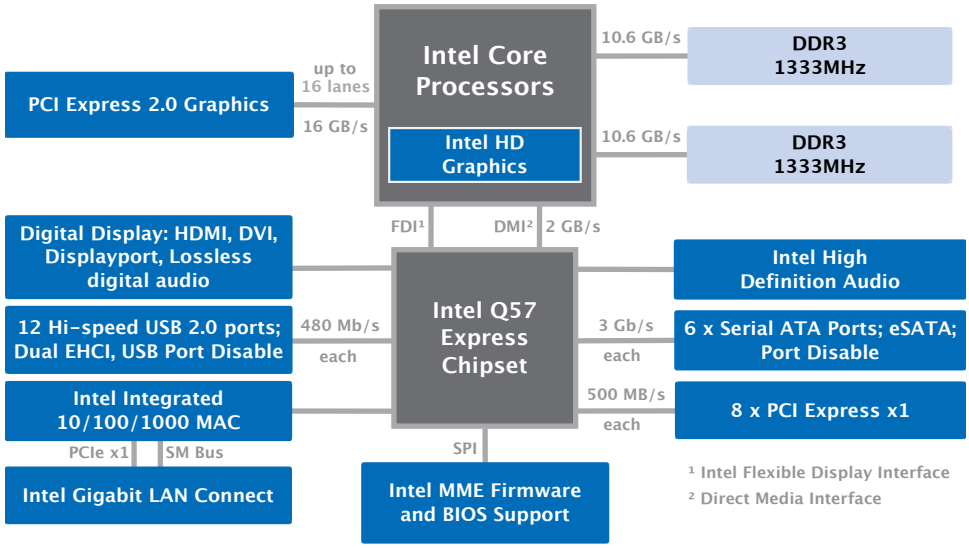

Figure 7. The layout of a modern high-end PC system based on Intel Westmere microarchitecture featuring the northbridge chips integrated onto the CPU die (Williams 2010).

3.3.1.2. Computational capabilities

As stated before, any real-time video processing platform needs a considerable amount of parallel computation power and a fast memory subsystem. PCs have traditionally not been very good in either one of these areas. The lack of parallel computation capacity started getting more and more attention in the industry in the early 1990's when multimedia and interactive 3D gaming grew in importance. Tasks like fluent video playback and rendering graphics scenes in real-time required more processing power and memory bandwidth what PCs at the time could offer. Increasing effort has been seen in this area from mid-1990's when the first SIMD Instruction Set Architecture (ISA) extensions for CPUs and early versions of special-purpose 3D graphics hardware started to emerge (Kehtarnavaz & Gamadia 2006: 10–11, 39–41.).

The CPUs based on x86-64 architecture, which dominate the PC processor market today, are general-purpose superscalar Chip Multiprocessors (CMP)[2] with a good performance in control intensive code, due to various innovations raising their ILP throughput. The X86 processor family has later been patched with an array of ISA extensions which

[2]CMP denotes multiple processor cores with dedicated and shared cache levels on a single silicon die (Peng et al. 2008).

provide a moderate SIMD calculation capability. The extensions include for example MMX, 3DNow!, Streaming SIMD Extensions (SSE) family, and Advanced Vector Extensions (AVX). Despite the improved vector calculation capabilities, the desktop CPU is still not sufficient for the most demanding video and graphics processing work. Many of these tasks have been offloaded to a dedicated graphics processor, the GPU, in modern PCs. The evolution of GPUs has been rapid in recent years and their transistor count, for instance, has exceeded that of the CPUs. (Stallings 2009: 505–517; Hyde 2001: 229–235, 262–263.)

Traditionally GPUs implemented deep graphics pipelines consisting of a chain of mostly fixed massively parallel hardware blocks. However, this approach turned out to have serious load balancing issues and was eventually replaced with the unified shader architecture. This architecture uses an array of parallel streaming multiprocessors, which can perform any type of operation in the pipeline. This allows parallelization and better load balancing of pipeline phases like geometry, vertex and raster operations, which all use the same instruction set unlike in previous architectures. Modern GPUs may include hundreds of parallel cores. Each core is usually hardware multi-threaded and contains several Arithmetic Logic Units (ALU) for floating-point arithmetics, plus optional SIMD units. This results in a very high data parallelism. (Owens et al. 2008; Do Dinh 2008; Luebke & Humphreys 2007.)

Besides better performance, the unified architecture brought the possibility of transparent execution of general-purpose calculations alongside the 3D rendering pipeline. This feature has many applications in video processing, scientific calculations, and other compute- intensive fields. The term General-Purpose GPU (GPGPU) is often used to refer to devices allowing this kind of computation. Another advantage of the unified shader architecture is better programmability of some pipeline phases, which is a result of using a unified instruction set. (Owens et al. 2007; McCool 2007; Luebke et al. 2004.)

3.3.2. Software platform

The software stack of the PC is complex, involving several layers of abstraction on top of another. This hierarchical nature of software helps application developers by hiding some of the underlying complexity, but on the other hand abstraction makes it harder to comprehend, how the system works as a whole. The overview of the PC software

stack given here, touches only the most important layers for evaluating its performance in online video processing. The software sub-systems and tasks whose performance is vital considering online video processing include at least:

- OS kernel-level process control,
- video encoding and decoding libraries,
- video post-processing libraries,
- control code handling the video streams, and
- the rendering sub-system used to present the results in a display.

3.3.2.1. Operating systems

Multitasking systems, which PC OSs usually provide, can be categorized based on how deterministic they are in terms response and execution times of various kernel operations like handling interrupts, context switches, memory allocations and task scheduling. In Real-Time Operating Systems (RTOS) the execution time is predictable in order to guarantee latency deadlines. However, for most of the tens or hundreds of processes running in a PCs, the predictable execution time is not that important. Thus, PC OSs usually implement a time-sharing design where fine-grained timing control and deterministic latency is sacrificed for better overall throughput and smoother multitasking (Stankovic & Rajkumar 2004; Bovet, Cesati & Oram 2000: 258–262.).

PC OSs usually employ elaborate priority based scheduling schemes, where priorities are automatically adjusted based on, for instance, what kind activities a process is engaged in or how much CPU time it has received recently. On some systems process priorities can also be permanently elevated, which ensures a privileged state for that process, but does not necessarily guarantee real-time performance (Bovet et al. 2000: 258–262). If even higher Quality of Service (QoS) is required from the video processing chain, often one of the RTOSs like Windows CE or real-time variants of regular PC OS kernels – mostly Unix and Linux based – should be considered (Stankovic & Rajkumar 2004).

3.3.2.2. Video processing software

Video processing can be classified into playback and post-processing related operations. In this definition playback operations account for video format conversions, like color

space transformations, deinterlacing, encoding, and decoding. The aim in these operations is to maintain the content and quality of the original video unchanged. Post-processing operations, in turn, alter the content to produce new information. Both of these categories involve computationally demanding algorithms.

A seemingly trivial task like playing back a video file in a GUI window is a fairly complex process, requiring a significant amount of control code to orchestrate the processing stages. A third category of software related to video processing can thus be called processing pipeline controllers. The purpose of the control code is to ensure that the processing runs smoothly, which may involve for instance: reservation of temporary storage buffers, synchronization and timing control, negotiation of stream formats, and invocation of the right processing algorithms in codecs (Taymans, Baker, Wingo, Bultje & Kost 2011.).

A controller is implemented in every video playback and processing application internally. Unfortunately, most of these applications are monolithic making their code not easily re-usable or extensible. Some exceptions do exist though. libVLC is example of a media pipeline controller Application Programming Interface (API), providing the playback functionality of the VLC media player for free use. GStreamer, in turn, is an example of completely generic media pipeline controller API. Unlike libVLC, GStreamer is not limited to building playback pipelines; it also supports plugging in additional post-processing algorithms (Taymans et al. 2011).

The video processing algorithms invoked by pipeline controllers are usually contained in multiple software libraries shared by the system. For example in Linux systems the open-source library libavcodec, part of the FFMPEG project, contains a wide collection of highly optimized audio and video codecs, used by nearly all the media processing applications on the platform. As the routines in libavcodec, the majority of all video processing code is still run on the main CPU (Wikipedia contributors 2011a).

For purely CPU-based video post-processing, there are few notable libraries which may help in writing custom algorithms. Open Computer Vision (OpenCV) is probably one of the most comprehensive collections of CPU-run image processing routines geared towards computer vision applications. OpenCV is open-source and able to use the optimized multimedia and data processing functions in Intel IPP as a backend. IPP functions,

which use the SIMD and other special capabilities of modern CPUs, may also be directly called from application code (Bradski & Kaehler 2008: 1; Landré & Truchetet 2007.).

In the era of high definition video content, even playback of a 1080p compressed video can be too much for a low-end CPU. Hence, more attention has been put into making the GPU do the heavy work. The two main ways for using the GPU for accelerating video processing operations are utilizing the dedicated 2D video decoding engines included in many modern GPUs and implementing custom algorithms with the GPGPU or graphics APIs. While all video processing operations do not benefit significantly from offloading to the GPU, some may experience speedups of several orders of magnitude. (Pieters et al. 2007; Shen, Gao, Li, Shum & Zhang 2005.)

Table 2. A comparison of some GPU accelerated video processing APIs listing their supported operations and formats. On the hardware level these APIs can use either the dedicated video engines or the graphics pipeline as a whole (Wikipedia contributors 2011b; Pieters et al. 2007).

Library	Platforms	Formats	Accelerated ops
VA API	Linux/Unix	MPEG–2, MPEG-4 ASP/H.263, MPEG–4 AVC/H.264, VC-1/WMV3	Variable-Length Decoding (VLD), inverse Discrete Cosine Transform (iDCT), Motion Compensation (MC) and Deblocking
XvMC	Linux/Unix	MPEG–2	MC
DxVA	Windows	H.263, H.261, MPEG–1, MPEG–2, MPEG–4 ASP	MC, iDCT, Huffman coding, color space and framerate conversion, up and down scaling, alpha blending, and deinterlacing
VDPAU	Linux/Unix	MPEG–1, MPEG–2, MPEG–4 AVC/H.264/ASP, VC–1, WMV3/WMV9	MC, iDCT, VLD and deblocking
Xv	Linux/Unix	raw RGB and YUV	color space transformations and scaling

The dedicated 2D video decoding engines support offloading some or all of the decode stages of the most common video formats in use today. Some engines also support common video quality related post-processing operations. How these features can be leveraged is system and GPU vendor specific. For example, accessing the NVIDIA's PureV-

ideo engine under Linux requires the use of a proprietary device driver and the Video Decode and Presentation API for Unix (VDPAU). On Windows, only the device driver is needed (NVIDIA 2010a). The most feature rich and GPU vendor independent video acceleration API for Unix-based OSs is the Video Acceleration API (VA API). It is a work in progress, but the aim is to accelerate an extensive set of video related tasks including encoding, decoding, blending and rendering, by using multiple backends such as VDPAU for NVIDIA and X-video Bitstream Acceleration (XvBA) for AMD GPUs (Wikipedia contributors 2011b). A collection of video acceleration APIs with a list of supported operations and formats is given in Table 2.

Table 3. A collection of frameworks that can be used for executing image processing code on the GPU. (McCool 2007; Nickolls & Dally 2010)

Developer	GPGPU API	Graphics API	Shader Language
Khronos Group	Open Computing Language (OpenCL)	Open Graphics Library (OpenGL)	OpenGL Shading Language (GLSL)
NVIDIA	Compute Unified Device Architecture (CUDA)	-	C for Graphics (Cg)
AMD	Stream SDK	-	-
Microsoft	DirectCompute	Direct3D	High Level Shading Language (HLSL)

The GPGPU APIs simplify writing of all kinds of accelerated video processing algorithms. In the literature there are many examples where the GPGPU APIs are used for implementing custom decoding, computer vision algorithms and others on the GPU. The main advantage of these APIs is that the programming concepts are much closer to CPU coding than the earlier solutions, the 3D graphics APIs and shader languages, where all computations had to be masqueraded as graphical operations. (Colic, Kalva & Furht 2010; Bui & Brockman 2009; Allusse et al. 2008; Owens et al. 2007.) For implementing GPU-run algorithms from scratch, the frameworks in Table 3 may be useful.

There are also a few libraries that provide a collection of image and video post-processing algorithms running on the GPU. These include NVIDIA's Performance Primitives (NPP), GpuCV and MinGPU. The idea of GpuCV is to provide the function palette of OpenCV, but run the operations on the GPU by using either the OpenGL graphics API and the related shader language GLSL or alternatively the NVIDIA's GPGPU API CUDA, when-

ever feasible (Allusse et al. 2008). NPP in turn provides basic image and video processing operations implemented with the CUDA framework (NVIDIA 2010b). MinGPU provides fast implementations for popular computer vision algorithms using Cg and OpenGL as backends (Babenko & Shah 2008).

3.3.2.3. Rendering models

A framebuffer is a memory area in the video memory of the graphics processing unit, consisting of multiple off-screen buffers and a write-out buffer, which contains the final outcome of the graphics pipeline, the output image. This image is continuously updated by a complex system of drawing routines run partially by the CPU and the GPU (Möller, Akenine-Möller, Haines & Hoffman 2008: 11–25). Although the organization of drawing models differ in different systems the principles are similar.

The CPU maintains a model of the scene to be drawn in the main memory. The model may consist of both geometrical primitives like lines, triangles, circles etc. and raster images, also referred to as bitmaps. The model is then used to feed the GPU via intermediate drawing APIs and device drivers which translate the model into GPU instructions. After that, the model is processed by the graphics hardware by various operations that turn the 2D or 3D model into a single raster image. (Owens et al. 2008; Möller et al. 2008: 11–27.)

PC video applications mostly run in graphical desktop environments which usually consist of 2D objects built from a mix of graphics primitives and bitmaps. The drawing infrastructure of each windowing systems is different, but traditionally 2D rendering has been implemented by graphics APIs that only support a sub-set of the capabilities of modern graphic hardware. These APIs handle drawing of 2D primitives like lines and rectangles but also bitmaps and fonts needed for a GUI (Foley, Van Dam, Feiner, Hughes & Phillips 1994). The graphics hardware is controlled by a command stream sent by the rendering library through kernel drivers. For many Unix and Linux based operating systems this work is done by the Xlib library in the X window system, by the Graphics Device Interface (GDI) in older Windows systems and by the QuickDraw library in older Macintoshes. (Wallossek 2010; Thompson 2006: 15–20; Paul 2000.)

However, with the rapid increase of the capabilities of the GPU and their affordable price, desktop operating systems have taken new hardware accelerated rendering models into

use. Most major desktop operating systems, including Windows 7, OSX and many Linux distributions, now have composited desktops which apply hardware acceleration to 2D rendering (Compiz community 2010; Thompson 2006: 27–33; Wallossek 2010). In composited windowing systems windows are rendered unclipped off-screen and then composited together with Porter-Duff composition, also known as alpha blending, for the effect of translucency (Ritger 2006) Most new rendering architectures utilize the graphics hardware better by utilizing 3D rendering APIs like OpenGL or Direct3D. This allows full control over the graphics hardware and thus use of the advanced hardware features not accessible with old 2D rendering APIs (Thompson 2006: 27–33).

4. ARCHITECTURE OF THE IMAGING STATION

In design of a performance-critical imaging application, especially on the PC, the selection of hardware and software platforms as well as successful application design are crucial. Understanding hardware technology is important for choosing a PC assembly that best fulfills the performance requirements of the application. The other key area is to ensure good utilization of the hardware by doing research on available software and following good software engineering practices. This chapter presents the hardware and software solutions used for building the imaging station.

4.1. Hardware configuration

The internal cameras of the operating microscope can be accessed directly from a composite output found on the back panel of the device. The HD camera stream is instead routed via a separate videohub (Image 1 HUB™ HD, Karl Storz GmbH & Co, Tuttingen, Germany), which transforms and down-scales the Full HD stream to SDTV composite format. The HD camera can then be accessed from the composite output of the videohub. The complete hardware setup of the imaging system is shown in Figure 8, where the red line represents the connection between the PC and the composite out of the operating microscope and the blue, the path from the HD camera to the PC via the video hub.

A dedicated PC will be used for hosting the imaging application. The acquisition of the PC is the responsibility of Töölö Hospital. To help this task, a reference PC assembly as a list of recommended hardware specifications, shown in Table 4, was delivered to the Töölö Hospital officials. The software installations and customization will be done later and are not a part of this project.

Factors in the selection of the reference PC components, ordered by descending priority, were: sufficient, but not excessive, performance, small form factor, updatability and prize. The reference PC has a powerful CPU with an integrated basic GPU on-chip. A dual channel memory controller and a PCI Express (PCIe) interface are also integrated in the CPU die. The system uses the latest DDR3 main memory and has a relatively fast traditional hard disk. The system supports updating with a dedicated GPU and a faster processor and main memory, if there is need for it.

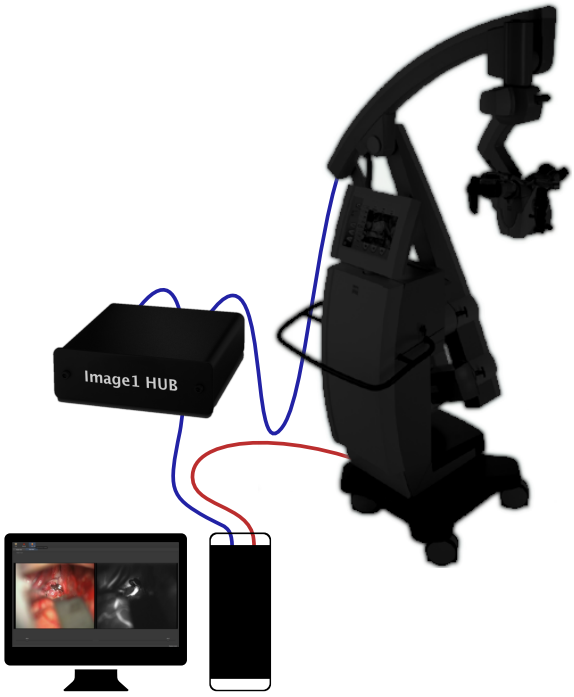

Figure 8. The connections between the PC imaging station and operating microscope imaging system. The red line represents the connection of the microscope-integrated cameras, while the connection of the HD camera is colored blue. The video hub is responsible for transcoding the HD camera stream to composite format.

In addition to the standard PC components, two framegrabbers are needed for capturing the HD and operating microscope camera streams. The University of Vaasa purchased two composite framegrabbers (mvDELTA, MATRIX VISION GmbH, Oppenweiler, Germany) using the standard PCI interface for this purpose. The cards were used for testing the application during development and after this project they are borrowed to Töölö Hospital for on-site testing.

4.2. Software platform

The imaging station application named iStation was implemented on a GNU/Linux distribution called Ubuntu. The main motivation behind choosing this platform was the desire to make the system open and extensible. Ubuntu is entirely based on Free and Open Source Software (FOSS) by default (Canonical 2010). The main components of the platform are the Linux kernel, the GNU operating system tools including the X win-

Table 4. A reference hardware specification delivered to Töölö Hospital officials to help in the purchase of the imaging station PC.

Component	Details
Case	COMPUCASE 7KJC, μATX
Motherboard	ASRock™ S1156, H55 chipset, μATX, 1×PCIe, 2×PCI
Processors	Intel® Core™ i5-661 @ 3.33GHz, Intel® HD graphics
Memory	4 GB, dual channel, DDR3 @ 1333MHz
HDD	Western Digital® Caviar 640GB, 6Gb/s, 7200rpm, 64MB cache

dow system and the GNOME desktop environment. How these layers build a complete system and what kind of environment they provide for efficient video processing is described in this section.

4.2.1. GNU/Linux

The GNU[1] project started by Richard Stallman in 1984 is an attempt to implement a mostly Unix-compatible operating system of completely free and open source code. The basic components of GNU include the GNU Compiler Collection (GCC), the GNU Binary Utilities (binutils), the bash shell, the GNU C library (glibc), and GNU Core Utilities (coreutils). The majority of GNU installations eventually ended up using the Linux kernel, instead of the official GNU kernel, the Hurd. Hence the name GNU/Linux is preferred to signify the use of a non-GNU kernel. (Stallman 1999.)

Linux is a monolithic kernel meaning that all kernel tasks like scheduling, memory management and device drivers are executed in the same process. The kernel tasks run in a privileged kernel-mode, whereas all application processes are run in user-mode and access hardware by requesting services from the kernel with system calls. Starting from version 2.6 kernel all Linux processes, including the kernel itself, are preemptive, which means they can be interrupted and switched for another process after any instruction. In this respect, the support for responsive or even soft real-time applications is quite good. (Bovet, Cesati & Oram 2006: 4,11–12.)

Linux processes are associated with two scheduling related characteristics: policy and

[1]The name GNU is a recursive acronym from the words "GNU is Not Unix" (Stallman 1999).

priority. Linux provides three scheduling policies: a default time sharing policy SCHED_OTHER and two real-time policies, round robin SCHED_RR and first-in-first-out SCHED_FIFO. The default time sharing policy splits the CPU time in equal time slices and periodically adjusts process priorities according to what they are doing; for example, the priority of a process that has received a lot of CPU time is decreased. Both of the real-time policies implement static priorities instead. Real-time processes are always scheduled before SCHED_OTHER processes, when they become runnable. SCHED_OTHER processes are also preempted in case a higher priority real-time process becomes runnable. The main difference between SCHED_RR and SCHED_FIFO is that the latter does not use time-slicing. The priorities of real-time tasks can be explicitly set in the range 1–99, while the priorities of SCHED_OTHER tasks can be affected with the `nice()` system call in the range -20–19. Lower values represent lower priorities. The real-time policies offer a simple way to ensure more CPU time and better responsiveness for performance critical applications. (Love 2010: 41–68.)

Memory management is another key factor considering efficient video processing. In Linux this is handled by the Kernel Memory Allocator (KMA). The KMA is responsible for satisfying all memory requests throughout the system. Linux uses the virtual memory system supported by the x86 architecture. Virtual memory is a layer of abstraction between the application memory requests and the hardware Memory Management Unit (MMU). The virtual address space used by applications is translated into physical memory locations by the kernel and the MMU in co-operation. Modern CPUs provide hardware circuitry to automate this translation process. In this system, each address in the virtual address space corresponds to a contiguous block of physical addresses, called a page frame. The address space of a process is thus divided into page frames of 4 or 8 KB in size. The drawback of virtual memory is additional latency incurred by address translations and further loss of predictability, because any data can be swapped to disk, unless specifically denied. The main advantages of virtual memory include the ability to:

- run several concurrent processes,
- run processes which require more physical memory than is available (swapping to disk),
- run a process that is only partially loaded in main memory,

- share a single memory image of a library or program. (Bovet et al. 2000: 31-34.)

In the Linux file system, I/O devices are accessed via associated files in the `/dev` directory. Thus the first video device attached to an arbitrary port is accessed, by default, via `/dev/video0`. The kernel needs a device driver in order to communicate with the device. In Linux device drivers are dynamically loadable kernel modules. The device drivers for video input and output are accessed through a uniform kernel interface called Video for Linux (V4L), or V4L2, which is the current major version. By using V4L2 the application developer can communicate with a wide set of devices with the same API. V4L2 also provides a way to get device capabilities and set parameters from the application code. The API also supports setting up advanced device features like DMA and memory mapping. (Schimek, Dirks, Verkuil & Rubli 2008; Bovet et al. 2000: 34–35.)

4.2.2. X window system

The X window system (X11) is a network transparent graphical user interface solution. It provides a basic framework, or primitives, for building GUI environments including drawing and moving windows on the screen and interacting with a mouse and keyboard. X relies on a client-server model depicted in Figure 9 where the X server provides I/O services to local and remote clients (applications). The server receives X protocol, or OpenGL, command streams from client applications and controls the display. The other half of the servers function is to route user input to the client applications. Network transparency ensures that the client application does not need to run on the same machine with the server. The X server does not include window management functionalities. This work is instead done by separate client application processes called window managers. (Welsh 2006: 571–573.)

X was developed in the era of simple 2D graphics. To support the 3D rendering features of modern GPUs, a new accelerated rendering model was necessary. The natural graphics API to be supported in X implementations is OpenGL. While there are several OpenGL implementations for Linux, the most prominent one is Mesa.

Because OpenGL is platform independent, an extension that glues OpenGL to the windowing system of X is needed. This extension, called GLX, allows OpenGL command streams to be sent over the X socket to the server. The GLX rendering model is known

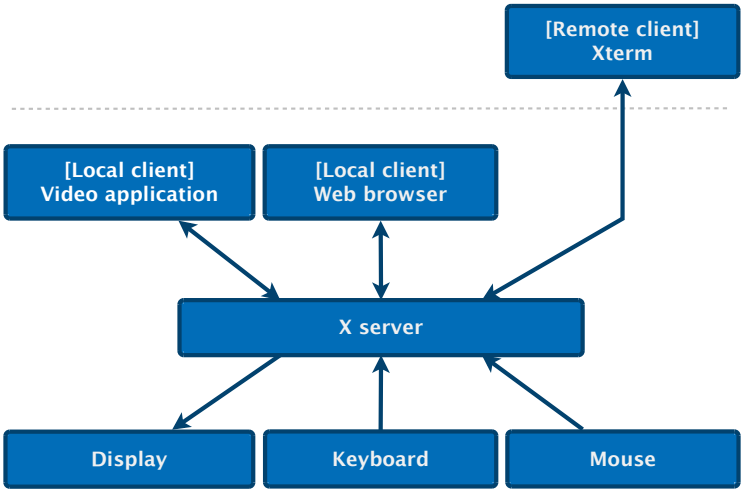

Figure 9. The server-client model used by the X window system. The applications (clients) request services, like drawing to a display, from the X server, which also relays the data from input devices, like mouse and keyboard, back to the correct application (Welsh 2006: 571–573).

as *indirect rendering*. Another, much faster, option is to bypass the X server and send OpenGL more directly to hardware. This rendering model, called *direct rendering*, is implemented by the Direct Rendering Interface (DRI) in the X.org version of X11, used in most Linux systems. When using DRI, the drawing is managed in the kernel by the Direct Rendering Manager (DRM). (Welsh 2006: 586–587.)

While the indirect rendering model is slower than direct, it is often sufficient for accelerating 2D rendering. The GLX model provides the advantages of better performance and access to advanced GPU-features via OpenGL, which the standard X drawing protocols and primitives do not. Indirect rendering also preserves network transparency of X, while direct rendering can only be used on the local machine. In all accelerated drawing a suitable driver needs to be loaded. Typically only the hardware vendor's proprietary drivers allow hardware acceleration – with the exception of Intel's integrated graphics drivers of which there are only open source versions. (Welsh 2006: 587–589.)

Because X does not provide window management, many different window managers have been implemented, some them utilizing the hardware acceleration methods provided by X more extensively than others. Compiz fusion is one of the fully accelerated compositing window managers for X. It is based on the Composite and GLX extensions

provided by X11 (Compiz community 2010). Compositing window managers handle windows as off-screen memory buffers. This allows windows to be composited with alpha transparency and other advanced effects in the graphics hardware (Ritger 2006).

X also does not provide an implementation of the window interiors. This work is done by the widget toolkit. A widget toolkit for X essentially contains a library of geometrical primitives and bitmaps which are used to build the user interface elements, often called widgets, like progress bars, check boxes, toolbars, panels and windows. Implementing a graphical application usually takes place in the way described in Figure 10. The client application does not talk to X server directly, it only uses a widget toolkit to create the user interface. The toolkit then uses Xlib to translate the higher level commands into X protocol ones. (Krause 2007: 1–13.)

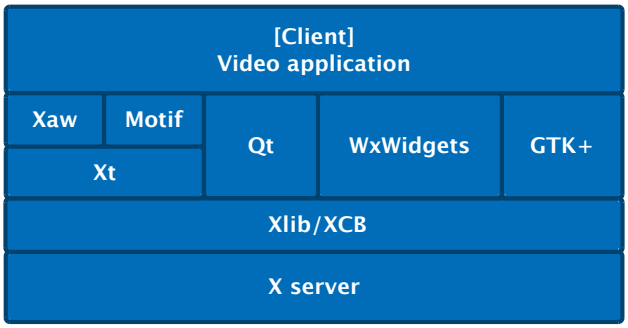

Figure 10. The hierarchy of software libraries used in the traditional, non-accelerated GUI programming for the X window system. The second layer from the top consists of widget toolkits, which use a common X protocol translator layer to communicate with the server (Krause 2007: 1–13).

4.2.3. GNOME desktop environment

The GNU Network Object Model Environment (GNOME) provides a graphical desktop environment built on top of the X window system, thus allowing it to be used with various Unix-like operating systems. GNOME consists of a large set of sub-projects which provide the look-and-feel, commonly needed productivity applications, utilities, and daemons as well as an application development environment. A few central sub-projects of GNOME include:

GIMP Toolkit (GTK+) - a cross-platform widget toolkit for creating GUIs,

Gconf - a framework for storing application settings,

GNOME Virtual File System (GVFS) - a virtual file system for transparent file access,

GNOME keyring - a way of storing encryption keys and security information. (Welsh 2006: 88–94.)

The most influential entity for the graphical desktop environment of GNOME is GTK+. It was first developed for the GNU Image Manipulation Program (GIMP) in 1997, but evolved into a generic toolkit for graphical programming over the years. From the second major version onwards, the portions of GTK+ that are unrelated to graphics are separated into the Glib library. Glib provides utilities like the object (GObject) and type system (GType), threading, timers, and various advanced data structures. Other important libraries related to GTK+ and graphics include the font-rendering engine Pango, the 2D vector graphics library Cairo and the main 2D graphics library GIMP Drawing Kit (GDK). (Krause 2007: 1–13.)

4.2.4. Application development libraries

The most critical and tedious task of this project was finding a suitable video processing framework. For the other components equally good alternatives are easier to find. The main software components used in this project include GTK+ for GUI programming and GStreamer for handling video streams. Several GNU utility libraries are also used in the application.

The application developed here only supports the specific hardware and software platform selected for the project, because it is not aimed for use by a broader public for the time being. All the components selected are cross-platform though, making sure that the application or parts of it can more easily be ported at a later time if necessary.

4.2.4.1. GTK+

GTK+ was chosen because it is the standard widget toolkit for the GNOME desktop environment, it is cross-platform, and provides an object-oriented programming model for high productivity – even though implemented with the standard procedural C (Krause 2007: 1–13). GTK+ also works well together with GStreamer since they both use the data type and object system provided by Glib.

4.2.4.2. GStreamer

GStreamer is a cross-platform media streaming framework for building arbitrary media handling applications. GStreamer was chosen for its flexible and powerful media processing model, its design for performance, and its support for a wide range of different audio and video formats. With GStreamer implementing different kinds of pipelines is simple thanks to the clear, modular pipeline concept. In GStreamer the pipelines consists of different types of elements, which can be categorized for example in source, sink, filter and flow control elements. Each element has its unique capabilities, that is, the formats it can handle and properties, which are the values of its internal variables. (Taymans et al. 2011.)

On the OS level, GStreamer consists of a small set of core libraries and everything beyond that is provided by plugins. Plugins are actually dynamically loadable shared object files coding the functionality of one or many elements. This makes extending GStreamer particularly easy, as one only needs to write a new plugin with the desired functionality, compile it, and make it available in the GStreamer search path. All plugins need to implement a specific interface, which is provided as a template code by the community. (Taymans et al. 2011.)

According the documentation, high performance is one of the fundamental design goals in GStreamer. The techniques used by GStreamer to enhance performance include:

- dedicated streaming threads, leaving scheduling to the kernel,
- extremely light-weight link between plugins resulting in minimal pipeline overhead,
- providing mechanisms to work directly with target memory minimizing copying of memory,
- loading plugins only when they are needed. (Taymans et al. 2011.)

The selection of GStreamer for this project is also supported by the fact that is provides an opportunity to switch to an OpenGL-based processing model in case the performance of some desired operation is not acceptable with the CPU-powered plugins, used by default (Taymans et al. 2011). The bridge to OpenGL on GStreamer is the `gstopengl` plugin, which provides the basic elements for content transfer between the GPU and CPU worlds

and some readily usable processing elements. These elements can be used for implementing, for example, video scaling and colorspace transformations on the GPU. The use of these plugins requires installation of the OpenGL software packages, plus a suitable GPU hardware and display driver that supports the OpenGL features used by the elements.

5. RESULTS

This chapter presents the contributions of this project. The main outcome of this work is the iStation application, which provides several new features to aid in online diagnostics. Before explaining the details of how iStation was implemented, the principles of selecting its features is briefly discussed. The other half of the results are the findings from a set of feasibility studies, carried out to select suitable video post-processing operations to be included in iStation. Although none of the post-processing features were eventually incorporated into the final application, these studies provided useful information about the clinical usefulness of the operations and their performance on the PC platform, and are therefore presented in a separate section.

5.1. Selection of features

The functionality chosen for implementation into iStation is mainly based on the feature requests by neurosurgeons M.D. Ph.D. Martin Lehecka and M.D. Ph.D. Aki Laakso. The most suitable operations for implementation considering factors like implementation time, hardware restrictions and technical difficulty were discussed and chosen during meetings and on the phone. The transcripts of these conversations are provided in Appendices 1–5. Due to the limited scale of this work a lot of the ideas that came up in the discussions were left out in favor of concentrating on the few most important features.

The most important feature standing out in the conversations was the ability to replay ICG-VA studies during surgery. This is important because the ICG fluorescence lifetime is relatively short and vital details could easily be missed during live viewing. Thus a dedicated ICG video recorder and player was decided to be implemented.

The second most important feature was considered to be the ability to compare the anatomic color video and the grayscale ICG-VA video side-by-side or combine their information in some way. Both of these methods would make it easier for the surgeons to map the veins in the ICG angiogram to the anatomic image. Both of these methods were also tried, but only the side-by-side version was found technically feasible at the end.

Additionally, pseudo coloring of the grayscale ICG videos was tested using a few different color palettes. This was supposed to enhance the contrast of the veins, based on

the fact that the human eye discerns more colors than gray levels. However, the clinical usefulness of this operation was not clear and thus it was left out at this point.

5.2. Implementation of the video processing application

The source code of iStation was entirely written with the C programing language and compiled with GCC 4.4.5. Although all of the software components are available for the three major PC platforms, Windows, GNU/Linux and Mac OSX, the source code will only compile in a Linux system in its present form. At the release time, iStation could handle two input streams; one intended for the stream from the HD camera and one for the stream from the internal cameras of the operating microscope.

5.2.1. User interface and features

The user interface was build with a graphical designer called `Glade`. The two most important design principles were simplicity and clarity. This is important for easy adaptation and avoiding confusion that could lead to unnecessary time loss during surgery. The coloring of the interface was also given special attention to provide pleasant viewing conditions and highlight important information. Because background of the ICG illuminated veins is dark, the user interface was also made dark. If the user interface was bright the eye would adjust to it, making it harder to distinguish details in the mostly dark ICG video.

The operation logic of iStation is built around three states: streaming, recording, and playback. The transitions between different states are triggered by pressing dedicated buttons on the toolbar. Below the toolbar is a tabbed view which is used to display the video outputs. The release version of iStation contains two tabs named `Single view` and `Dual view`, where one or two input video streams can be viewed respectively. This structure was selected to allow views to be easily removed and added without a major redesign of the operation logic. Standard playback controls were deliberately omitted from the interface for simplicity. Instead, sliders underneath every video output show the stream duration and allow rewinding. Streams are toggled between paused or playing states by left mouse clicks on video outputs. A screen shot of the GUI is shown in Figure 11.

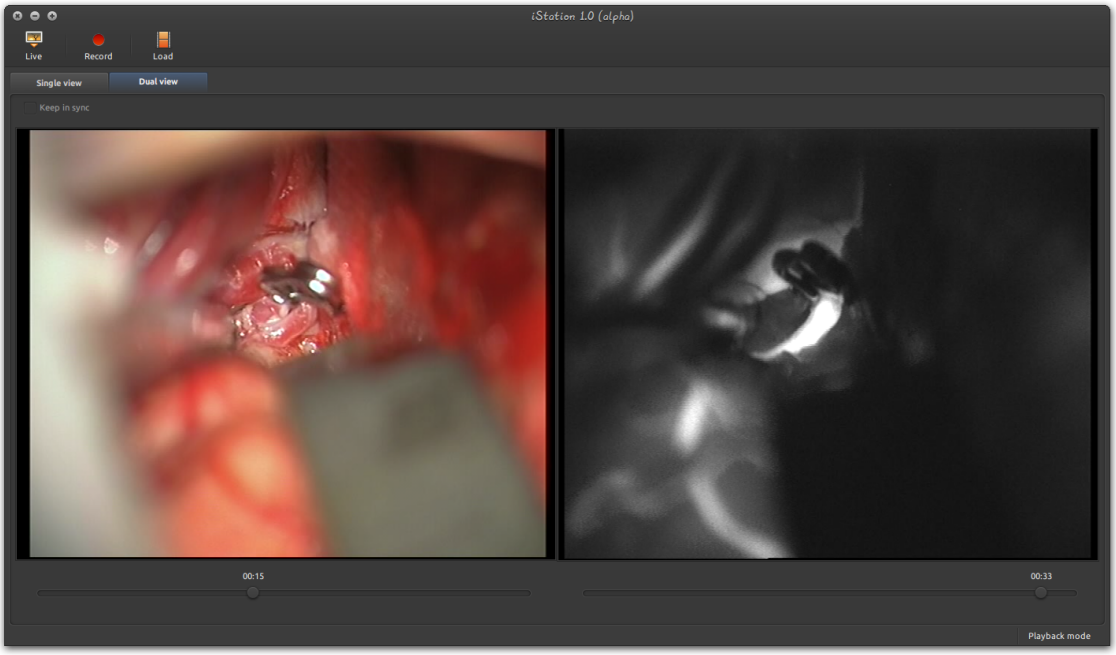

Figure 11. The user interface of iStation showing the playback mode in side-by-side view. The GUI consists of a toolbar (top), a tabbed viewing area (middle), and a status bar (bottom). Application state changes are triggered by pressing the buttons in the toolbar. The viewing mode is changed by clicking the tabs.

iStation dual view is intended for making comparisons between two video streams and the single view for giving a better level of detail on one stream. The minimum size of the video windows is constrained, but scales upwards as much as the screen space allows. Either one of the view modes can be activated at any time and they are available for both live and recorded streams.

At the moment, up to two video inputs are automatically detected and chosen during startup and on certain state changes. The configuration of the properties of these video inputs is made available by an external application, called `v4l2ucp` (V4L2 Universal Control Panel), launched from the menu (*Edit* > *Preferences*). The settings made in `v4l2ucp` are stored in device specific configuration files (`video<n>.cfg`), from where they are also loaded upon iStation startup.

The default state of iStation is streaming. When the streaming mode is activated, a subroutine is run to detect all suitable video devices, of which, two first discovered are used by the application. After the device check, the available live streams are displayed in the

video windows. The recording state also starts with a device check before recording to detect possible changes since last update.

While recording, the stream or streams are also shown in the active video outputs. The resulting files are named using a prefix `icg` or `hd` and postfix formed by a unique time stamp for easy recognition and maintainability. Recording is ended by pressing the `Record` button again. After that the recorded study is played back the same way as in playback state. The other way to initiate the playback state is to press the `Load` button on the toolbar, which will pop up a file chooser dialog. The dialog prompts to select one or two files, which will then be played back. If only one is chosen, it will be played back in the dual view mode to allow comparison of the video at different positions. If more than two files are chosen, the first two selected will be played back.

When the playback state is entered and there are two files to play, the playback synchronization checkbox is activated in the dual view mode. This checkbox allows the user to select whether the two files are kept in synchronization in respect to each other, even if either of them is paused or rewinded. If playback or rewinding results in an end of file from either file, that video will be reset to the beginning and the checkbox is unchecked. If the stream positions are different when the checkbox is checked, the right hand side video is rewinded to the same position as the other video. This feature was included to make it easier to compare corresponding time instances in streams that were simultaneously recorded.

5.2.2. Video processing pipelines

GStreamer was used to implement all six video processing pipelines used in iStation. A GStreamer pipeline is managed simply by chaining a set of elements, setting up their properties and controlling the state of the resulting pipeline object. Setting the stream properties, like specifically requesting for some video format or framerate, is implemented by adding special elements called capability filters to suitable places. If possible, the capability filters override the default stream properties. Before the pipeline is started, a process called capability negotiating is performed. In the negotiation phase, the formats between different elements are chosen so that a complete pipeline can be built. (Taymans et al. 2011.) A summary of the pipelines used in the design includes:

- ❖ two pipelines for displaying notification video loops for unavailable inputs,
- ❖ two pipelines for recording and simultaneously displaying the available live input streams,
- ❖ and two pipelines for displaying the available live input streams and playing back video files.

Diagrams clarifying the playback, recording, and notification pipelines are shown in Figures 12, 14, and 13 respectively.

The playback pipeline uses the `playbin2` element (dashed element in Figure 12), which is a standalone media player element capable of playing back a variety of sources presented to it as an URI. The `playbin2` actually consists of a set of dynamically assembled elements linked upon creation, which is signified by the element with a question mark in Figure 12. The sink element of `playbin2` must still be implicitly set. `playbin2` was chosen, because it automatically detects the video format and settings for the given source. Thus, iStation readily supports a wide variety of file formats if the format of the recorded studies, for instance, should change in the future.

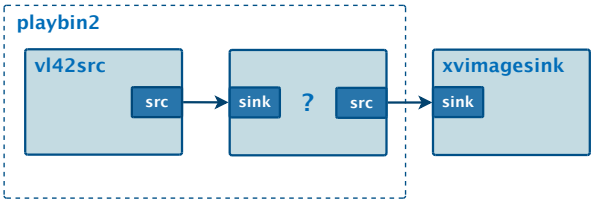

Figure 12. A data flow diagram of the two pipelines used for playing back the live streams and video files in iStation.

The live input streams in iStation are always accessed with the `v4l2src` element. This element essentially integrates the video device handling capabilities of the V4L2 framework to the rest of GStreamer. The element also provides the ability to get a list of suitable devices present in the system through the `GstPropertyProbe` interface. The probe interface is used in iStation to update the device list every time the state changes to streaming or recording.

The output video streams are drawn on the display areas by the `xvimagesink`, which uses the X video extension (Xv). Xv is a video output method for the X window system that

supports off-loading scaling and color space transformations to the graphics hardware (Wikipedia contributors 2011c). iStation also uses the `GstXOverlay` interface, which in this case enables correct timing when assigning the output X window ID for the related `xvimagesink`. When changing from the single view to dual or vice versa, the `xvimagesink` of the affected pipeline is simply remapped to a different output window ID.

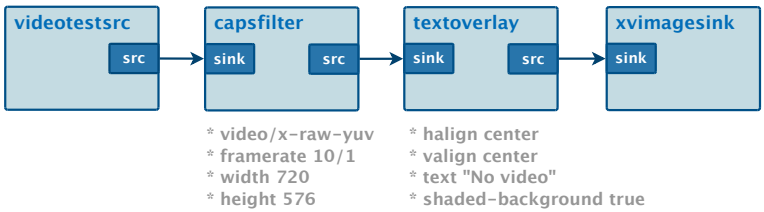

Figure 13. A data flow diagram of the notification pipelines used to relay messages to the user as video overlays in iStation.

The recording pipeline forks a single `v4l2src` into two sinks, a `filesink` and an `xvimagesink`. The branches are synchronized and buffered by using two `queue` elements at the beginning of each branch. Buffering a live source is important, because it prevents momentary gaps in the source stream from causing lost frames in the recorded branch. The file formats of the output files are statically set to use the Audio Video Interleave (AVI) container with the stream encoded as M-JPEG. The M-JPEG codec was chosen to minimize the encoding and decoding latency for smooth rewinding, and still maintaining a reasonable file size for the recordings.

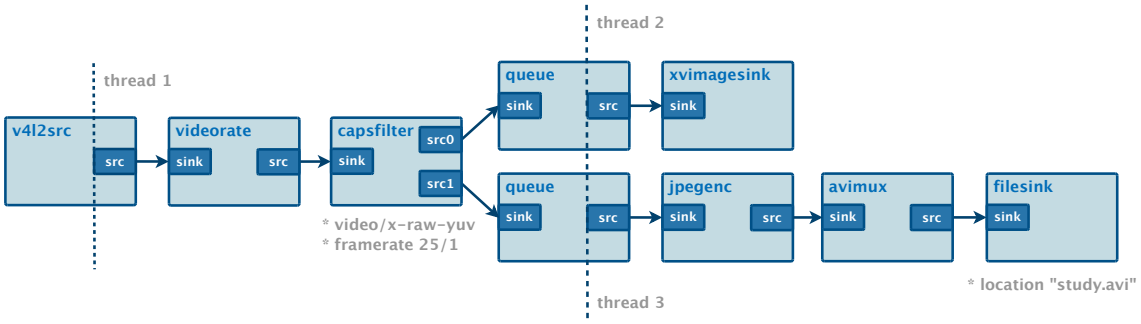

Figure 14. A data flow diagram of the recording pipelines used for simultaneously storing studies and displaying the live stream in the GUI of iStation.

5.2.3. Performance

iStation showed good performance from the start, which was considered a result of the relatively light processing operations and a successful selection of the software stack. The results reported here were acquired with a reasonably powerful previous generation Intel assembly (see PC1 in Table 5 for details). Some basic tests were conducted to ensure the quality of the self-written application code. Runs with the Valgrind profiler did not show any memory leaks or other problems, which could cause performance degradation and crashes over longer periods of continued use.

When testing with two 720×576 streams, the percentage of CPU time taken by iStation was around 12% during live steaming, 20% during video playback, and 50% when simultaneously streaming and recording. In fast back and forth rewinding, both CPU cores were loaded close to 100%, but there was no perceptible jerkiness in video playback. No dropped frames were reported in the recording tests. These results gave no incentive for further optimization of the software or switching to a GPU-based processing model.

The second most important performance indicator, when no video processing related ones were discovered, was the user experience. The state transitions of iStation were considered relatively fast with the test machine giving good user experience. The playback was also found smooth even with two 720×576 videos playing concurrently. The overall responsiveness of the application was found sufficient, although there was a great deal of unnecessary software running on the system concurrently and the *nice* value of iStation was left untouched.

5.3. Image post-processing experiments

During the course of the project a few image post-processing operations were studied with the aim of using them in iStation. OpenCV was used for most of the image processing experiments, because of its extensive set of image processing routines. None of these experiments led to inclusion in iStation though. This was due to either imaging system constraints or questionable clinical usefulness. This section introduces the post-processing operations tried during the study.

5.3.1. Blending angiograms with anatomic images

The clinically most prominent post-processing operation tried during the study was combining the ICG and anatomical videos. Inclusion was not possible due to two properties of the operating microscope. Firstly the two internal cameras cannot be accessed simultaneously. Secondly the ICG and anatomical cameras are not aligned to the same optical axis, resulting in misaligned video streams, which are not well suited for blending purposes.

The first issue, which prohibits online blending with the current imaging setup, was known from the beginning. A small feasibility study was still decided to be done, because the idea of blending was endorsed by the surgeons and the results could later be used with suitable next generation hardware. The second point was discovered only after testing the idea of blending with the actual concurrently recorded video material copied from the internal HDD of the operating microscope.

A simple method of blending, called Porter-Duff blending or alpha blending[1], was used in the tests. The operations itself is a simple pixelwise linear combination of two colors (Porter & Duff 1984). For example, the resultant color of two overlapping, partially transparent, pixels in an RGBA color space can be calculated according to Equation 1:

$$C_{res} = \alpha C_1 + \beta C_2 \qquad | \, \alpha, \beta \in [0,1] \tag{1}$$

In the equation, α and β represent the transparency of C_1 and C_2, the original colors of the two pixels, and C_{res} the resulting color. Alpha blending was a natural choice for prototyping, because of its simplicity and the fact that there are many implementations, including hardware accelerated ones, already available. This is because alpha blending is routinely used in PCs.

5.3.1.1. Performance tests

Alpha blending was tested with two different processing frameworks. One used a GStreamer pipeline, shown in Figure 15, with the `videomixer` element and the other a small GCC 4.4.5 compiled C program using OpenCV routines. Information about the

[1] Alpha blending was originally introduced in 1984 in an article by Thomas Porter and Tom Duff (Porter & Duff 1984).

test environment is gathered in Table 5. In the GStreamer test, the `gst-launch` utility was used to find out, how significantly decoding of a relatively complex format, MPEG–2, affects the performance, compared to alpha blending of two raw streams. With the OpenCV program, the aim was to study the clinical usefulness of the alpha blending method and to analyze the performance of the alpha blending together with a simple pseudo coloring scheme in CPU-based processing.

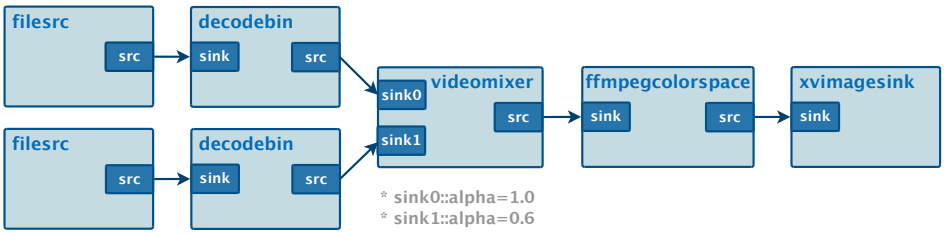

Figure 15. The GStreamer pipeline used for alpha blending two video streams in the feasibility studies. The blending takes place in the `videomixer` element.

The colorspace in all tests was a chroma subsampled planar YUV 4:2:0 (FOURCC=I420), meaning that a transformation to an RGBA space was needed at some point before rendering. In case of GStreamer, the blending took place in I420 format in the `videomixer` element, while the conversion to RGB(A) was done in `ffmpegcolorspace` element using the highly optimized routines provided by the FFMPEG project. The GPU accelerated `xvimagesink` was also used for rendering. In the OpenCV implementation the video frames were first transformed from the I420 to BGR24 format, then the red and blue channels in the ICG frame were zeroed and the frames blended with the function `cvAddWeighted()`. Although the routines in OpenCV are claimed to be fairly optimized, one should still get a better performance by using the Intel IPP backend. On the test machine, the OpenCV program informed that it is using a software scaler for the colorspace conversion, because an accelerated version was not found. Thus the results of OpenCV tests hardly represent the best CPU processing performance achievable.

In the tests with the OpenCV program, an approximation of the wall clock time was measured for the processing functions (P), the read-process-display sequence (RPD), and the whole processing loop including a wait state (RPDW). The results of these measurements can be found in Table 6. The time was measured with the function `gettimeofday()` be-

Table 5. The setup of the PC used for the feasibility tests on alpha blending.

ID	CPU	Memory	Graphics	Software
PC1	Intel® Core™2 Duo, E8500 @ 3.16GHz	4 GB, dual channel, DDR2 @ 1066MHz	Intel® GMA X4500	Mesa DRI Intel® 20100330; Linux 2.6.35-22 x86-64; GCC 4.4.5

longing to the Portable Operating System Interface for Unix (POSIX) specification. This function provides supports a microsecond granularity, but the actual granularity also depends on the timer hardware of the machine. The results here are all rounded to milliseconds. Because OpenCV does not support interrupt based event handling, a static wait state (W in Table 6) had to be used in the main loop to process user input. This feature was used to limit the framerate in the test application. The CPU load is also given for each test done with OpenCV. In the tests with GStreamer the relative CPU load was measured first using the uncompressed videos and then with the MPEG–2 encoded ones. The results of these tests are given in Table 7.

Table 6. The results of the alpha blending performance tests on the OpenCV application. The execution time for various parts of the processing loop (W=wait state, P=processing operations, R=read from file, D=display) was measured. The framerate was calculated based on the elapset time for the whole loop. The CPU load is an average of several runs.

Setup	API	ms/W	ms/P	ms/RPD	ms/RPDW	fps	%/CPU time
PC1	OpenCV 2.1.0-2	2	4	8	13	77	68
PC1	OpenCV 2.1.0-2	10	4	8	17	59	50
PC1	OpenCV 2.1.0-2	35	4	8	42	24	20

The results indicate that even a previous generation Intel PC running Ubuntu 10.10 and using virtually no hardware acceleration in the processing, is more than sufficient for fluent playback and light post-processing in 720×576 resolutions. When using the OpenCV library without any hardware accelerated backends, the processing time per frame including the color space conversion and rendering routines was found to be close to 8 ms. This corresponds to a framerate of 125 fps. The pseudo coloring and alpha blending op-

Table 7. The results of the alpha blending performance tests on GStreamer, using the `gst-launch` utility. In this test, the average percentage of CPU time taken by the implemented alpha blending pipeline (see Figure 15) process was measured.

Setup	API	Container/Codec	Resolution	fps	%/CPU time
PC1	GStreamer 0.10.30	AVI/Uncompressed planar YUV 4:2:0	720×576	25	7
PC1	GStreamer 0.10.30	MPEG/MPEG–2 planar YUV 4:2:0	720×576	25	15

erations took on average only 4 ms per frame on the test machine, giving a comfortable sustained rate of 250 fps.

The tests with GStreamer showed much lower CPU utilization than the OpenCV ones with similar framerate. This may be for a number of reasons. For one, the GStreamer version does not include the pseudo coloring phase. GStreamer might also have better hardware acceleration features, especially in color space transformations, and its pipeline implementation might be more efficient, because it is designed for media streaming unlike OpenCV. Either way, the study with GStreamer showed that it can also quite easily be used for alpha blending and that adding the complexity of decoding an MPEG–2 stream is not really an issue for a relatively modern CPU with these levels of video resolution.

5.3.1.2. Clinical usefulness

The clinical feasibility could be better tested with the OpenCV program, because the alpha values of the frames were tied together and controlled by a slider in the GUI shown in Figure 16. The white light and NIR frames were given a complementary alpha value in the application. Thus, an alpha value of 0.2 for the ICG frame, means 0.8 for the other. Zeroing the red and blue channels was done to give a distinct color for the ICG. The value of the slider control in the user interface is the alpha as a percentage used for the NIR video. The text overlay showing the framerate of the resulting video was used in the performance measurements.

The usefulness of the method was confirmed by the test application, although it was acknowledged that it still needs some refinements. Unfortunately this potentially useful method had to be left out of iStation because of the two properties of the operating microscope mentioned earlier. The issue with the stereoscopic camera setup of the operating

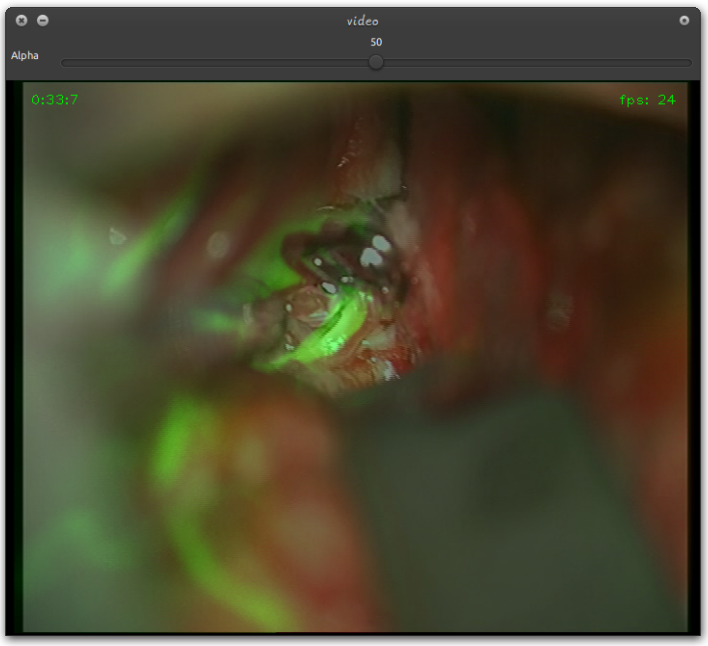

Figure 16. A screen shot of the OpenCV demo application used in the feasibility studies on alpha blending. The application blends two videos having a complementary alpha. The value of alpha can be adjusted with the provided slider control.

microscope produces the lateral shift between the ICG and white light frames, visible in the still image of Figure 16 and even clearer in live video.

5.3.2. Pseudo coloring angiograms

A pseudo coloring scheme was tested in the early phases of this project. This method had theoretical potential to enhance the contrast and visibility of subtle changes in intensity in ICG angiograms. Several different palettes were tried with a medical image analysis software Amide. Some of these are shown in Figures 17 and 18. Although the experiment showed some initial promise, it was not well received by the surgeons as such. The feature was eventually dismissed from iStation to focus on more useful features.

A decent performance study was never conducted on a pseudo coloring implementation. A prototype Java program implementing a "rainbow" palette for a live SD quality stream was implemented, but not properly benchmarked. The image processing in the application was done with the routines in the ImageJ image processing library. In fact, benchmarking was not even necessary, as the application was clearly short of meeting

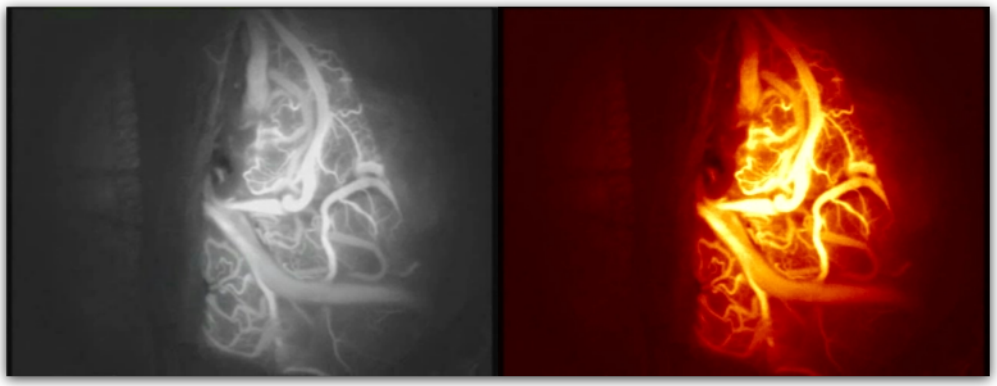

Figure 17. A pseudo coloring example using the "hot metal" palette from Amide. The original ICG angiogram (left) and the pseudo colored image (right).

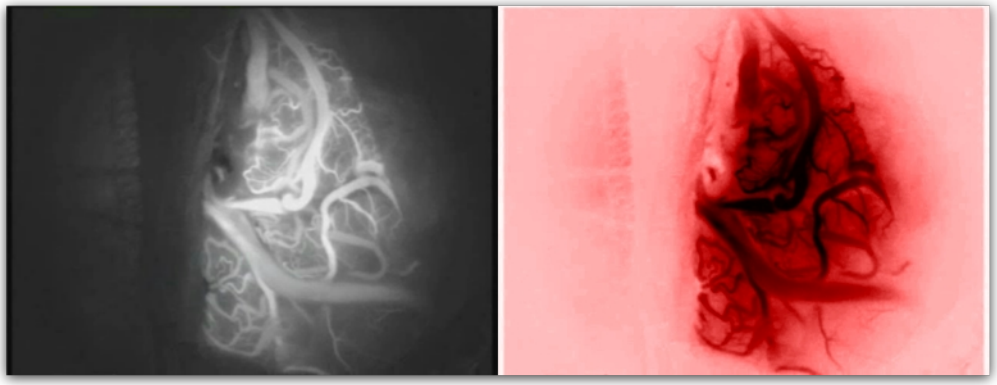

Figure 18. A pseudo coloring example using the "inverse red" palette from Amide. The original ICG angiogram (left) and the pseudo colored image (right).

fluent playback requirements, with the sustained framerate being approximately 10–15 frames per second on a slightly outdated Pentium® D test machine. As this was a study on the clinical usefulness of the rainbow pseudo coloring method, also reflected by the choice of programming language, there was very little effort to optimize the performance of the application. Hence, these trials are not considered representative for assessing the computational feasibility of the pseudo coloring method.

6. CONCLUSIONS

This study was about finding and implementing new clinically useful features for an operating microscope integrated imaging system at Töölö Hospital. For successful completion of these tasks, both the medical expertise of the operating neurosurgeons and the technical know-how acquired by thorough background work was invaluable. The implemented features were incorporated into a single PC video processing application, which uses the live streams of the operating microscope. The PC platform was chosen to host the application because of its unparalleled prototyping features and sufficiently high video processing power for the expected processing tasks.

During the project, the priorities of the objectives cleared up and providing implementations for the most basic usability related features took preference. These included the possibility to pause, rewind and compare the videos produced by the two modes of the operating microscope, the ICG-VA and the white light mode. Finding beneficial video post-processing operations and testing their feasibility, to see if they could be included in the application, formed the secondary goal.

The main achievement of this project was the custom medical imaging application iStation. The key features of this software include the ability to simultaneously view and record upto two live video streams and play back the recordings in either a single, bigger screen or side-by-side comparison mode. The playback modes provide full media player controls, enhanced with a frame-by-frame precision rewind, in an intuitive and responsive interface. Even though iStation is specifically designed to aid in video based medical diagnostics at Töölö Hospital, it could be used anywhere, where a transient phenomenon needs to be carefully evaluated or two simultaneously recorded sources compared. The feature set of iStation can also easily be modified because of the GUI design and a clean separation of the video streaming code from that of the rest of the application.

The other half of the project's results was the information provided by the feasibility studies on image post-processing. This knowledge could later be used to enhance this or other imaging systems with similar aspirations. Experiments with alpha blending revealed that the operation is not possible with the current version of the operating microscope, but the clinical usefulness and computational feasibility on the PC platform were encouraging. The performance of the OpenCV based alpha blending demo application

easily satisfied the requirements of fluent playback with two SD quality videos, although run completely on the main CPU with standard optimizations. The tests with pseudo coloring schemes for the grayscale ICG angiograms were not equally encouraging, as the results were not considered to improve diagnostic accuracy expectedly. The feasibility studies also revealed that even a previous generation PC hardware has quite good prerequisites for implementing useful online video processing operations, even without resorting to hardware acceleration.

A great portion of the project's duration was spent on familiarizing with the hardware and software architectures for efficient video processing and finding the right software frameworks to implement the imaging station application on the PC platform. Eventually, many of the high-throughput methods, like utilizing the GPU, were not used in iStation, as its performance was already found satisfactory. The good performance of the application was not completely unexpected considering the processing power of modern PCs, the relatively light processing operations, and moderate-quality 720×576 videos used. Even though the chosen video processing architecture allows utilizing hardware acceleration, which may be required for higher resolution videos or more demanding operations, this potential was not tested during this project.

At the end, most of the goals of this project were achieved. The biggest disappointment being the exclusion of the alpha blending feature from iStation. Some areas could still benefit from further study. It would be interesting to test the performance difference of using a GPU-based processing model as opposed to a CPU-based one and see how easily the GPU code can be integrated into the processing pipeline in practice. As the priority in this project was the implementation of the basic playback and recording functionality, more time could be used for finding feasible post-processing operations and incorporating them into the application in the future. No direct follow-ups for this project are currently planned though.

BIBLIOGRAPHY

Alander, J., T. Pätilä, T. Spillmann, A. Laakso, I. Kaartinen, M. Huotari & V. Tuchin (2011). A review of indocyanine green contrast agent in surgery [online]. Unpublished medical review [cited 17.1.2011]. Available at: <URL:http://lipas.uwasa.fi/~TAU/luukku/ICGAreview.pdf>.

Allusse, Y., P. Horain, A. Agarwal & C. Saipriyadarshan (2008). GpuCV: An opensource GPU-accelerated framework for image processing and computer vision. In: *Proceeding of the 16th ACM International Conference on Multimedia*, 1089–1092. New York: ACM.

AMD (2002). AMD HyperTransport Technology-based system architecture [online]. A white paper [cited 21.3.2010]. Available at: <URL:http://www.amd.com/us-en/assets/content_type/white_papers_and_tech_docs/AMD_HyperTransport_Technology-Based_System_Architecture.pdf>.

Avissar, O., R. Barua & D. Stewart (2001). Heterogeneous memory management for embedded systems. In: *Proceedings of the 2001 International Conference on Compilers, Architecture, and Synthesis for Embedded Systems*, 43. New York: ACM.

Babenko, P. & M. Shah (2008). MinGPU: A minimum GPU library for computer vision. *Journal of Real-Time Image Processing* 3:4, 255–268.

Björnsson, Ó., R. Murphy & V. Chadwick (1982). Physicochemical studies of indocyanine green (ICG): absorbance/concentration relationship, pH tolerance and assay precision in various solvents. *Cellular and Molecular Life Sciences* 38:12, 1441–1442.

Bovet, D., M. Cesati & A. Oram (2000). *Understanding the Linux Kernel*. Sebastopol: O'Reilly & Associates.

Bovet, D., M. Cesati & A. Oram (2006). *Understanding the Linux Kernel*. 3. ed. Sebastopol: O'Reilly & Associates.

Bradski, G. & A. Kaehler (2008). *Learning OpenCV: Computer Vision with the OpenCV Library*. Sebastopol: O'Reilly Media.

Brookwood, N. (2010). AMD Fusion family of APUs: enabling a superior, immersive PC experience [online]. Product brochure [cited 2.12.2010]. Available at: <URL:http://sites.amd.com/us/Documents/48423B_fusion_whitepaper_WEB.pdf>.

Bui, P. & J. Brockman (2009). Performance analysis of accelerated image registration using GPGPU. In: *Proceedings of 2nd Workshop on General Purpose Processing on Graphics Processing Units*, 38–45. New York: ACM.

Burger, D., J. Goodman & A. Kägi (1996). Memory bandwidth limitations of future microprocessors. In: *Proceedings of the 23rd Annual International Symposium on Computer Architecture*, 78–89. New York: ACM.

Canonical (2010). The Ubuntu Project [online]. Official project website [cited 1.6.2010]. Available at: <URL:http://www.ubuntu.com/project>.

Carl Zeiss Meditec (2008). See more. INFRARED 800. BLUE 400 [online]. Product brochure [cited 1.6.2010]. Available at: <URL:http://www.zeiss.de/C1256E3C00562B7C/0/5059BF784D4EEDCBC12573DE005902FC/$file/en_30_010_070_iii.pdf>.

Carl Zeiss Meditec (2009). UNIQUE OPMI Pentero [online]. Product brochure [cited 1.6.2010]. Available at: <URL:http://www.zeiss.de/C1256E3C00562B7C/0/04C893615DD200AAC1256E97002D4119/$file/en_30_010_077v.pdf>.

Carl Zeiss Meditec (2010). FLOW 800. A new dimension in intraoperative fluorescence [online]. Product brochure [cited 12.7.2010]. Available at: <URL:http://www.meditec.zeiss.com/C1256CAB00599F5D/Contents-Frame/6D984F8897BC53A88825726C00010C31>.

Chang, P., S. Mahlke & W. Hwu (1991). Using profile information to assist classic code optimizations. *Software: Practice and Experience* 21:12, 1301–1321.

Cherrick, G., S. Stein, C. Leevy & C. Davidson (1960). Indocyanine green: observations on its physical properties, plasma decay, and hepatic extraction. *Journal of Clinical Investigation* 39:4, 592.

Colic, A., H. Kalva & B. Furht (2010). Exploring nvidia-cuda for video coding. In: *Proceedings of the first annual ACM SIGMM inproceedings on Multimedia systems*, 13–22. New York: ACM.

Compiz community (2010). What is Compiz? [online]. Official project website [cited 24.9.2010]. Available at: <URL:http://www.compiz.org/>.

Dashti, R., A. Laakso, M. Niemelä, M. Porras, Ö. Celik, O. Navratil, R. Romani & J. Hernesniemi (2010). Application of microscope integrated indocyanine green videoangiography during microneurosurgical treatment of intracranial aneurysms: A review. In: *Surgical Management of Cerebrovascular Disease*, 107–109. Berlin: Springer.

Dashti, R., A. Laakso, M. Niemelä, M. Porras & J. Hernesniemi (2009). Microscope-integrated near-infrared indocyanine green videoangiography during surgery of intracranial aneurysms: The Helsinki experience. *Surgical Neurology* 71:5, 543–550.

de Oliveira, J., J. Beck, V. Seifert, M. Teixeira & A. Raabe (2008). Assessment of flow in perforating arteries during intracranial aneurysm surgery using intraoperative near-infrared indocyanine green videoangiography. *Neurosurgery* 62:6, 63–73.

Desmettre, T., J. Devoisselle & S. Mordon (2000). Fluorescence properties and metabolic features of indocyanine green (ICG) as related to angiography. *Survey of Ophthalmology* 45:1, 15–27.

Di Stefano, L., M. Marchionni & S. Mattoccia (2004). A PC-based real-time stereo vision system. *Machine Graphics & Vision* 13:3, 197–220.

Do Dinh, M. (2008). GPUs-Graphics Processing Units [online]. Vertiefungsseminar Architektur von Prozessoren [cited 3.11.2010].

Fallon, H., A. de Lattre, J. Bilien, A. Daoud, M. Gautier & C. Stenac (2002–2004). VLC user guide [online]. Online software documentation [cited 23.1.2011]. Available at: <URL:http://www.videolan.org/doc/vlc-user-guide/en/vlc-user-guide-en.html>.

Fischer, G., A. Stadie & J. Oertel (2010). Near-infrared indocyanine green videoangiography versus microvascular Doppler sonography in aneurysm surgery. *Acta Neurochirurgica* 152:9, 1–7.

Foley, J., A. Van Dam, S. Feiner, J. Hughes & R. Phillips (1994). *Introduction to Computer Graphics*. Boston: Addison-Wesley.

Frangioni, J. (2003). An operational near-infrared fluorescence imaging system prototype for large animal surgery. *Technology in Cancer Research & Treatment* 2:6.

Ge, G. (2008). Design and Implementation of 2-Channel Digital Image Processing System Based on DSP + FPGA Architecture. *Ship Electronic Engineering* 7.

Gómez-Luna, J., J. González-Linares, J. Benavides & N. Guil (2009). Parallelization of a video segmentation algorithm on CUDA-enabled graphics processing units. *Euro-Par 2009 Parallel Processing* 924–935.

Gong, M., A. Langille & M. Gong (2005). Real-time image processing using graphics hardware: A performance study. In: *Lecture Notes in Computer Science*, vol. 5636, 1217–1225. Berlin: Springer.

Granlund, K. (2004). *Laitetekniikka*. Helsinki: Docendo.

He, L. & Z. Zhang (2006). Real-time whiteboard capture and processing using a video camera for remote collaboration. *IEEE Transactions on Multimedia* 9:1, 198–206.

Hennessy, J., D. Patterson, D. Goldberg & K. Asanovic (2003). *Computer Architecture: A Quantitative Approach*. Burlington: Morgan Kaufmann Publishers.

Hyde, R. (1996). The Art of Assembly Language Programming [online]. Unpublished book [cited 26.11.2010]. Available at: <URL:http://www.arl.wustl.edu/~lockwood/class/cs306/books/artofasm/fwd.html>.

Hyde, R. (2001). The Art of Assembly Language [online]. Unpublished book [cited 3.1.2011]. Available at: <URL:http://212.14.233.133/portal_resources/downloads/programming/assembly_language32bit_edition.pdf>.

Imizu, S., Y. Kato, A. Sangli, D. Oguri & H. Sano (2008). Assessment of incomplete clipping of aneurysms intraoperatively by a near-infrared indocyanine green video angiography (Niicg-Va) integrated microscope. *Minimally Invasive Neurosurgery* 51:4, 199–203.

Kayi, A., T. El-Ghazawi & G. Newby (2009). Performance issues in emerging homogeneous multi-core architectures. *Simulation Modelling Practice and Theory* 17:9, 1485–1499.

Kehtarnavaz, N. & M. Gamadia (2006). *Real-time Image and Video Processing: From Research to Reality*. San Rafael: Morgan & Claypool Publishers.

Kogure, K. & E. Choromokos (1969). Infrared absorption angiography. *Journal of Applied Physiology* 26:1, 154.

Kohno, T., T. Miki, K. Shiraki, K. Kano, M. Matsushita, K. Hayashi & J. De Laey (1999). Subtraction ICG angiography in Harada's disease. *British Journal of Ophthalmology* 83:7, 822–833.

Krause, A. (2007). *Foundations of GTK+ development*. New York: Springer.

Kulyabina, T. & V. Kochubey (2006). The interaction of indocyanine green with blood plasma and features of crystallization. In: *Proceedings of SPIE*, vol. 6163. Bellingham: SPIE.

Kumar, R. & J. Friedman (2009). Intraoperative angiography during cerebral aneurysm surgery. *Neurocritical Care* 11:2, 299–302.

Kuroda, I. & T. Nishitani (2002). Multimedia processors. *Proceedings of the IEEE* 86:6, 1203–1221.

Landré, J. & F. Truchetet (2007). Optimizing signal and image processing applications using Intel libraries. In: *Proceedings of SPIE*, vol. 6356. Bellingham: SPIE.

Landsman, M., G. Kwant, G. Mook & W. Zijlstra (1976). Light-absorbing properties, stability, and spectral stabilization of indocyanine green. *Journal of Applied Physiology* 40:4, 575.

Lapsley, P., J. Bier, E. Lee & A. Shoham (1996). *DSP Processor Fundamentals: Architectures and Features*. New York: Wiley-IEEE Press.

Lee, T. & P. Chang (2008). Real-time foreground segmentation for the moving camera based on H. 264 video coding information. In: *Future Generation Communication and Networking (FGCN 2007)*, vol. 1, 385–390. Washington: IEEE.

Leica microsystems (2008). Leica FL800. Integrated vascular fluorescence [online]. Product brochure [cited 28.2.2010]. Available at: <URL:http://www.leica-microsystems.com/fileadmin/downloads/Leica%20FL800/Brochures/Leica_FL800_bro_en.pdf>.

Love, R. (2010). *Linux Kernel Development*. 3. ed. Upper Saddle River: Pearson Education.

Luebke, D., M. Harris, J. Krüger, T. Purcell, N. Govindaraju, I. Buck, C. Woolley & A. Lefohn (2004). Gpgpu: General purpose computation on graphics hardware. In: *ACM SIGGRAPH 2004 Course Notes*, Article 33. New York: ACM.

Luebke, D. & G. Humphreys (2007). How GPUs work. *IEEE Computer Society* 40:2, 126–130.

Mathivanan, N. (2003). *Microprocessors, PC Hardware and Interfacing*. New Delhi: PHI Learning Private Limited.

McCool, M. (2007). Signal processing and general-purpose computing and GPUs. *IEEE Signal Processing Magazine* 24:3, 109–114.

Möller, T., T. Akenine-Möller, E. Haines & N. Hoffman (2008). *Real-time Rendering*. Natick: A K Peters Limited.

Murdocca, M. & V. Heuring (2000). *Principles of Computer Architecture*. Upper Saddle River: Pearson Education.

Nair, P. (2008). Image and video processing using FPGA technology for medical applications. In: *The Latest Technologies and Tools in Electronic Design*, 1–8. Herts: IET.

Nickolls, J. & W. Dally (2010). The GPU computing era. *IEEE Micro* 30:2, 56–69.

Nienstedt, W., ed. (2006). *Lääketieteen Termit*. 4. ed. Helsinki: Duodecim.

NVIDIA (2010a). Bringing the high-definition home theater experience to your PC and mobile device [online]. Official corporation website [cited 22.9.2010]. Available at: <URL:http://www.nvidia.com/page/purevideo.html>.

NVIDIA (2010b). NVIDIA performance primitives [online]. Official corporation website [cited 23.9.2010]. Available at: <URL:http://developer.nvidia.com/object/npp_home.html>.

Owens, J., M. Houston, D. Luebke, S. Green, J. Stone & J. Phillips (2008). GPU computing. *Proceedings of the IEEE* 96:5, 879–899.

Owens, J., D. Luebke, N. Govindaraju, M. Harris, J. Kruger, A. Lefohn & T. Purcell (2007). A survey of general-purpose computation on graphics hardware. *Computer Graphics Forum* 26:1, 80–113.

Paul, B. (2000). Introduction to the direct rendering infrastructure [online]. Linux World Conference [cited 5.10.2010]. Available at: <URL:http://www.sfdesignsalon.com/opengl/dri_intro.pdf>.

Peng, L., J. Peir, T. Prakash, C. Staelin, Y. Chen & D. Koppelman (2008). Memory hierarchy performance measurement of commercial dual-core desktop processors. *Journal of Systems Architecture* 54:8, 816–828.

Pieters, B., D. Van Rijsselbergen, W. De Neve & R. Van de Walle (2007). Performance evaluation of H. 264/AVC decoding and visualization using the GPU. In: *Proceedings of SPIE: The International Society for Optical Engineering*, 5–18. Bellingham: SPIE.

Porter, T. & T. Duff (1984). Compositing digital images. *ACM SIGGRAPH Computer Graphics* 18:3, 253–259.

Poynton, C. (2003). *Digital Video and HDTV: Algorithms and Interfaces*. Burlington: Morgan Kaufmann Publishers.

Puaut, I. & C. Pais (2007). Scratchpad memories vs locked caches in hard real-time systems: A quantitative comparison. In: *Proceedings of the Conference on Design, Automation and Test in Europe*, 1489. San Jose: EDA Consortium.

Raabe, A., J. Beck, R. Gerlach, M. Zimmermann & V. Seifert (2003). Near-infrared indocyanine green video angiography: A new method for intraoperative assessment of vascular flow. *Neurosurgery* 52:1, 132.

Raabe, A., J. Beck & V. Seifert (2005a). Technique and image quality of intraoperative indocyanine green angiography during aneurysm surgery using surgical microscope integrated near-infrared video technology. *Zentralblatt für Neurochirurgie* 66:1, 1–6.

Raabe, A., P. Nakaji, J. Beck, L. Kim, F. Hsu, J. Kamerman, V. Seifert & R. Spetzler (2005b). Prospective evaluation of surgical microscope-integrated intraoperative near-infrared indocyanine green videoangiography during aneurysm surgery. *Journal of Neurosurgery: Pediatrics* 103:6.

Ranganathan, P., S. Adve & N. Jouppi (1999). Performance of image and video processing with general-purpose processors and media ISA extensions. In: *Proceedings of the 26th Annual International Symposium on Computer Architecture*, 124–135. Washington: IEEE Computer Society Press.

Reichman, J. (2000). *Handbook of Optical Filters for Fluorescence Microscopy*. Vermont: Chroma Technology Corp.

Revel, D., C. Cowan, D. McNamee, C. Pu & J. Walpole (1997). Predictable file access latency for multimedia. In: *Proceedings of 5th International Workshop on Quality of Service*, 401–404.

Ritger, A. (2006). Using the existing XFree86/X.Org loadable driver framework to achieve a composited X desktop. White paper, NVIDIA corp. Available at: <URL:http://http.download.nvidia.com/developer/presentations/2006/xdevconf/compositing-with-current-framework.pdf>.

Rosen, R., M. Hathaway, J. Rogers, J. Pedro, P. Garcia, G. Dobre & A. Podoleanu (2009). Simultaneous OCT/SLO/ICG imaging. *Investigative Ophthalmology & Visual Science* 50:2, 851–860.

Schimek, M., B. Dirks, H. Verkuil & M. Rubli (2008). Video for Linux two API specification (0.24) [online]. Online software documentation [cited 8.12.2010]. Available at: <URL:http://v4l2spec.bytesex.org/spec/>.

Shen, G., G. Gao, S. Li, H. Shum & Y. Zhang (2005). Accelerate video decoding with generic GPU. *IEEE Transactions on Circuits and Systems for Video Technology* 15:5, 685–693.

Shimpi, A. (2010). Intel's Sandy Bridge architecture exposed [online]. A tech review website [cited 2.12.2010]. Available at: <URL:http://www.anandtech.com/show/3922/intels-sandy-bridge-architecture-exposed/5>.

Stallings, W. (2009). *Computer Organization and Architecture: Designing for Performance*. 6th ed. Upper Saddle River: Pearson Education.

Stallman, R. (1999). The GNU operating system and the free software movement. In: *Open Sources: Voices from the Open Source Revolution*, 53–70. Sebastopol: O'Reilly Media.

Stankovic, J. & R. Rajkumar (2004). Real-time operating systems. *Real-Time Systems* 28:2, 237–253.

Sugita, K., T. Naemura & H. Harashima (2003). Performance evaluation of programmable graphics hardware for image filtering and stereo matching. In: *Proceedings of the ACM Symposium on Virtual Reality Software and Technology*, 176–183. New York: ACM.

Takagi, Y., K. Kikuta, K. Nozaki, K. Sawamura & N. Hashimoto (2007). Detection of a residual nidus by surgical microscope-integrated intraoperative near-infrared indocyanine green videoangiography in a child with a cerebral arteriovenous malformation. *Journal of Neurosurgery: Pediatrics* 107:5, 416–418.

Talla, D., L. John & D. Burger (2003). Bottlenecks in multimedia processing with SIMD style extensions and architectural enhancements. *IEEE Transactions on Computers* 52:8, 1015–1031.

Talla, D., L. John, V. Lapinskii & B. Evans (2000). Evaluating signal processing and multimedia applications on SIMD, VLIW and superscalar architectures. In: *Proceedings of the International Conference on Computer Design*, 163–174. Washington: IEEE Computer Society Press.

Taylor, S. (2007). *Optimizing Applications for Multi-Core Processors, Using the Intel Integrated Performance Primitives*. 2. ed. Santa Clara: Intel Press.

Taymans, W., S. Baker, A. Wingo, R. Bultje & S. Kost (2011). GStreamer application development manual (0.10.32) [online]. Official software documentation [cited 12.2.2011]. Available at: <URL:http://gstreamer.freedesktop.org/data/doc/gstreamer/head/manual/manual.pdf>.

Thompson, R. (2006). *Quartz 2D Graphics for Mac OS X® Developers*. Boston: Addison-Wesley.

Thompson, R. & B. Thompson (2003). *PC Hardware in a Nutshell*. 3. ed. Sebastopol: O'Reilly & Associates.

Uluç, K., G. Kujoth & M. Başkaya (2009). Operating microscopes: Past, present, and future. *Journal of Neurosurgery: Pediatrics* 27:3.

van der Wolf, P. & T. Henriksson (2008). Video processing requirements on SoC infrastructures. In: *Proceedings of the Conference on Design, Automation and Test in Europe*, 1124–1125. New York: ACM.

Wallossek, I. (2010). The 2D GDI for Windows XP through Windows 7, in detail [online]. A tech review website [cited 24.9.2010]. Available at: <URL:http://www.tomshardware.com/reviews/2d-windows-gdi,2547-2.html>.

Welsh, M. (2006). *Running Linux*. 5. ed. Sebastopol: O'Reilly Media.

Wieber Jr, J. & G. Zopetti (2008). How to use intrinsics [online]. Official corporation website [cited 26.11.2010]. Available at: <URL:http://software.intel.com/en-us/articles/how-to-use-intrinsics/>.

Wiederspahn, M. (2004). UNIQUE OPMI Pentero [online]. Product brochure [cited 29.10.2010]. Available at: <URL:http://www.zeiss.de/C1256A770030BCE0/WebViewAllE/67FFDB168894BE16C1256E8600293C3C>.

Wikipedia contributors (2010a). Comparison of container formats [online]. Wikipedia, The Free Encyclopedia [cited 21.11.2010]. Available at: <URL:http://en.wikipedia.org/wiki/Comparison_of_container_formats#cite_note-WMV_in_MP4-23>.

Wikipedia contributors (2010b). Motherboard [online]. Wikipedia, The Free Encyclopedia [cited 7.12.2010]. Available at: <URL:http://en.wikipedia.org/wiki/Motherboard>.

Wikipedia contributors (2010c). RGB color model [online]. Wikipedia, The Free Encyclopedia [cited 1.8.2010]. Available at: <URL:http://en.wikipedia.org/wiki/Truecolor#Truecolor>.

Wikipedia contributors (2011a). Libavcodec library [online]. Wikipedia, The Free Encyclopedia [cited 5.3.2011]. Available at: <URL:http://en.wikipedia.org/wiki/Libavcodec>.

Wikipedia contributors (2011b). Video acceleration API [online]. Wikipedia, The Free Encyclopedia [cited 15.1.2011]. Available at: <URL:http://en.wikipedia.org/wiki/Video_Acceleration_API>.

Wikipedia contributors (2011c). X video extension [online]. Wikipedia, The Free Encyclopedia [cited 2.3.2011]. Available at: <URL:http://en.wikipedia.org/wiki/X_video_extension>.

Wikström, K. (2010). Uusinta PC-tekniikkaa: Voimaa ja vauhtia. *Prosessori* 2010:6-7, 20–25.

Williams, R. (2006). *Real-time Systems Development*. Burlington: Butterworth-Heinemann.

Williams, R. (2010). Intel's H55, H57 & Q57 chipsets. A tech review website [cited 2.12.2010]. Available at: <URL:http://techgage.com/article/intels_32nm_clarkdale_-_nehalem_for_everyone/4>.

Yang, Z., Y. Zhu & Y. Pu (2008). Parallel image processing based on CUDA. In: *Proceedings of the 2008 International Conference on Computer Science and Software Engineering*, vol. 3, 198–201. Washington: IEEE Computer Society Press.

Yaşargil, M. (1984). *Microsurgical Anatomy of the Basal Cistern and Vessels of the Brain, Diagnostic Studies, General Operative Techniques and Pathological Considerations of the Intracranial Aneurysms*. Stuttgart: George Thieme Verlag.

APPENDICES

APPENDIX 1. Transcript of a meeting at Töölö Hospital, 12.1.2009

Present: M.Sc. student Kari Koivuporras, Professor Jarmo Alander, neurosurgeon Martin Lehečka

Evaluation of the current situation:

- ICGA is an online tool for the surgeons and thus the focus should be in video processing that can be done online.
- The application needs to be simple, easy to use and fast, because everything in the operating room needs to work fluently. Any settings tweaking should happen before surgery.
- The biggest challenge today is the momentariness of (ICG) video. There is a need for a software that provides a sort of backup memory for the surgeons. Instead of memorizing all the details in the ICG video, there should be functionality to freely pause, rewind, and process the video, or maybe compare images of before and after ICG injections side by side. This would be especially important in aneurysm surgery.
- The pseudo coloring of ICG video could have potential. It may improve contrast and aid in diagnosis, but more important at the moment is to begin research to find the best solutions for implementing the prototype imaging station. It is also important to find more suitable image processing operations, which would help in diagnostic work.

A few ideas on what could be developed, formulated into a prioritized list by Martin Lehečka:

1. Combination of the information in the ICG and anatomical images. (Alander suggested alpha blending for this task)
2. Analyzing the behavior of ICG over time: Lehečka liked the features of Amide (an open source medical imaging data examiner), where activities in specific regions in video image can be tracked in time. A blood flow analysis tool that would visualize the temporal changes of flow of blood and provide other useful information like direction of flow would be ideal.
3. Detecting the thickness of vein walls based on intensity of fluorescence: a bright spot in the image suggests a thin wall. It was however acknowledged right away that it could be difficult to exclude other factors that affect intensity captured by camera.
4. Finding the most suitable way of presenting ICG-images: pseudo coloring, subtraction to exclude background etc.

❺ Implementing a tool for measuring tissue oxidation using ICGA. Maybe it would be possible to measure the intensity of fluorescence before and after a brain bypass surgery and conclude how much tissue oxidation was improved. Again the same challenges as with the previous point.

APPENDIX 2. Transcript of a meeting at Töölö Hospital, 22.2.2010

Present: M.Sc. student Marianna Kontulainen, M.Sc. student Kari Koivuporras, Professor Jarmo Alander, neurosurgeons Martin Lehečka and Aki Laakso

Evaluation of the current situation:

- General discussion about ICGA.
- Discussed about the optional blood flow dynamics tool (FLOW 800) for OPMI Pentero. The surgeons would be interested in something similar. At the moment the 60.000€ price for FLOW 800 was considered too high.
- The high-resolution camera attached to the co-observation tube is shown in a big screen HDTV. The zoom and field of view are different from the internal cameras though. Moreover, rotating the microscopes imaging head makes the high-res camera image rotate as well, because the camera does not rotate with the optics.
- 3D imaging also available for OPMI Pentero for something close to 100.000€.
- Useful tasks listed by the surgeons:
  - Recording ICGA, with a possibility to rewind back and forth
  - Combining the ICG and anatomic images (alpha blending)
  - Tissue oxidation measurements (nowadays they use special sensors)
  - Determining blood filling direction from ICG video (blood flow analysis)
- The microscope does record both the ICG and anatomic video on the internal HDD.
- Discussions about the possibility of including Computed Tomography (CT) viewing software into the imaging station. That way someone in the operating room could compare findings of ICG and CT, for example.

APPENDIX 3. Transcript of a meeting at Töölö Hospital, 12.5.2010

Present: M.Sc. student Kari Koivuporras, Professor Jarmo Alander, neurosurgeon Aki Laakso

Evaluation of the current situation:

- Early GUI mock-up presented by Kari Koivuporras
- Image processing demo with harpia by Kari Koivuporras - not done due to harpia problems.
- More information on how the operating microscope actually behaves: The output of the operating microscope is always on. Normally the visible light video is outputted. Surgeons trigger the ICG mode from a button in the scope handle causing ICG to be switced to the output.
- Hardware: The idea of buying two cameras and an optical divider attached to the extra camera slot of the operating microscope was brought up again as the only thinkable option to implement real alpha blending of visible and ICG channels.
- Project guidelines: Hardware purchasing was agreed to be decided at a later time. It was agreed to start with a scheme where a frame grabbed from visible light stream will be used as a background image for alpha blending with ICG video.
- Software design: The base design presented was approved. No criticism was brought up so we will continue with the idea of two-part GUI logic, where the application is statically in a configuration/preview mode and automatically switches to a playback mode when ICG stream is detected in input. Image processing methods can be changed also on-the-fly in playback mode and the video can be paused and rewinded both ways.
- Feature requests: Possibility to display ICG video and the still visible light image side-by-side in playback mode.

APPENDIX 4. Transcript of a meeting at Töölö Hospital, 6.9.2010

Present: M.Sc. student Kari Koivuporras, Professor Jarmo Alander, neurosurgeon Aki Laakso

Evaluation of the current situation:

- New GUI design presented by Koivuporras. iStation now consists of a single view where different modes are selectable with an exclusive checkbox group on top.
- The logic of the design is accepted, no suggestions about look'n'feel.

Feature requests:

- Alpha blending: not feasible with current hardware. 1) Still frame to video found unhelpful and 2) HD camera rotation in relation to other cameras destroys possibility for HD-to-SD blending.
- Pseudo-coloring: a nice feature, but not a vital one at this point.
- Recording a patient archive: SD recording already exists in two forms; DV recording from operating room's video bus and internal HDD of the microscope. There is no need for further at the moment.
- Feature request: Synchronized rewind possibility to dual-view.
- Feature request: Dual-view with HD anatomic and SD ICG video.

Future plans:

- Laakso wanted the final release platform to be a laptop. It was assumed though that finding a composite frame grabber for laptops would be difficult and fitting two of them into a laptop even more so.
- Tasks for Aki Laakso: see what outputs there are in the Storz's video hub and operating microscope.
- Tasks for Kari: 1) see what hardware is needed to interface the inputs 2) make the adjustments to the application 3) get the hardware.

APPENDIX 5. Transcript of a meeting at Töölö Hospital, 15.2.2011

Present: M.Sc. student Kari Koivuporras, Professor Jarmo Alander, neurosurgeons Martin Lehečka and Aki Laakso

Evaluation of the current situation:

- Reiteration of the GUI design presented by Koivuporras. iStation now has a full menu and toolbar. Three states for live streaming, recording and playback. There are two view modes accessible in any state: one for viewing one video and another for side-by-side viewing. No video post-processing yet.
- The logic of the design is accepted, no suggestions about look'n'feel.

Feature requests:

- Slow down of video playback, low priority.
- Sync for different length videos, high priority.
- Fine-grained rewind (with arrow keys), medium priority.
- Tooltips and other instruction improvements, medium priority.

Future plans:

- The PC hardware will be purchased by the Hospital, Koivuporras will send preferred specs to Laakso.
- iStation will still go through an iteration of new features and testing with the actual frame grabbers.
- University of Vaasa will lend the frame grabbers for testing the feasibility of the application.