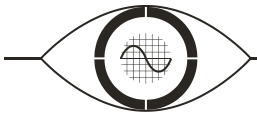

Lehrstuhl für Bildverarbeitung
Institute of Imaging & Computer Vision

RealTimeFrame A Real Time Processing Framework for Medical Video Sequences

Gross, Sebastian and Stehle, Thomas
Institute of Imaging and Computer Vision
RWTH Aachen University, 52056 Aachen, Germany
tel: +49 241 80 27860, fax: +49 241 80 22200
web: www.lfb.rwth-aachen.de

in: Acta Polytechnika Journal of Advanced Engineering. See also BibTeX entry below.

BibTeX:

```
@article{Gross2008c,
  author = {Gross, Sebastian and Stehle, Thomas},
  title = {RealTimeFrame  A Real Time Processing Framework for Medical Video
Sequences},
  journal = {Acta Polytechnika Journal of Advanced Engineering},
  year = {2008},
  volume = {48},
  number = {3},
  month = {June},
  ISSN = {ISSN 1210-2709},
  owner = {gross},
  timestamp = {2008.12.10}
}
```


RWTH AACHEN
RHEINISCH-WESTFÄLISCHE TECHNISCHE HOCHSCHULE AACHEN

RealTimeFrame – A Real Time Processing Framework for Medical Video Sequences

S. Gross, T. Stehle

Imaging technology is highly important in today's medical environments. It provides information upon which the accuracy of the diagnosis and consequently the wellbeing of the patient rely. Increasing the quality and significance of medical image data is therefore one the aims of scientific research and development.
We introduce an integrated hardware and software framework for real time image processing in medical environments, which we call RealTimeFrame. Our project is designed to offer flexibility, easy expandability and high performance. We use standard personal computer hardware to run our multithreaded software. A frame grabber card is used to capture video signals from medical imaging systems. A modular, user-defined process chain performs arbitrary manipulations on the image data. The graphical user interface offers configuration options and displays the processed image in either window or full screen mode. Image source and processing routines are encapsulated in dynamic library modules for easy functionality extension without recompilation of the entire software framework. Documented template modules for sources and processing steps are part of the software's source code.

Keywords: Medical image processing, software framework, real time image processing, multithreading.

1 Introduction

Medical imaging is important in today's clinical environments. Multiple imaging modalities such as X-ray angiography or endoscopy gather information on the patient's well-being and medical status [1]. Visual information is essential for the medical practitioner to confirm a diagnosis and to administer a therapy.

An example is coronary angiography, where the inflow and outflow of a contrast agent in the coronary arteries provide decisive information for the planning of interventions [2]. For example, image processing algorithms can be used for image enhancement, which can lead to a reduction of the required x-ray dose for an accurate diagnosis.

Medical image information often has to be acquired from various imaging modalities, enhanced, analyzed, interpreted, and displayed for the medical practitioner. Integration of these steps in a single software framework is a complex task, but also offers the chance to create a foundation for the operation and development of medical image processing algorithms.

The rest of the paper is organized as follows. Chapter two lists the requirements for a real time software framework for medical image data processing. Chapter three illustrates the work flow for our software framework. A more detailed view on the software design pattern and techniques used in our software framework is the focus of chapter four. The fifth chapter concentrates on the hardware and software we used to implement our project. The results and a report onclinical trials are highlighted in chapter six. Chapter seven sums up the main points and concludes with final remarks.

2 Requirements

The intention of our project was to design a platform for complex image and video processing algorithms. Image frames are acquired from a medical image source, often processed with appropriate algorithms and finally displayed for the medical practitioner. The aim of the development was to combine these steps in a fast and efficient framework.

A catalogue of criteria to be met by *RealTimeFrame* was developed. The requirements can be divided into three categories: features, performance and expandability, which are described in the following sections.

2.1 Features

Images have to be acquired from a medical image source and may be preprocessed for display in real time. During interventions, video data is supplied via the S-video connector of the endoscopy system's camera control unit. Thus, frame grabber functionality is mandatory to digitise the video stream. For testing and presentation purposes it must be possible to read video streams from the hard drive to simulate a real medical image source.

A further requirement is that arbitrary operations can be performed on the image data. Such operations could enhance image quality and significance or store the data as video or image sequences on the hard drive.

The results must be displayed with minimal time delay for use during diagnosis or surgery, and a full screen mode is needed to provide the best possible view for the medical practitioner.

2.2 Performance

The images processed are 720×576 pixels in size. The refresh rate required for clinical evaluation is 50 fields per second. The images are RGB coded, and each of the three channels is sampled with 8-bit colour depth. Handling such a stream leads to a total of 31,104,000 bytes (=29.66 megabytes) of image data per second. The delay between image acquisition and image display must not exceed 100 ms. Otherwise, a physician would experience an irritating time lag during the intervention.

2.3 Expandability

Expandability was mandatory for *RealTimeFrame*. The aim was to create a system flexible enough to be constantly expanded with further image sources and image processing algorithms, while providing performance for real time application in the medical field.

Ideally, additional image sources and processing algorithms can be integrated into the program without recompiling of the whole software framework.

3 Work flow

RealTimeFrame was designed to provide a flexible platform for medical image processing. Fig. 1 illustrates the work flow of the framework.

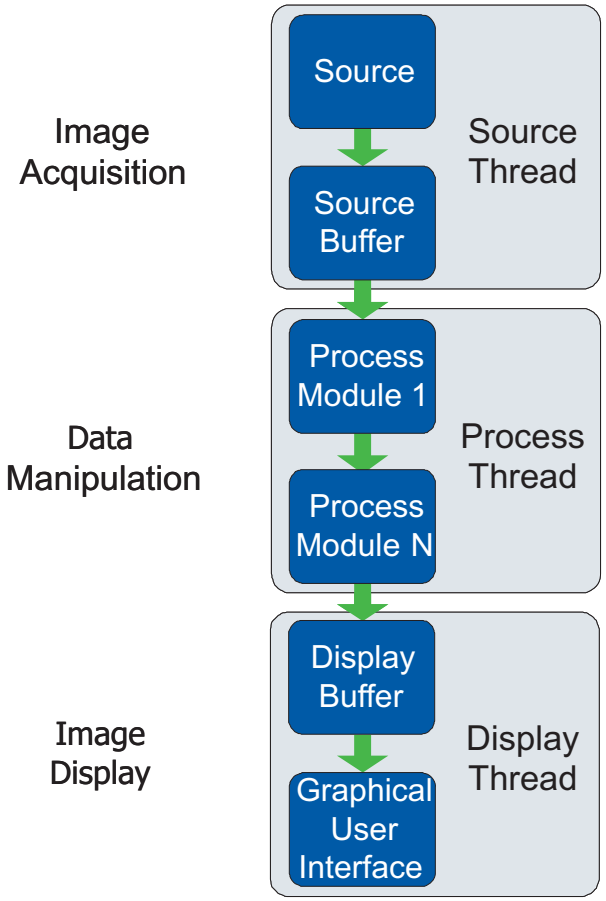

Fig. 1: Work flow concept for the *RealTimeFrame* software framework

3.1 Image Acquisition

A source is chosen by the user to generate image data. This source is encapsulated in a DLL file and uses the interface defined by the framework to place image data in the source buffer.

The image source can be replaced by any module implementing the image retrieval process for a new source. No change to the framework has to be made. An example of this would be the replacement of the installed frame grabber card by another model. A new source module can be implemented and then easily selected via the GUI.

3.2 Image processing

Data from the source buffer is sent to a user defined chain of processing modules implementing image manipulation and processing steps. These steps may include algorithms for improving the image quality or significance, filtering, and storing in videos or file sequences. The functionality of the processing modules is again encapsulated in DLL files.

The process chain can be configured to hold an arbitrary combination of processing modules, and is not limited by any implemented restriction. With respect to real time processing and the required CPU time for the process chain, however, there are limitations.

3.3 Graphical user interface

The graphical user interface (GUI) delivered with our *RealTimeFrame* is depicted in Fig. 2. It offers control options for all operations performed by RealTimeFrame on the left side of the window.

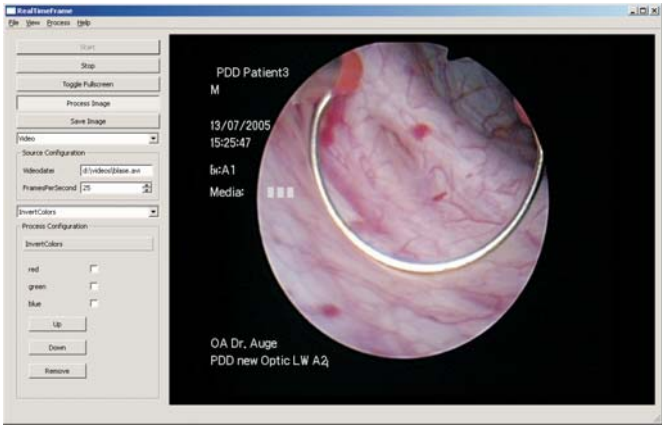

Fig. 2: Screenshot of the main window, showing configuration buttons and a processed medical image

Drop down menus list all available sources and processing modules. Parameters for the selected modules are represented by check boxes, spin boxes, and text lines. These parameters and their visual representations are automatically extracted from the associated modules and integrated into the GUI. The process chain can be configured via GUI and turned on and off during an intervention.

The right hand side of Fig. 2 displays the processed image. A full screen mode is available, offering an enlarged image display area during medical interventions.

3.4 Available modules

Video source plug-ins shipped with *RealTimeFrame* are SourceFrameGrabber for accessing Euresys Picolo series frame grabber cards, SourceVideo for accessing video files and SourceSingleImage for loading image sequences from the hard drive.

Processing modules shipped with *RealTimeFrame* include a median filter, a colour channel inversion module and the Canny-Edge-Detector.

Documented examples and ready-to-use templates for source and processing modules are included in the source

code. Therefore, creating new modules based on the template files is fast and relatively easy.

4 Software concepts

The premises for the software require careful planning, and especially the postulated flexibility and expandability have to be considered. An object-orientated approach is a prerequisite for successful creation of an easily extendable software framework.

4.1 Model-View-Controller concept

The Model-View-Controller (MVC) concept is a well-known and widely-used pattern in object orientated software architecture [4]. The central idea is to increase the flexibility of the software project, to simplify the reuse of code parts and to shorten the time needed for familiarisation with the code.

The concept introduces functional units and enforces encapsulation of the associated code. The software developer can concentrate on certain parts of the code, which can be modified or even replaced without any change to the rest of the framework. The only necessity is to leave the interfaces to the other program parts unchanged. Applying the MVC concept, the functionality of a program can be divided into three functional blocks:

- Model
- View
- Controller

Fig. 3 shows an organisational diagram of *RealTimeFrame*, indicating the separation of program parts into functional blocks.

The model is responsible for storage of the processed and unprocessed image data. It does not have any knowledge of the source of the data or its meaning. In RealTimeFrame,s case, the model does not hold any information on whether the stored frame is from a video or a single image, a frame grabber or any other image source. The model does not know if the data will be processed or displayed at all. It simply offers memory to store and read the frame data.

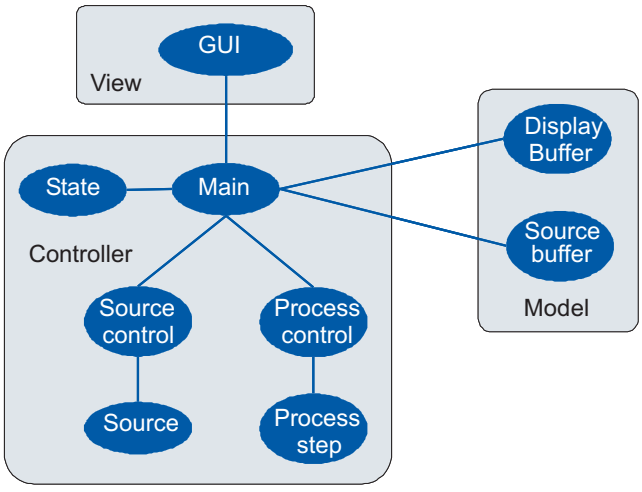

Fig. 3: Model-View-Controller concept for the *RealTimeFrame* software framework

The task of the view is to display data. The view is separated from acquisition, storage or processing of the data. Data handed over to the view module is presented to the user.

The controller is responsible for modifying the data and for administering the image manipulation process. It is separated from the data storage in the view module and the representation of the data in the view module.

4.2 Multithreading

Multithreading enables the work load to be spread into different work flows. These work flows, so-called threads, run simultaneously and independently. Interaction is possible between threads, and enables them to exchange data, to wait for signals or to start and stop each other. This also enables user interaction while running program tasks in the background.

However, multithreaded programming has some drawbacks [5]. The method usually leads to less understandable code. It is left to the programmer to identify all possible problems with thread interactions and to solve them. Bug tracking in these cases is difficult and time consuming.

4.3 Parallel processing

Performance is one of *RealTimeFrame*'s key requirements, as we are working under real time conditions. Performance can be boosted in several ways. One of the ways, if a multiprocessor computer is available, is parallel processing.

Having more than one processor core enables the programmer to distribute the work load. Theoretically, one would expect four cores to work four times faster than a single core. This, however, is not the case, for the following reasons:

- Coordinating the distribution and ensuring load balancing for four processors will create a computational overhead.
- If the process waits for hardware resources or user input, the number of processors is of no consequence for the duration of the delay.
- Not all problems can be distributed evenly or at all. Some results may depend on others and calculations may need previous results. This prevents operations from being run simultaneously.

Even taking these constraints into account, there is still a considerable performance increase when running applicable tasks on multiple CPUs and, therefore, we make use of parallel processing whenever possible, as image manipulation algorithms can often be vastly accelerated by parallel processing, in comparison with a single processor system.

4.4 Dynamic link libraries

The linking process was invented to bind multiple object-files created by the compiler from different source code files into one single executable. Libraries were used to supply additional functionality, and to reduce code size and compile time. An often used library was likely compiled into multiple programs. Thus it finally used a large amount of hard drive space and, even more important, a large amount of the computer's main memory.

In the late 1960s dynamic linking was developed to cope with problems of disk space. The linker process only records the necessary library and the entry point for the functions

called, but does not include the object code of the library itself in the binary. A library only has to be stored once on a hard drive of the computer and its path must be made known to the operating system. All programs can then load this library if needed to perform their operations.

Dynamic loading on the other hand seeks to free up system memory by directing all programs in need of a certain library to the same memory location. The library is loaded only once. Thus, multiple programs use the same instance of the library, which is unloaded when the last program that depends on the library exits.

Dynamic linking and loading offers more advantages than just the originally intended effect of saving disc space and memory. Replacing a library with a new version takes effect for all programs which refer to it. The programs do not need to be changed and are completely separated from the implementation of the library. There is no need to recompile the programs, and introducing updated versions or additional functionalities in libraries is therefore uncomplicated. However, the library creator has to ensure that the library's interface remains unchanged and the entry points referenced in the code still exist.

The expandability postulated for *RealTimeFrame* can be achieved with dynamic link libraries (DLL) without the need to recompile the main program. We decided to define an interface for all source and processing modules to use. At start-up RealTimeFrame will check for available DLLs and load each of them.

Adding a source or processing module therefore only requires moving the DLL into the appropriate directory. This will help developing additional functionality and servicing customer deployed copies of *RealTimeFrame*, as new or updated functionality can be installed easily and no change to the rest of the framework is necessary.

5 Development Platform

We chose our hardware and software to fit the requirements defined in Chapter 2.

5.1 Software

The software was chosen for performance, features and flexibility as well as for improved and simplified implementation.

We opted to use Windows XP Professional rather than Windows Vista. Windows Vista was introduced on January 30, 2007, and users then experienced various problems with the new operating system, including numerous driver compatibility and availability issues. This convinced us to use Windows XP Professional, as the manufacturers of our hardware provided drivers for this operating system and most issues with this software platform are well-known and have been solved in the meantime.

The development platform was Microsoft Visual Studio 2003.Net. We decided to replace the standard Microsoft compiler with the Intel C++ Compiler 10.0 for Windows, as it provides better support for multithreading and generates faster code.

Apart from the choice of the operating system and the integrated development environment (IDE), the choice of the underlying libraries has a major influence on the implementation. Many functions and algorithms are provided by libraries. Using these well tested, reliable libraries speeds up the implementation process.

The graphical user interface (GUI) is based on Trolltech's Qt. It offers fast and comfortable GUI design as well as an interface for multithreading and inter-thread communication. The images are displayed using Microsoft DirectX9 to take advantage of hardware acceleration.

We use Intel's Open Computer Vision library (OpenCV) for image handling and manipulation. As OpenCV uses the outdated Video for Windows interface to access video files, we replaced this functionality with FFmpeg's libavcodec. Thus, *RealTimeFrame* is able to handle video files larger than 2 GB, which is reached rapidly when we store medical video data in high quality.

5.2 Hardware

Today's hardware offers ample performance to create a system capable of acquiring images, manipulating image data and displaying results with only an insignificant lag. We decided to use standard personal computer hardware components as they offer solid performance at a reasonable price.

Our hardware platform consists of two Intel Xeon 5140 Dual Core processors with 2.2 GHz. The two CPUs are placed on a Tyan server motherboard with 4 GB RAM.

We use two 70 GB hard drives in a raid setup for the operating system, *RealTimeFrame*, and the development tools. Two 500 GB SATA2-hard drives offer performance for simultaneous data storage and reading.

We use an NVIDIA GeForce 7950 GX2 graphics adapter which offers the possibility of performing matrix calculation operations via a programmable interface. For video acquisition from medical devices, we installed a Euresys Picolo series industrial frame grabber card.

6 Results

RealTimeFrame is run on two identical systems. One is the development platform, which remains at the Institute of Imaging and Computer Vision, RWTH Aachen University. The other system is used in clinical trials.

RealTimeFrame has proved to be of exceptional value for the development, testing, and clinical evaluation of algorithms for medical image processing.

Our framework was recently used as a basis for the implementation of our fascia enhancement algorithm [6]. It currently serves as the operating platform in the associated clinical trials during surgical interventions.

Clinical trials have shown that RealTimeFrame introduces an acceptable time lag. The complete system, including the endoscope used in the interventions, delayed the image by about 90–110 ms on average. Activating the fascia enhancement algorithm increases this number to 130 ms. The endoscope alone causes an average lag of 60 ms. Taking this into account, the postulated maximum increase by 100 ms has not been exceeded.

The software framework itself causes a very low CPU load. At a refresh rate of 25 frames per second, about 2 % of the CPU capacity on our presented hardware platform is con-

sumed by image acquisition via a frame grabber card and display on the screen.

7 Summary and conclusion

RealTimeFrame is a solution for real time video processing in medical environments. It provides services for image acquisition from various image sources. The data is processed by a chain of processing steps before it is displayed.

The process chain may include filtering, analysis, and saving frame sequences or entire videos in an arbitrary combination.

We have designed source and processing modules to encapsulate the functionality. The module initialisation is performed at start up and the GUI offers control and configuration options for all RealTimeFrame plug-ins found. Thus, adding new modules is straightforward, and is done without changes to the framework.

RealTimeFrame meets the requirements listed in chapter two. The concepts of the implementation were illustrated in chapters three to five. Based on the results highlighted in chapter six, we are convinced that *RealTimeFrame* will be of exceptional value for medical image processing in clinical environments and also in the development of new algorithms in the future.

8 Acknowledgments

We would like to thank Prof. T. Aach of the Institute of Imaging and Computer Vision, RWTH Aachen University for supervising the research described in this paper.

We would also like extend our thanks to Prof. Dr. med. Bruch and Dr. med. Keller at the University Medical Center Schleswig-Holstein, Lübeck, Germany for their cooperation and their efforts in connection with the clinical evaluations employing *RealTimeFrame*.

Part of the project was funded by Olympus Winter & Ibe GmbH, Hamburg, Germany. We are grateful to them for supporting our research.

References

[1] Aach, T., Schiebel, U., Spekowius, G.: Digital Image Acquisition and Processing in Medical X-Ray Imaging. *Journal of Electronic Imaging*. Vol. **8** (1999), p. 7–22.

[2] Aach, T., Mayntz, C., Rongen, P., Schmitz, G., Stegehuis, H.: Spatiotemporal Multiscale Vessel Enhancement for Coronary Angiograms. *Medical Imaging 2002: SPIE*, Vol. **4684**, SPIE, p. 1010–1021, San Diego, USA.

[3] Gross, S., Stehle, T., Behrens, A., Aach, T.: RealTimeFrame – A real time image processing solution for medical environments. In: *41. DGBMT-Jahrestagung Biomedizinische Technik 2007*.

[4] Gamma, E., Helm, R., Johnson, R., Vlissides, J.: *Design Patterns – Elements of Reusable Object-Orientated Software*. Addison-Wesley, 2003, p. 4–6

[5] Lee E. A.: The problem with threads. *Computer*, Vol. **39** (2006), No. 5, p. 33–42

[6] Stehle, T., Behrens, A., Bolz, M., Aach, T.: Visual Enhancement of Facial Tissue in Endoscopy. *SPIE Medical Imaging 2008* (to appear).

Sebastian Gross
e-mail: sebastian.gross@lfb.rwth-aachen.de

Thomas Stehle
e-mail: thomas.stehle@lfb.rwth-aachen.de

Institute of Imaging and Computer Vision

RWTH Aachen University
Sommerfeldstraße 24
D-52074 Aachen, Germany

