Proceedings of the 5th WSEAS Int. Conf. on SIGNAL, SPEECH and IMAGE PROCESSING, Corfu, Greece, August 17-19, 2005 (pp185-190)

A Framework for Developing Image Processing Algorithms with Minimal Overhead

Fabian Wenzel and Rolf-Rainer Grigat
Vision Systems Department
Hamburg University of Technology

Hamburg, Germany

ABSTRACT
We introduce a framework for developing image processing algorithms. Its design is targeted at the needs of developers who should be able to focus on their specific tasks as much as possible instead of technical side-effects that arise in software development. After finding general requirements for being able to develop modular algorithms we outline the plug-in based architecture of our system and describe details that help developers and researchers implement new modules. Our cross-platform framework is written in C++ due to portability and performance. Even though it has been designed for developing image processing algorithms, the underlying techniques can be utilized in other signal processing fields as well.

KEY WORDS
Software Design, Image Processing Applications, Evaluation Platforms

1 Introduction

Researchers and developers in the field of signal processing often consider software aspects as a side-effect. On the other hand, as implementing methods plays a central role in their domain, architectures and designs are an important issue. We motivate our work by outlining some characteristics in the field of image processing.

Many solutions for an image processing problem share a common pool of methods. Hence, *reuse* plays an important role such that software libraries have evolved during the last years. A developer can benefit from a huge amount of existing code in order to solve a specific problem — on the other hand, existing software forces developers to adapt to a given environment that may, or may not, be compliant with their needs. This is a problem especially for languages like C or C++ which are not flexible regarding the interoperability of data types. On the other hand, both are still the languages of choice for the majority of real-time algorithms due to low level and portability.

Image acquisition and visualization is an important part in almost every application. Thus, other technical issues like the underlying operating system, GUI toolkits, or threading play a role. Dealing with them is mostly time-consuming and does not contribute to solving an image processing problem.

Modular software frameworks are able to deal with some of these problems. Blocks for acquisition and visualization may be assembled with others in order to build complex algorithms. On the other hand, they often restrict the scope of applicability even more if they do not provide ways to integrate custom data structures.

We present *VistaLab*, a cross-platform plug-in based software framework for image processing algorithms. Its design is different from existing frameworks for image processing as it provides a high degree of flexibility by minimizing the overhead for developing new modules. This way, existing software and custom data types can easily be integrated. Our work does not primarily focus on runtime aspects like parallel processing as it can be found in other publications in this field [1]; it is targeted at the development and integration of new algorithms.

This paper is organized as follows. First, we describe generic requirements of a software framework for image processing. In the following sections we provide solutions for each requirement: First we reintroduce the well-known "pipes & filters" architecture that can be found in many existing frameworks. Then, a generic filter interface is introduced. Section 4.1 demonstrates how such an interface can be implemented with almost no overhead. Subsequently, *VistaLab* is introduced, a framework in which those techniques have been successfully applied. Finally, some sample applications will be presented and an outlook of possible extensions to other fields of signal processing will be given.

2 Requirements for Developing Image Processing Software

The process of developing algorithms in image processing is characterized in this section. Each aspect leads to a requirement that frameworks for software development need to provide:

1. **Code reuse:** Techniques in the signal processing domain share a common pool of methods that may already be available.

⇒ **Modularity:** Modular software provides a high degree of reusability as modules, i. e. individual components that perform specific atomic tasks, are ready to be combined to build complex algorithms.

2. **Functional focus:** Researchers want to solve a specific problem in their domain. Secondary aspects like interaction with users via graphical user interfaces (GUIs) do not gain high priority.

⇒ **Automation:** In order to focus on the functional core of an algorithm, other technical aspects have to be automated as much as possible. Configurable parameters of a module are an example for items that ought to be changed via a GUI during runtime or stored in a project file. It is desirable to generate corresponding code automatically such that writing configuration dialogs does not fall into a developer's responsibility but can be done by the framework.

3. **Uncertain requirements:** In the beginning of a development phase, aspects like the target operating system or final toolkits to be used are not fixed and are likely to change in the future, either because of limitations of a current setup or because of other issues like modifications in a customer's specification.

⇒ **Platform independence:** In order not to be fixed to a specific software platform, tools have to be chosen that are independent of

- the underlying operating system
- language extensions
- external dependencies

In the case of image processing, external dependencies often cannot be avoided, but in this case they ought to be chosen in a platform independent fashion.

Proceedings of the 5th WSEAS Int. Conf. on SIGNAL, SPEECH and IMAGE PROCESSING, Corfu, Greece, August 17-19, 2005 (pp185-190)

In the next sections, it will be shown how those requirements can be met in practice. Each section proposes a solution for a single item mentioned above.

3 Pipes & Filters

An established architecture for processing signals like speech or images is the "pipes & filters" design [2]. Besides its simplicity, this design directly implies a high degree of modularity and serves well for signal processing applications. It is shown in figure 1 and is also used in software layers like Microsoft's *DirectShow* [3].

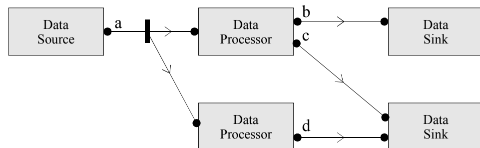

Figure 1. The "pipes & filters" - design for signal processing

Filters are atomic entities that receive data as input, process it and provide processed data at their outputs. Pipes serve as data channels and can be thought of as links between modules. A software framework based on this design enforces a standardized interface in order to be able to connect different modules. Moreover, modules can be implemented as plugins such that they can be easily distributed and changed even during runtime.

4 Generic Filter Interface

A close-up of an individual filter reveals components that are usually visible externally from a software point of view (see figure 2): Input- and output-pins. These two types of pins identify *dynamic information* that changes during runtime. An input pin must provide information about the data type which the module is able to process. Similarly, an output pin has to describe the data type it produces. Both pins must contain hooks for data transfer as well as identifiers. Hence, *meta-information* about incoming and outgoing data is needed.

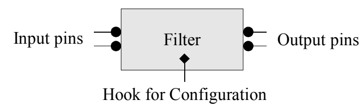

Figure 2. Externally visible parts of a filter

Parameters on the other hand define the state of the filter. They usually remain constant during a session. In contrast to input/output data, parameters are *static* and therefore cannot be modeled with the "pipes & filters" design as they have to be changed interactively. A common way is to provide an entry point for custom configuration. We propose another way to deal with parameters. In our design parameters are exposed in a similar way as pins, i. e. each configurable parameter has to provide an identifier as well as a hook for data transfer and the underlying data type. Hence, our design does not distinguish between dynamic and static data of a module, it requires both, input/output data and parameters as information that has to be visible externally in a similar way. Figure 3 illustrates this design.

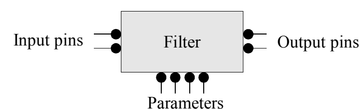

Figure 3. Exposing configurable parameters like I/O-data

Proceedings of the 5th WSEAS Int. Conf. on SIGNAL, SPEECH and IMAGE PROCESSING, Corfu, Greece, August 17-19, 2005 (pp185-190)

By providing meta-information about externally visible data, it is possible for the framework to transfer data into and out of a filter — while processing or when changing parameters. This way, automatic construction of GUI elements for configuration is possible. Hence, a developer can *focus on the functional part* of a module and leave technical tasks like building configuration dialogs to the framework.

4.1 Automation

In the previous section, an interface for image processing modules has been introduced that exposes parameters and input/output pins. Technically, exposing internal data to an external interface means that a developer must register data in some way by conforming to the appliction interface (API) of the framework. This is a problem as some requirements introduced in section 2 are not met: The API may change in future versions; also, a developer cannot focus on the functional core of a module. But as setting up a module's external interface cannot be avoided, corresponding code can be found in available signal processing frameworks, e. g. [4].

Our solution is to automate the process of exposing data as much as possible by generative programming [5]. Pins and parameters are identified in the source code at a high-level; the implementation for making them externally visible is done by a generator. We suggest a C/C++-based approach that makes use of the C preprocessor as generator. This way, source code does not contain C/C++-parts for exposing data to the framework at all. Moreover, the overhead for exposing data can be kept minimal. To demonstrate our technique, two examples are given: A configurable parameter of an integer type and an input pin for real-valued data:

```
// Declaring data to be exposed
INPUTPIN (double, m_inputData);
PARAMETER(int   , m_parameter);
```

```
// Adding Descriptions
CONFIGINPUTPIN
 (m_inputData, "Input Value" );
CONFIGPARAMETER
 (m_parameter, "My Parameter");
```

It can be seen that declaring externally visible data items can be done with two macros. The first one is located inside the filter's declaration, the second one in its implementational part. The description that is attached to each data with the second macro is also used as a run-time identifier or a tag in an XML project file, apart from e. g. displaying a label in a configuration dialog. No data structures of the framework have to be used such that changes in its interface do not require changes in a module's code. In addition, it is easy to use modules for different applications by simply changing macros. This might be important if software frameworks serve as an evaluation platform only. Data type information and references to memory are automatically generated by using C++ runtime information such that custom data structures can be passed through our framework. Our approach makes heavily use of the C/C++ programming language while still being standard conform, as opposed to similar techniques, e. g. [6]. The introduced ideas can be transfered to other languages that support generative techniques as well.

5 VistaLab: A Framework for Image Processing

In this section we present *VistaLab* [7], a framework for developing image processing algorithms in which the aspects introduced above have been successfully implemented. A platform independent toolkit [8] has been chosen such that it is possible to develop and evaluate algorithms on common operating systems. As no language extensions are used, VistaLab can be used in conjunction

Proceedings of the 5th WSEAS Int. Conf. on SIGNAL, SPEECH and IMAGE PROCESSING, Corfu, Greece, August 17-19, 2005 (pp185-190)

with any standard compliant C++-compiler. It is even suitable for cross-compilation such that the development system can differ from target systems. This way, the requirements of section 2 could be met.

Configurable parameters are exposed to the framework by the macros introduced in section 4.1. They are used by the framework for assembling configuration dialogs with matching GUI controls. Also, parameters are displayed in a graphical configuration tree and used when saving current setups to a project file.

In our system each module can expose custom hooks, in addition to static and dynamic data, by following the same approach described in this paper. These hooks are used as entry points for other modules for visualization, logging and unit-tests. A plugin-wizard helps generate a code skeleton such that newly created, "empty" modules can be integrated into the framework immediately.

Input/output data is realized via dynamic memory. This way, modules expose *references* to processed items at their output pin such that data transfer between modules can be done quickly without duplicating data. Also, the framework frees dynamic memory automatically. Finally, threading is possible by just declaring a module to be encapsulated into an own thread.

5.1 Sample applications

We introduce two setups to demonstrate our system. First, we show how feedback loops can be realized. Then we demonstrate how a stereo computer vision application can benefit from our flexible design.

5.1.1 Feedback loops

Our system calls modules sequentially such that circular dependencies like feedback loops cannot be modeled in a straightforward way. However, they can be simulated by using references as data-type between modules. Figure 4 illustrates how they can be realized with *VistaLab*. Module `A` gets feedback `z` from module `B` as `B` is able to access the reference to `z` provided by `A`.

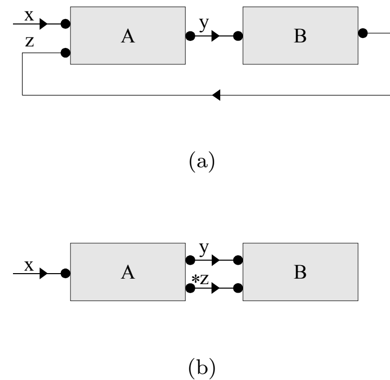

Figure 4. Feedback loops. (a): Schematic illustration. (b): Realization without circular connections. $*$ indicates a reference.

5.1.2 Stereo Computer Vision

A second example shows how our design can be utilized for stereo computer vision. In order to do some kind of 3D processing, two or more images of a scene are needed. They may originate from different cameras simultaneously or be captured by a single moving one.

Figure 5 demonstrates two possible module configurations. 5(a) shows a stereo setup with two cameras that run in parallel. As soon as both images have been captured, motion vectors between the first and second image are computed such that 3D calculations can follow. 5(b) illustrates that the same methods can be used in case of a single video stream: A delay line serves as an image buffer such that motion can be estimated between

Proceedings of the 5th WSEAS Int. Conf. on SIGNAL, SPEECH and IMAGE PROCESSING, Corfu, Greece, August 17-19, 2005 (pp185-190)

different snapshots of a scene.

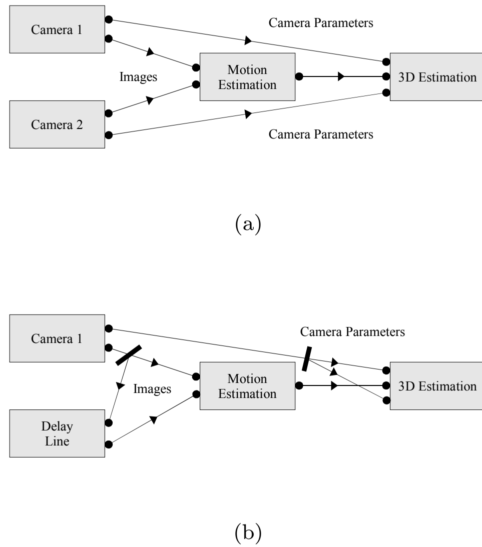

Figure 5. 3D Computer Vision Scenario: Two images of a scene can either be taken simultaneously by two different cameras (a), or they have to be taken at different times (b)

6 Future Extensions

Even though developing image processing methods is the main target of our work, it may be adapted other signal processing fields as well: The pipes & filters design is not limited to transferring images (see also section 5.1). In fact, due to the generic interface of our modules, it is possible to integrate other data like audio. The only limitation of our system is the fact that modules cannot operate at different frequencies. However, the ideas presented in this paper can be combined with others like [1], such that aspects like flow scheduling can be implemented with minimal overhead.

7 Conclusion

In this paper we discussed requirements and solutions for software frameworks in the field of image processing in order to develop algorithms with minimal overhead. Our approach is based on a generic interface for processing modules and generative programming. This design has two consequences: First developers are not required to gain knowledge of foreign domains like GUI programming. Secondly the source code for new methods can be kept in a future-proof way. We have shown how such an approach can be realized in C/C++.

References

[1] Alexandre R.J. François, Software Architecture for Computer Vision, in G.Medioni and S.B. Kang (Eds.), *Emerging Topics in Computer Vision*, (Prentice Hall, 2004) 585–654

[2] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, *Pattern-Oriented Sofware Archicture, Vol. 1 — A System of Patterns* (Wiley, 1996)

[3] Microsoft DirectX `http://www.microsoft.com/windows/directx`

[4] P.F. Whelan and D. Molloy *Machine Vision Algorithms in Java: Techniques and Implementation* (London: Springer, 2000)

[5] Krysztof Czarnecki and Ulrich W. Eisenecker, *Generative Programming* (Addison-Wesley, 2000)

[6] The Qt Toolkit `http://www.trolltech.com`

[7] The VistaLab Image Processing Framework `http://www.ti1.tu-harburg.de/vistalab`

[8] The wxWidgets Toolkit `http://www.wxwidgets.org`