A Testing Environment for Video-Based Multiview Computer Vision Research

Efstathios Stavrakis and Margrit Gelautz

Interactive Media Systems Group

Institute for Software Technology and Interactive Systems

Vienna University of Technology

Favoritenstrasse 9-11/188/2, A-1040 Vienna, Austria

e-mail: stathis@ims.tuwien.ac.at, gelautz@ims.tuwien.ac.at

Abstract

We describe the methodology, design considerations and practical implementation of an environment used in developing and testing video-based, multiview computer vision algorithms. The environment we have built is composed by a variety of heterogeneous hardware devices for acquisition and display, connected together into a coherent system by object-oriented software components. Much attention has been paid into the development of a Vision-Device Abstraction Layer (VDAL) that handles the operation of hardware image acquisition devices. The software components are divided into User Interface and Low-Level computer vision constituents. These are combined into more complex entities, Modules, that perform a complete computer vision process. Our environment is capable of providing output on disk, in real-time on common monitors, but can also take advantage of large stereo displays. It is designed with modularity, extensibility and reusability in mind. The main contribution of this paper is to describe the setup and efficient implementation of a coherent environment for video-based multiview applications and algorithms' development. We provide our experience of how a variety of different hardware and software has been glued together to form the basis of our computer vision processing environment.

1 Introduction

Computer vision video-based research and applications lack a common framework definition and guidelines due to the diversity of algorithms and requirements, as well as the fast emerging advances in the field. The advent of more powerful hardware in conjunction with the development of software tools that take advantage of its capabilities have enabled research and applications in computer vision to be increased exponentially. However, we believe that relevant literature and insight on setting up, designing and implementing unified systems needs to be enriched.

A multitude of image acquisition devices exists in the market. These range from cheap webcams to expensive high-speed cameras and have different functionality. A common limitation is mixing heterogeneous acquisition devices. Devices may support certain features, such as specific resolutions, frame rates, etc. This poses no problem for single view processing, but has undesired effects in multiview processing. In most cases introducing functionality that does not exist on hardware is not possible. However, via proper software design alternative functionality may be provided or operation between homogeneous devices may be enforced when this is necessary. Many researchers prefer to use identical acquisition devices on their research platforms in order to alleviate themselves from the overhead of dealing with different hardware and its management. As pointed out above, the field is evolving at such a rate, that newly introduced devices produce better results and give solutions to computational bottlenecks. Having the possibility of using various different acquisition devices is always an advantage. To achieve this, an abstract way of handling them is desired so that heterogeneous devices may be incorporated within a system.

Apart from the hardware of a system, its software plays a central role. While there is an enormous amount of different software tools[5] that perform computer vision tasks, only a few can be used out-of-the-box and interfacing such systems with image acquisition devices may impose significant effort. Nevertheless, systems such as Khoros and Matlab are preferred, for their algorithms collection and visual programming tools, but sometimes they are inadequate when performance is critical and manual intervention of the programmer is needed right on the lower-level core of the system, which is not available. Contrary to those systems, there are also open source software libraries that may be used. These are quite detached and isolated from user interfaces, mainly for efficiency and portability. Sometimes it is not clear to the developer how to bring to life his envisioned application using these computer vision libraries together with Graphical User Interface (GUI) libraries. Also the selection of the appropriate software tools that can fulfill a research task is at large a dependable process to the previous experience of the researcher.

The above pieces of software usually take advantage of processor special instructions to improve processing speed. On the other hand, there is a trend towards hardware accelerated implementations on the Graphics Processing Unit (GPU) found in most commodity hardware graphics cards. Using the GPU frees the main processor from expensive tasks and it can be used to perform other operations such as disk access, user interfacing and processing in parallel to the GPU for even more optimized performance, especially in multiview temporal processing.

User interface components can be developed with most existing GUI libraries available, which vary in complexity, but also in functionality. Apart from the native Window Manager system libraries (i.e. MFC for Windows, or Xlib for X11), one may use any other object-oriented library for GUI component development. Selecting a library to use in building components for an application is mostly a matter of preference, but is also heavily depending on the selection of the working platform.

Reviewing the above considerations on software design and use of off-the-shelf libraries to implement a working environment, one should make a compromise between installation subtleties, management effort, development complexity and environment usability, reusability,

performance and extensibility. In other words, the implementation of highly complex and elaborate environments for the purpose of research due to time constraints is usually not an option for many computer vision researchers, hence balance between complexity and time has to be preserved.

Common types of output of such systems are most commonly either 2D images or 3D scenes. For their visualisation, conventional monitors are used as output devices or files are written directly on disk for later use. However there are cases where stereo or immersive displays are more appropriate to perceive the results.

In section (2) we describe the methodology and theoretical basis upon which we have founded the construction of our environment. In section (3) we describe the System Architecture together with the setup and implementation considerations of our own environment. Finally in section (4) we summarize and propose directions for future work.

2 Methodology

The most important part in video-based research, but also applications, is to identify the desired outcome of the system. This outcome has to be thought of as a general entity or broader definition of a goal and not as a practical quantity, such as an image or a video stream. In practice this goal may be the development of a novel algorithm for 3D reconstruction or a real-life surveillance application. A careful decision and analysis on this initial step will allow proper design and implementation of the individual tasks required in fulfilling this goal.

Naturally, after the identification and analysis of the system's goal, it is vital to choose the type of system input sources. The selection of input sources is crucial for such an environment, because it can directly affect the possibilities and capabilities of the system. This successively also has an impact on the quality of the achieved results. Some algorithms may require special types of image stream acquisition devices, such as stereo cameras, etc.

Apart from a goal and input sources, it is necessary to also develop software components that will enable the use of these acquisition devices, will perform operations on the acquired data and also provide a suitable format of the processing results for communicating them effectively to the user.

Handling and using hardware acquisition devices, especially when they have special features, is always a necessary task. In addition, interfacing algorithms with new devices can be time consuming, but the definition of a common layer of functionality can minimize the required effort.

An aspect that is sometimes overlooked in research is user interfacing and its implications to the workflow. Even though many proficient researchers prefer to have direct access to the lowest level and may develop and use a command line tool, rather than a complete environment, sometimes the complexity and multiple-step nature of todays algorithms is prohibiting for such an approach. Since research involves the inspection, testing and evaluation of multiple algorithms of the same domain, switching between algorithms that perform only one step of the complete algorithm provides insight on the suitability of a particular technique. For example, on a 3D reconstruction algorithm, it is not uncommon to inspect various stereo matching

algorithms before deciding which one presents the best possible solution.

To tackle the problems related to hardware devices and user interfacing in our approach, we have created an environment that such workload is reduced by providing high-level access to hardware devices and a set of easy to use reusable object-oriented GUI components. In addition, high-level results of multiview temporal algorithms are better visualised and evaluated via video streams, so comparison and evaluation may require viewing them in parallel via custom elegant user interfaces that are not readily available in the GUI libraries. We consider communication of multiview video streams to the user an important aspect within our environment and for this reason we separately define and handle it as an individual component.

The requirements of a multiview video-based computer vision environment as identified above are summarised below:

1. Environment Goals: the multiview video-based research we would like to conduct within our environment.

2. Environment Inputs: hardware devices for acquisition, their installation and operation, as well as file formats for disk based stream input, if appropriate.

3. Environment Bridging Components: middleware software for user interfacing, data acquisition, processing, testing, evaluation and comparison. The combination of multiple Bridging Components into unified processing units of the environment are defined as Modules.

4. Environment Outputs: display devices used for visualisation of results, but also file formats used to store outputs on disk.

In the next section, steps (2) through (4) are described in more detail in respect to our multiview video-based research goal.

3 System Architecture

The methodology described in the previous section allows one to selectively setup or develop parts of the environment. As stated earlier, the environment is build with reusability, modularity and expandability in mind, which successively allows one to use the same environment for several research tasks by adding new parts and performing minor modifications in software. In our System Architecture, as presented in more detail on Fig. 1, we take an object-oriented multi-threaded approach in developing our software. We use the C++[7] programming language for its speed and widespread adoption, but any other object-oriented language could be used. The goals of our environment include research in the areas of 3D reconstruction, stereo viewing and tracking from multiple views. We will describe our environment parts in a generic form, but we will also give examples and experience notes linked to our current research projects.

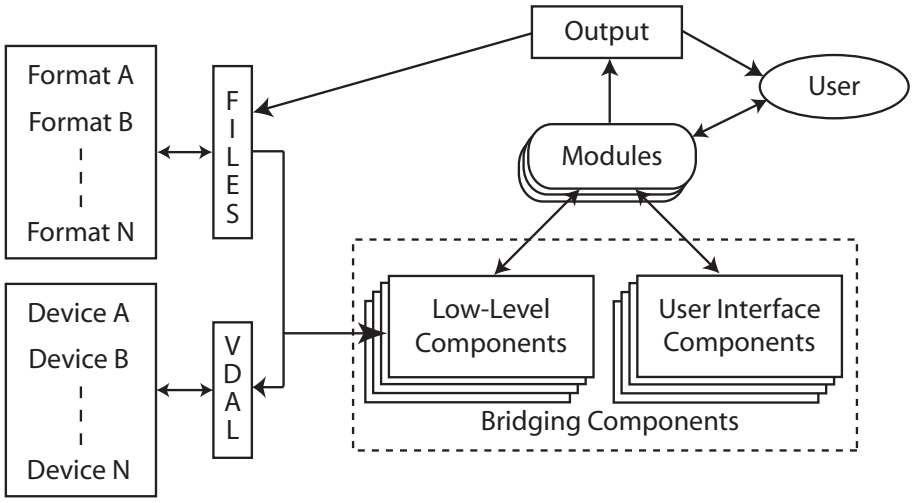


Figure 1: System Architecture

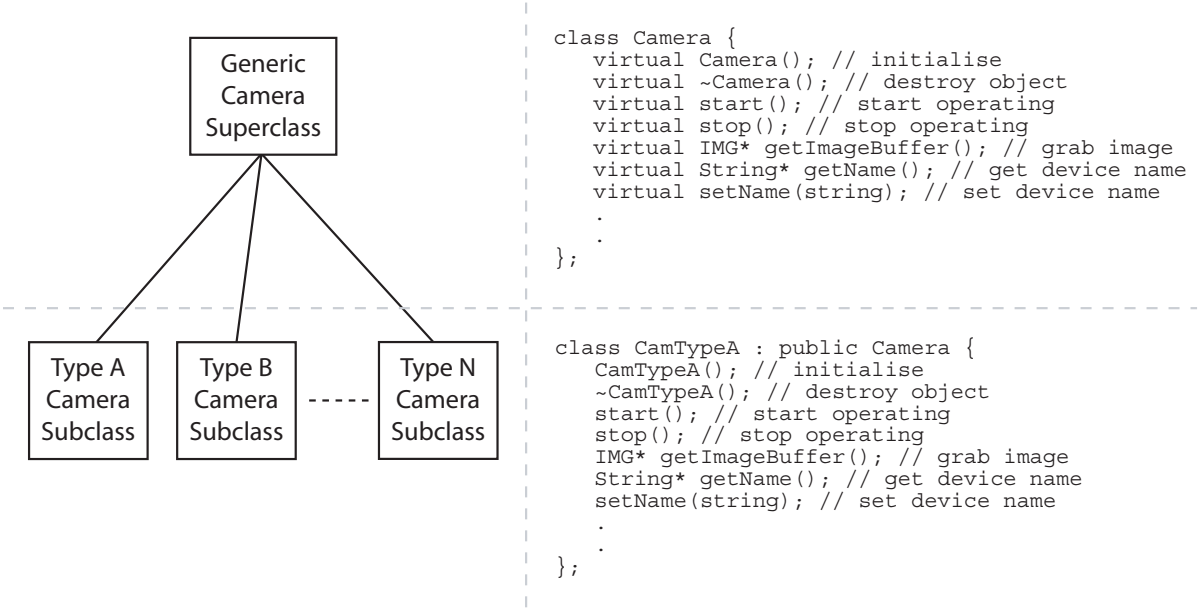


Figure 2: Vision Device Abstraction Layer Implementation

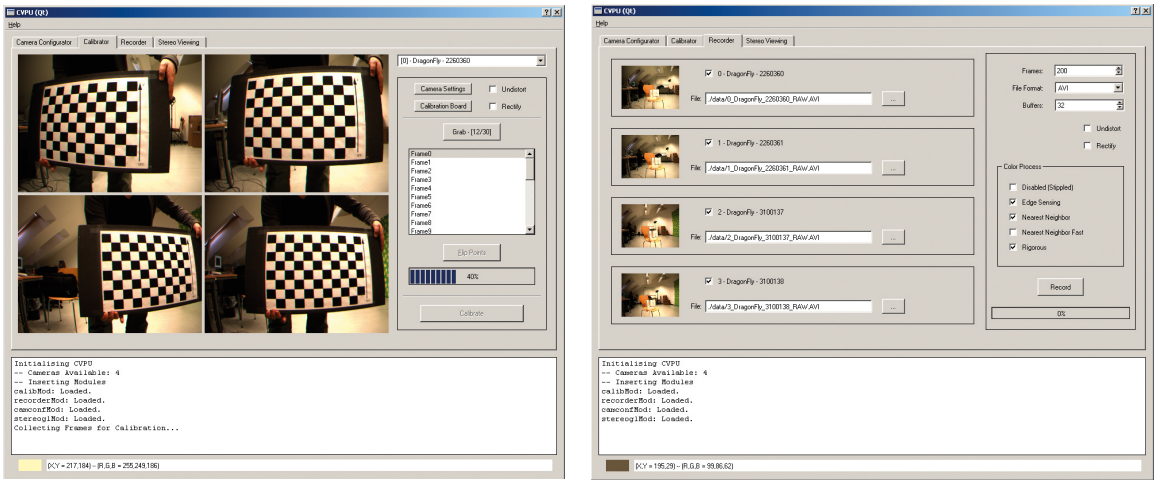
3.1 Inputs

Inputs can be acquired via hardware devices and their respective software APIs. To relieve ourselves from the task of reimplementing functionality from program to program and one device to another, we have developed an Application Programming Interface (API) in the form of a system library. We call this API, 'Vision-Device Abstraction Layer' (VDAL). Its function is to provide a common programming interface mechanism to hardware image acquisition devices. The VDAL is built on top of the lower-level proprietary APIs provided by the hardware vendors of particular devices. This abstraction layer encapsulates the functionality of the hardware devices, so that new types of hardware can be added to the framework with minimal effort. In software engineering terms, each device has an associated object-oriented class that is a subclass of a superclass that describes an acquisition device in a generic form, as shown in Fig. 2. This generic form exposes access points such as retrieving an image buffer, setting or retrieving common camera parameters, i.e. buffer width, height, device shutter speed, etc. Of course there are features on some devices that are not present on others, usually specialised. We give direct access points to the developer for those and provide alternative functionality for those that lack features, whenever possible, or appropriate feedback that can be parsed by our Bridging Components. This allows the developer to perform more elaborate specialised work if required. As an example, we have a Point Grey Bumblebee stereo camera which provides disparity maps at interactive frame rates, whereas our other devices do not have such a feature. We allow the developer to request the disparity map whenever required instead of calculating it, as he would do when dealing with other cameras.

In our VDAL we handle multiple types of video devices. In particular we use a High-Speed RedLake[2] MotionPro10000 camera, a number of Point Grey[6] DragonFly, FireFly and BumbleBee cameras, and also Philips[4] ToUcam Pro webcams. This allows us to carry out research in our environment by enabling us to use both specialized and commodity end-user hardware acquisition devices selectively. In practice, the VDAL generates a list of available supported hardware acquisition devices upon initialisation. The VDAL is then internally responsible of maintaining control of them, providing to the developer the functionality that is exposed via the API. It should be noted that the VDAL is an API that is used by other environment components in order to provide high-level interfacing to the user, as described in the next section.

3.2 Bridging Components

The glue of our environment is the middleware software that puts together user-interfacing, hardware device accessibility and output. We have split the Bridging Components in two sub-categories that are tightly related; namely, the User-Interface Components and the Low-Level Components. Modules of the system are created by the combination of a collection of Bridging Components. Most of the Bridging Components are separate threaded objects that give our environment multithreaded capabilities, that can take advantage of multiprocessor architectures.



(a) Calibration Module (b) Recorder Module

Figure 3: Screenshot of the actual software. (a) Example Module used to calibrate multiple stereo pairs, (b) another Module that is used to record from multiple cameras on disk.

3.2.1 Low-Level Components

Low-Level Components include programmed algorithms that perform common low-level operations in the computer vision domain, such as edge detection, image resampling and transforms. An advantage of these Low-Level Components is that the developer has the choice of implementing low-level algorithms from scratch or alternatively he can create wrappers around off-the-shelf computer vision software libraries. In our particular implementation we mainly use OpenCV[3], but migration and mixture with other libraries is supported by the component-based modular design of our system architecture. Since performance is one of the most important factors in image processing and analysis, some of our already implemented Low-Level Components are taking advantage of graphics hardware to utilise the processing power of the Graphics Processing Unit (GPU). We do this by using OpenGL and the OpenGL_ARB_Imaging extention found in most modern graphics cards. Particulars for developing hardware accelerated Low-Level Components can be found in [8].

3.2.2 User Interface Components

The User-Interface group of software components is a collection of GUI widgets. These widgets provide access to the VDAL functionality, they include enhanced widgets that deliver output on display devices and also a variety of common reusable template widgets, such as drop-down boxes that group together similar Low-Level Components for easy access and interchange by the user on the fly. We would also like to point out that user interfacing in computer vision is sometimes sacrificed for efficiency or simply overlooked, however, we found that accessing algorithms via GUIs speeded up our progress and increased our work capacity.

3.2.3 Modules

Nowadays most computer vision algorithms are usually composed by a smaller set of tunable operations. Combining and swapping individual parts of such multifaceted algorithms can be a cumbersome and time consuming task if an object-oriented programming paradigm is not followed. We prefer to build the low-level parts and also respective user interfaces as reusable objects. By combining several Bridging Components we build Modules that are complete software supersets of the previously described parts of our system and these are exposed to the end-user. Such an approach relieves the user from typing command line parameters and also presents a solution in interchanging between available algorithms and tuning their parameters. The usefulness of this approach, especially in research, is the flexibility of being able to selectively or in parallel generate and visualise output of algorithms in the same domain, for comparison and evaluation. Furthermore, related Modules are designed to interoperate so that information and data can be propagated from one another whenever desired. An example of our system modules is shown in Fig. 3, we have built a Calibration Module (see Fig. 3a), that is used to calibrate stereo pairs. The calibration data are then broadcasted within the unified environment of our Modules, which allows us to correct our images for lens distortions and to rectify them from within other Modules of the system. The same principle of Module interoperation can be used with other type of algorithms if required.

Figure 4: Stereo Viewing on a Baron BARCO display.

3.3 Outputs

Output of the system is desired to be stored on disk or shown on screen in realtime. Both of these tasks are handled by the Bridging Components. We developed Low-Level Components that can use multiple file formats for storing and retrieving the video streams to and from disk. In realtime processing mode we use widgets to draw the image buffers. At first sight this appears common practice and for simplicity it would make sense to disguise it under the structure of the Bridging Components. Nevertheless, we found that it is much more intuitive and easier to manage Outputs if treated separately within the environment's architecture. The unique set of custom widgets we developed so far for realtime visualisation of the results of algorithms provide utilities through the user interface that are independent of displayed output. Such utilities include verbal and visual communication of color information at user defined image coordinates, gradient estimation, but also more elaborate operations such as scanline alignment checking between views of different rectified images. This set of utilities that are attached directly to the widgets, can be expanded to include other operations one may commonly use within the scope of his research interests. Another reason for treating the output functionality separately is that sometimes visualisation of results from a variety of algorithms is not meaningful without special display hardware. The motivation came directly from our on-going research in video-based stereo viewing. To visualise the stereo pairs of image sequences we developed a more specialised type of output widget that takes advantage of OpenGL's stereo capabilities, we rendered the output on a high resolution Baron BARCO display[1] and viewed it using active stereo shutter glasses, as shown in Fig. 4. By separating the outputs of our system, we managed to develop a small set of reusable elements that are transparently handling the output of our Modules onto disk and the display devices we currently use. Essentially this is another layer of abstraction that handles the output functionality in our environment, similar to the VDAL described for our input facilities.

4 Conclusions and Future Work

We have presented the methodology, system architecture and implementation directions for setting up and using a coherent testing environment for multiview video-based research. Our environment is based on the principles of object-oriented software engineering paradigm and takes a multithreaded stance. Particular attention has been paid into the implementation of a Vision-Device Abstraction Layer that manages hardware image acquisition devices. We built middleware that is aparted from both low-level algorithms and also user interface reusable and modular elements. When these components are composed together, they provide coherent Modules for multistep computer vision research tasks.

We plan to expand our system by developing more Bridging Components and integrating other types of acquisition devices into the VDAL. Currently the environment is implemented for Microsoft Windows platforms, however we have intentionally chosen the underlying software libraries to provide cross-platform support, so that other operating systems can be supported in the future. Finally, we are planning to develop a Scene Graph-based processing engine and accompanying GUI tool that will enable the researcher to combine Bridging Com-

ponents into Modules via high-level interfacing, instead of programming, providing visual programming functionality within our environment.

5 Acknowledgement

This work was supported by the Austrian Science Fund (FWF) under project P15663.

References

[1] Barco. Baron. URL: http://www.barco.com/VirtualReality/en/products/product.asp?element=312.

[2] Roper Scientific B.V. Redlake. URL: http://www.redlakeeurope.com.

[3] Intel Corporation. Open source computer vision library. URL: http://www.intel.com/research/mrl/research/opencv/.

[4] Philips Electronics. Philips webcams. URL: http://www.pc-cameras.philips.com.

[5] Computer Vision Homepage. Image processing toolkits. URL: http://www-cgi.cs.cmu.edu/afs/cs/project/cil/ftp/html/v-source.html.

[6] Point Grey Research Inc. Hardware image acquisition devices. URL: http://www.ptgrey.com.

[7] Bjarne Stroustrup. *The C++ Programming Language - Special Edition*. Addison-Wesley, February 2000.

[8] Mason Woo, Jackie Neider, Tom Davis, and Dave Shreiner. *OpenGL Programming Guide - Third edition*. Addison-Wesley, 1999.