echo & narziss

submitted by

Daniel Fischer

in partial fulfillment
of the requirements
for the degree of

Diplom-Mediengestalter

to Bauhaus-Universität Weimar,
Fakultät Medien
Weimar, May 13, 2004

supervised by

Hochschuldozent Ralf Homann

Prof. Dr. rer. nat. Bernd Fröhlich

»When Narcissus died the pool of his pleasure changed from a cup of sweet waters into a cup of salt tears, and the Oreads came weeping through the woodland that they might sing to the pool and give it comfort.

And when they saw the pool had changed from a cup of sweet waters into a cup of salt tears, they loosened the green tresses of their hair and cried to the pool and said, ›We do not wonder that you should mourn in this manner for Narcissus, so beautiful was he.‹

›But was Narcissus beautiful?‹ said the pool.

›Who should know better than you?‹ answered the Oreads. ›Us did he ever pass by, but you he sought for, and would lie on your banks and look down at you, and in the mirror of your waters he would mirror his own beauty.‹

And the pool answered, ›But I loved Narcissus because, as he lay on my banks and looked down at me, in the mirror of his eyes I saw ever my own beauty mirrored.‹ «

Oscar Wilde, ›The Disciple‹

Table of Contents

1. Introduction

In some ways, this is a project about Freedom. Freedom, and what you make of it, has become a focal point of my studies of Media Arts and Design at Bauhaus-University Weimar. I was lucky enough to find an environment where I could freely seek for what it is that drives my work. I have not found it, and hope I never will. Yet, I have learned how to move in such a free space, how to use it for my own good and the good of others. I have learned how to use my talents, and how to extend my abilities, within a liberal, encouraging and changing environment.

At the same time, this project is about constraints, about external and internal forces that impose limits upon that freedom. Time, for one part, seems to be the most forceful constraint. There's never enough of it, there's always so much that could have been done given enough time.

The limits of the technology available to us is another forceful constraint limiting our work. Contrary to ever progressing time, though, out tools can be changed. Because we made them ourselves, and if we extend our technology we extend the possibilities of our work.

Within this contrast of freedom and constraints I like to place my work. For this is the same antagonism as the everlasting battle of artistic expression and technical possibility, of vision and realization.

Consequently, most of my work deals with the development of new *tools*. With the advent of affordable computing power, the possibilities seem endless, no matter which way you look or which goals you want to achieve. These possibilities, though, are of a purely theoretical nature. What we can do with today's software seems to have no relation to what we could do theoretically. This is why the practice of my work has shifted more and more to programming, even while the goals I want to achieve remain of a dedicatedly artistic[1] nature.

1 The proper term here would be *gestalterisch*, for which there is no direct English translation.

2. Background

Every work stands within its own context, and without at least shallow knowledge of that context, it cannot be understood. This is especially true for a project that, like echo & narziss, is primarily defined by means of its extension in time, as it tries to progress towards general goals that cannot be reached within a single concentrated effort. While I have set out reachable goals for the project itself, if only to enable my supervisors to judge the progress made, the long-term goals remained in focus throughout the working period. These goals (and thereby this project) are to be seen in context of some global developments affecting our use of computing technology in general, in relation to the local background posed by my studies of Media Arts and Design in Weimar, and personally in line of my personal abilities, working style and history.

2.1. Global Context

Even as computers in the form of gray boxes and laptops have largely pervaded our daily work and live, they remain mysteries to most of us. We handle them mostly through an established set of interfaces consisting of Keyboard, Mouse, Windows, Menus and so-called ›Dialogs‹. While the commercial success and widespread use of this interface might support the initial vision that this is an interface everybody can understand and use, I believe it's not the end of the story. The possibilities of a ›normal user‹ remain too limited to the constraints imposed by developers, the style of interaction too predefined by the application's design, as to say that computers are used as the generic all-purpose information processing tools they could be.

Today, we have to submit to the logic of the computer, or at least to the thinking of the application developers. Instead, I envision the computer submitting to our logic, adapting to our own personal working style. This would include a general *invisibility* of the interface whenever we don't need it, coupled with *availability* when we do. It would in-

clude us being able to *teach* the computer new stuff, so that the computer becomes the one executing repetitive tasks, and not us. And, it would include the computer being *aware* of its real-world environment, as to note when an interruption can be appropriate, to judge our current working condition and to respond to various other external situations.

2.1.1. Computers, Video and Dance

Video technology had its influence on contemporary dance since its general availability in the sixties. With the advent of affordable hardware capable of processing video in real-time, the use of video technologies in dance has undergone a major shift, away from using canned video sequences or mere one-to-one live transmission, towards using the potential of computers in both analysis of performer motion and live generation (synthesis) and/or manipulation of video material visible to the audience.

The area of contemporary dance provides rich grounds for experimenting with computer vision and real-time video synthesis technologies, and the aesthetics enabled by those. The intriguing experience of watching a person move skillfully can be extended and amplified using near-real-time displays of visual analysis, and coupling of the performer's motion to the production of sound and accompanying imagery. Within this area of analysis/synthesis, one can find large potentials for artistic expression, as I try to prove with this work.

As of lately, many dance companies[2] have begun working with computer scientists and exploiting the possibilities that this combination can offer. With increasing spread of and accessibility to these technologies[3], I expect to find more and more such experiments and hopefully also productions that exceed the experimental stage, and harness the possibilities of technology to both extend the realm of artistic expression and amplify the audience's experience in watching dance.

2.1.2. Computer Vision

The possibilities of applying computer vision algorithms in arts, and general human-computer inter-

2 For example, [TroikaRanch] and [Palindrome].

3 See [Jitter], [EyeCon] and my own Warsaw and Pakt (4).

action, are vast. With them, computers can be made more aware of their environment - even simple information like approximately how many people are around the room (any at all?) could make a huge difference in user experience when integrated with the overall interface. Probably more important than awareness about the environment, cameras can be used as input devices, allowing the computer to know where the user is looking and/or pointing at. Gesture recognition, shape recognition and shape tracking can prove to be valid technological bases for new interaction methods, as they don't require the user to use any kind of special tools but his own body. Combined with other emerging technologies like natural language understanding, such methods have the potential to thoroughly change our current modes of interaction with machines.

It has been a project goal to evaluate some of the currently available computer vision tools for their applicability within a more playful context - that of a contemporary dance performance. Within this context, the performer becomes the primary ›user‹ within a reactive space. The potential of such a reactive space can be estimated by the successes of for example the experimental performances of David Rockeby with the ›Very Nervous System‹[4] or Sony's ›EyeToy‹-based PlayStation games[5].

The ›EyeToy‹ case exemplifies another factor that plays an important role within this work: the availability of increasingly powerful commodity hardware that has by now reached a stage where complex video processing and analysis becomes possible with a standard desktop workstation, with even enough processing power left to do other things. In fact we can say that the speed of our hardware doesn't pose the major limits any more. What remains to be solved for computer vision technology to enter the mainstream of user interface design are mainly problems of accessibility and integration.

4 [Rockeby]

5 [Sony]

2.1.3. Open-Source/Free Software

Part of the accessibility problem is addressed by the use of Free/Libre/Open-Source Software within this project and within my work in general. While an Open-Source license does of course not guarantee that a software is accessible in a sense that it is understandable and user-friendly, it does guarantee access to and availability of the software's source code and enables further improvement and modification of the software[6].

As a large set of computer vision algorithms is freely available, their integration into an open-source media processing environment seems a logical step forward. An effort has been taken to do so, and indeed a significant number of algorithms available from the OpenCV[7] library have been integrated into the Warsaw framework (see 4.4.6) as part of the project.

6 For a detailed account of the benefits of Open-Source, consult [Raymondoo].

7 [OpenCV]

2.2. Local Influences

In retrospect, the most important change triggered by my studies is a shift in my conception of art. Shortly after I came to Weimar to study, I had defined my alleged subject, which I then translated as ›Media Design‹ (see below), as follows:

1. giving *form* to *content* of some *medium*; includes layout, screen design, web design, video and audio production.
2. integrated (›holistic‹) engineering of some information service; this includes a fair amount of *interface design* with lots of thoughts spent on *efficiency*, *usability* and *accessibility* as well as on the *structure* of the underlying content.

Near the half of my studies, I had extended that definition with an important third part:

3. the *development* of *new media*; the invention of new forms of transmitting information and new forms of interactivity. As the computer integrates more and more traditional forms of broadcast or general communication, truly unprecedented ways of getting the content from the

sender to the receiver can evolve.

By now, near the end of this personal era, I would like to extend the definition by another perspective, but that task turns out to be increasingly difficult. My shift in perception of the profession can probably best be observed with my adoption of a different English translation for the official term ›Mediengestaltung‹: *Media Arts and Design*.

I will not extend this argument into a general discussion about ›What is Art?‹ Instead, I will just mention that I regard the area of artistic work as *conducting research* for the more applied area of design. One of the primary differences between Art and Design, then, could be that any work classified as artwork is more free from commercial constraints and from any strict a-priori definitions of the work's outcome. It should be clear that this doesn't make either of the perspectives any less relevant than the other. Personally, I strive to achieve a sound balance between the two, by trying to finance free *art* projects from applied *design/technology* work in the industry.

2.3. Personal Background

I regard my ›core‹ competences to be a multitude. During the five years of my studies, I had the chance to play with various media technologies, to develop concepts for very different kinds of projects (and implement them), and to cooperate with a wealth of neighboring professions.

I reckon as one of my central competences the ability to mediate between artistic/design-oriented and technical professions. While programming a lot myself, I do not feel systems architecture to be one of my major goals. Likewise, while creating visual designs myself, I do not hold myself to be an artist or designer in any conventional meaning of the word.

My skill, then, is to know enough about both (or all involved) perspectives to develop concepts spanning the arts/technology gap, to be able to understand the processes and designs involved, to judge and direct the work of others in their special areas, to ›help out‹ on various issues myself, and finally to keep an overview throughout the project. In short, I don't care as much for a *deep* than a *broad* understanding of things.

Still, I have of course developed some more concrete skills. I code enough to regard programming as my primary tool for expressing and implementing my ideas, and to be able to implement most of my concepts myself at least in a prototypical or ›sketched‹ fashion (in C/C++ and Java). Meanwhile, I collected an amount of experience in event organization and technical implementation (lights, sound, cabling, crowd management). Also, I feel that I know a lot about visual design, issues of form, color and typography. I run a set of Internet servers in varying contexts, use Linux as my primary operating system since about two years, and master most of the web languages (HTML/CSS, XML, PHP, SQL, XSLT) fluently.

During the major part of my studies (during the last ca. 3 years) I have experimented a lot with optical tracking technology and motion image synthesis in artistic and playful contexts like interactive installations and dance performances.

2.4. This Paper

This paper documents primarily the technical results of the process that finally leads to the presentation of a dance performance involving most of the developed software. As the performance itself is yet to take place as this document is being written, in some areas it necessarily remains vague. The documentation will be complemented with a video recording of the performance, which should, as I hope, clear up most remaining questions.

It should also be noted that this paper does not constitute a strictly scientific work. Many experiments that ultimately led to the results presented here are left out, as are many details of the implementation or algorithms used. As the work mostly integrates techniques already presented elsewhere, including all the implementation details would unnecessarily bloat this document. The contribution of my work is to be found in the bridging between the theoretical possibilities of computer vision in human-computer interaction and the concrete problems of conducting a real-life performance that realizes some of these possibilities.

3. Goals

As the goals within this project have not substantially changed over the project period, I am reproducing here (from the following paragraph up to section 3.2.4) the list of goals that I had mentioned in an initial project outline, modified only in terms of internal references and some nomenclature:

3.1. General Goals

The most important goals of the project can be summed up as follows:

- to evaluate the use of video analysis/synthesis software in the realm of contemporary dance.
- to verify utility of the Pakt framework in a performance context.
- to (further) explore the aesthetic properties and possibilities of direct graphics programming.
- to broaden my knowledge about these subjects.

3.2. Personal Expectations

I regard my final work as the possibly last chance to work freely and exclusively on a certain set of own ideas. I want to act out and use that freedom intensively, and ›finally do what I so long wanted to be doing‹, such as:

3.2.1. Make a concentrated effort

While I have gained some experiences in cooperating with contemporary dance performers, they all stem from highly experimental, ad-hoc projects. I yearn for the possibility to work with performers over a slightly longer period of time than just a few days. The current time frame suggests two blocks of rehearsal with both performer and audio artists (each lasting at least a weekend) over a quarter-year period[8].

3.2.2. Have a controlled situation

Most of my past installations using optical tracking took place in relatively unforeseeable, changing conditions. The chance to apply the same (or

8 Obviously, the *original* time frame suggested that.

extended) techniques in a controlled situation (especially in regards to lighting conditions) seems intriguing. A stage setting can easily provide a high level of control.

3.2.3. Advance the technology

I want to take the specific demands of the performance as a chance to advance the Pakt framework (see 4.3), and extend and mature my core set of video processing tools. Additionally, some more specific solutions will have to be developed, extending my knowledge about computer vision and motion image synthesis.

3.2.4. Flabbergast the audience

An important aim is to create an *intriguing* aesthetic experience, and to *touch the hearts* of people. This has little to do with the actual technology implemented, but very much with how it is being applied[9].

3.3. My current view on these goals

Looking back at these definitions of goals set out at the beginning of the project, I am relatively content with the outcome as it is presented in this documentation. While some parts of the personal expectations could not be fulfilled (most notably, no rehearsal including the dancer has been conducted yet, making it impossible to incorporate her feedback directly within this project), I think I have made some significant steps forward in the direction I set out for.

While the initial goals have been deliberately formulated in a scalable way to accommodate for unexpected obstacles on the way (of which there were quite many), I like to say that I have reached or exceeded the most important of them, especially in the area of technological advancement of the Pakt framework and the set of processing tools.

But please draw your own picture.

9 To detail this goal: The technology should *extend and amplify* the audience's experience of watching a performer dance, as to *enrich* the audience within their personal universes.

4. Technology

4.1. GStreamer

GStreamer is an open-source ([LGPL]) project that implements a graph-oriented media processing framework for the GNU/Linux architecture. It is based on the widespread GObject/glib/GTK libraries, and like those, is implemented in plain C following the object-oriented programming paradigm. GStreamer in its current incarnation (version 0.8) is relatively stable, enjoys a very active development community, and is in the process of being ported to both Mac OS X and Windows. It has been chosen as the default media infrastructure for the GNOME desktop environment, and is being considered for KDE as well.

Graph-oriented media processing models follow the idea that complex processing tasks can be achieved by combining simple, reusable elements into ›processing pipelines‹. A single component has a defined interface, consisting of a number of inputs and outputs, media formats it can handle, and parameters that control the encapsulated algorithm.

Such a model allows for rapid prototyping and experimentation with arbitrarily complex setups. Similar models have already been implemented (or are in the process of being implemented) by such prominent representatives of the software industry as Microsoft [DirectX], SGI [libdm], Be [MediaKit] and the X Consortium [MAS]. With audio synthesis, the approach seems a natural fit to conventional hardware-based setups, so one can find it implemented by a few major audio applications (eg, [Reaktor]). Additionally, the media arts community displays widespread use of software exhibiting very similar properties for ›real-time‹ interactive installations and performances: Cycling74's MAX/MSP and [Jitter], [Isadora] and [Keyworks] can serve as examples here. Finally, within the video effects arena one can find various examples of the model, for example [Combustion] and [Houdini].

Image 1: Graph of an example GStreamer pipeline

4.2. Overview

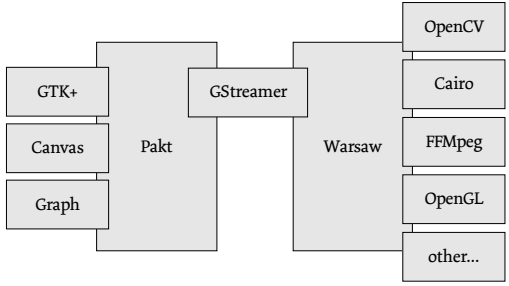

Image 2: Pakt and Warsaw integrate other libraries with GStreamer from two different sides

I have developed two integrative software systems that both interface to the graph-based media processing platform GStreamer (see 4.1), but from two sides: *Pakt* integrates GStreamer,GTK+ and other libraries into a high-level XML-based model daemon that allows

1. access for various network clients to setup, run and manipulate a server-side GStreamer pipeline (4.3.1), and
2. construction of GUI clients for remote control of such a server-side pipeline (4.3.3).

Warsaw, on the other end, integrates various third-party signal and image processing libraries for use as GStreamer elements. A wealth of such libraries is available as Free Software, and Warsaw enables arbitrary combination of processing algorithms from different sources by wrapping their functionality into the well-defined framework of GStreamer components.

4.3. The ›Pakt‹ framework

I have started development of the ›Pakt‹ framework in June 2003[10], originally to allow network access to setup and manipulate server-side GStreamer pipelines. Since then, Pakt has evolved to a generic XML serialization/deserialization and network access library for the GObject model. Various libraries based on that model have been integrated in an experimental Pakt module called ›Glasnost‹, mainly to enable construction of prototypical GUI-driven control clients (see 4.3.3).

I will not describe the functionality of Pakt in great detail here[11], but only give a short example-based introduction to the basic idea.

10 Following the strict definition of a graduation project by its extents in time, this excludes Pakt from being a direct part of my diploma project. It should be noted, however, that Pakt forms the basis of the tools developed in course of this project, and significant modifications and improvements to Pakt have also been made during the diploma period.

11 See the Pakt Accessor's Manual on the accompanying CD-ROM.

4.3.1. XML abstraction for the GObject model

A Pakt server maintains a local object hierarchy and allows network clients to read and manipulate an XML representation of that model. A client can request arbitrary fragments of the hierarchy. For example, the pipeline depicted in Image 1 represented as XML with a depth of one:

```
<gst:thread name="thread0" priority="NORMAL"
            state="STATE_PLAYING">
  <gst:videotestsrc name="videotestsrc0"/>
  <gst:i420split name="i420split0"/>
  <gst:average name="average0"/>
  <gst:difference name="difference0"/>
  <gst:graytoi420 name="graytoi4200"/>
  <gst:xvimagesink name="xvimagesink0"/>
</gst:thread>
```

(result of a <get target="/threado" depth="1"/> request on the example pipeline)

One of the elements of that pipeline will be represented as:

```
<gst:average name="average0"
        bypass="wConverterBypass_process"
        roi="" mask="255" weight="0.010000"
        state="STATE_PLAYING">
  <gst:pad name="sink"/>
  <gst:pad name="src"/>
  <gst:pad name="thru"/>
</gst:average>
```

(result of a <get target="/threado/averageo" depth="1"/> request on the example pipeline)

A simple performance interface running on the same or a different computer could provide the user with a set of sliders and other controls for the parameters of a specific processing pipeline. When the user modifies one of the controls, the client can communicate the update in the form of simple *set* messages:

```
<set target="/thread0/threshold0/threshold"
     value="87"/>
```

The client can also register to receive updates whenever a parameter changes:

```
<monitor source="/thread0/threshold0/threshold"
         destination="/foo"/>
```

Details of the Pakt protocol can be found in the accompanying ›accessor's manual‹[12].

4.3.2. Modular integration of third-party libraries

Pakt integrates other libraries (like GStreamer and GTK) in the form of modules that provide the binding from an XML namespace's allowed element names to wrapper classes for the library. The functionality of a Pakt server (or Pakt-based client) can be

12 Included on the accompanying CD-ROM.

defined at calling time (and even run-time) by specifying which modules to load. The same program (paktd/paktc) can at one time be the wrapper to execute a GStreamer pipeline, and at another run a GUI client to control such a pipeline.

4.3.3. Prototypical Access Clients

Three proof-of-concept clients utilizing the Pakt framework have been implemented. All of them work by loading specific Pakt modules needed to run the client, requesting a copy of the currently running pipeline from a Pakt server, transforming the received XML with an XSLT stylesheet and instantiating the resulting XML to display a GUI window with the relevant controls. Apart from extending the set of Pakt modules (in this case, for GTK, Graph Visualization and simple MIDI messages), no actual programming was involved in the construction of these clients[13]. Their core functionality can be found in the respective XSLT stylesheets (to be found in the accompanying code archive).

13 In case of the Generic Controller, some objects linking the server-side parameters to client controls have been written; by now, this functionality could be replaced using more generic mechanisms.

4.3.3.1. Generic Controller

Image 3: Screenshot of the generic Pakt/GStreamer client

The generic controller application connects to a Pakt daemon running on a possibly remote host, and transforms the XML representation of the running GStreamer pipeline into a model of a GUI window with controls for all the properties of the pipeline's elements, and instantiates that model. The controls are connected to the server components bidirectionally, that is, they will update the server-side properties when they are changed, and likewise display the new value when the original property is changed (from another client, some other external controller, or by the element itself).

4.3.3.2. *Graph Visualization*

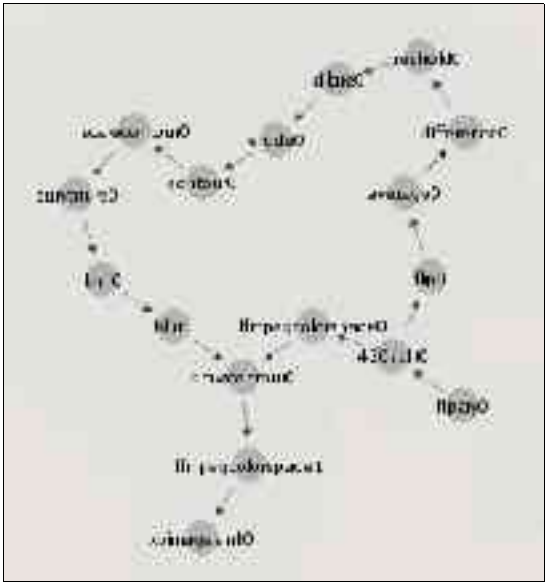

Image 4: Screenshot of the graph visualization client showing an example GStreamer pipeline

The graph visualization client transforms the server-side pipeline to a set of GNOME Canvas[14] elements in a GTK window. The Canvas elements are enclosed by special invisible *node* and *edge* elements that iteratively determine the graph layout based on a simple two-dimensional force-driven layout algorithm[15].

4.3.3.3. *Midi Dial Configuration*

In addition to the two relatively generic clients described above, a prototypical example client for a very specific application has been developed. As I own a *Doepfer PocketDial*, a hardware box with sixteen rotary encoders sending MIDI messages, I have started to implement a simple MIDI module for Pakt. To connect the elements representing the dials on the box to parameters of the pipeline, I have developed a simple client that will open a window with a button and text-entry widget for each dial, connect to the server to receive a list of valid parameters and provide that list as a pop-up menu for each dial. When a specific parameter is chosen from the menu, the corresponding dial will be

14 [libgnomecanvas]

15 Also called *spring-graph*, for example, see [Golovchinsky95]. The graph layout algorithm is implemented in the *graph* submodule of *glasnost*.

connected to that parameter. The text entry widget allows setting of the step size, the value by which each tick of the dial will in- or decrease the respective parameter.

Image 5: Screenshot of the MIDI Dial configurator

4.4. The ›Warsaw‹ library

4.4.1. Introduction

The *Warsaw* library enables easy integration of third-party image processing libraries into GStreamer. It's basic function is to separate the *processing model* of a component from the *data representation*. This way, the library can provide base classes for various processing models and wrappers for specific data types, which a single component can then use together to produce a specific processor for a specific type of data. By using Warsaws functionality for encapsulating the details of the GStreamer format negotiation and adapting the proprietary data types of some integrated library, an actual filter instance that calls a library function with a certain set of parameters becomes a trivial piece of code[16].

4.4.1.1. GObject to C++

The GObject model implements the object-oriented programming paradigm in plain C. While this has some obvious advantages (namely portability and

16 For example, see the cvBlur element in the code archive.

language binding support), it also has one important disadvantage: object-oriented (OO) features (inheritance, polymorphism, virtual functions, etc) are not directly supported by the language syntax and therefore harder to follow, require more typing and the code is harder to understand. The C++ language does support the OO paradigm within its syntax, and thus is a superior choice in a project that doesn't need language bindings and high portability support. As Warsaw is for supporting plug-in development, the language binding argument is negligible: plug-ins are accessed via the defined GStreamer plug-in API, which is obviously supported by Warsaw-derived elements. The portability argument remains, but C++ seems well supported on all major platforms that qualify for use in video processing.

Warsaw uses C++ primarily to have a nice structure for deriving new Elements. Additionally, a specific C++ feature (template classes) is used to separate the processing model from the data representation (see below). Within a new element, one is free to use C++ features for substructures, or stick with plain C, thanks to the fact that C++ is›only‹ an extension to C.

4.4.1.2. Filter types as base classes

Warsaw implements a range of filter types as base classes for new elements. These filter types represent common processing models and can be chosen to fit the processing task at hand. If a new filter doesn't fit any of the implemented filter types, it is still easy to derive a new element that will.

This model has the obvious advantage that a specific processing model is implemented only once, at a central location. Features that should be pervasive to all filters following a certain processing model, and indeed the major part of the interface to GStreamer's plug-in architecture, is implemented there. The *Bypass* properties and the idea of *Thruputs (see 4.4.3)* are examples of such features. If the idea for such features changes or is extended, or if the GStreamer model changes, it is easy to adapt the whole set of elements by modifying the base classes[17].

17 Note that, of course, this behavior follows directly from the OO programming paradigm, and as such it could have been implemented with the GObject model. See 4.4.1.1 for reasons for the choice of C++ over C.

4.4.2. Processing models (filter types)

As described above, one of the primary features of the Warsaw library is that it separates the processing model (filter type) from the representation of the data to be processed (data type). A new element is implemented as a combination of a filter type (it's template base class) and a number of data types (template arguments). This way, once a certain set of each is implemented, one can derive new elements as arbitrary combination of those, and concentrate on the actual processing to be done. When integrating a third-party library, once the data types used by the library function are integrated, the creation of a new filter that wraps a library function becomes trivial.

While filter types implement the processing model and interface to GStreamer's chain/loop/get-style scheduling paradigms, data types wrap the incoming and outgoing raw data buffers to specific data representations (eg., image structures used within a certain library) and interface with the ›caps negotiation‹ part of GStreamer, that is, they describe the content of the buffers in a way GStreamer can understand, to allow for the complex format negotiation between the elements to be connected.

This section describes the filter types (base classes) currently implemented in the Warsaw library. Note that I do not attempt to cover all possible processing models that could be implemented with the GStreamer infrastructure, but only a very basic set of important models.

4.4.2.1. Producer

The Producer class implements a component that emits buffers of a certain format. It has exactly one output. It can be used for both elements that produce data ›out of nothing‹, like a test image generator, and elements that interface to some external sources (like hardware interfaces, or data stored on disk).

4.4.2.2. Filter

The basic filter type is an element that has exactly one input of a certain data type and exactly one output of the same. It is used for implementing strictly ›in-place‹-style filters that only modify part of the received data, such as a

text overlay. It can also be used for ›consumer‹-type elements (such as a video display component or a network sender) that provides a Thruput of the (unmodified) data, so that it can be subjected to further processing.

4.4.2.3. Converter

Probably the most common processing model, a Converter is an element with one input and two outputs, where one of the outputs provides the identity of the data received on the input (is a Thruput). The other (main) output delivers data in the same format, but with modified content. It is used for all ›out-of-place‹-style filters, those that touch each part of the input data anyhow, so the additional copy makes no difference in processing time. In other words, the Thruput feature comes at close to no cost and can thus be implemented for all components of this type.

4.4.2.4. Transverter

A Transverter element is similar to the Converter model but differs in that the main output format can be different from the input format. The *Bypass mode* cannot be generically implemented with this kind of element, and additional logic to adapt the output and input formats is required.

4.4.2.5. Mixer

The Mixer filter type is used for components that join two streams of data of the same format. It has exactly two inputs and three outputs. Two of the outputs provide the input material (are Thruputs), the third (main) output delivers the mixed signal.

4.4.2.6. Inserter

The Inserter model is similar to the Mixer type in that it again joins two data streams. With Inserters, the streams can be of different format though, and Inserters will not provide a copy of the first (›master‹) input, but will render data from the second (›slave‹) input unto the first (insert the slave stream into the master stream).

4.4.2.7. *Other/more complex processing models*

As you can see by looking at the source code[18] for the base classes described here, they are not very complicated. Likewise, implementing other types of processing models is not hard. Warsaw is, in its current state, still limited to a fixed number of inputs/outputs and thus it is impossible to implement some kinds of elements that require a variable number of connections without modifying libwarsaw itself. Yet, for other component types with a fixed number of connections, implementation is near-trivial.

4.4.3. Thruputs and Bypass

There are two kinds of features implemented in all filter types whose processing model allows them: *Thruputs* and *Bypass* mode.

›Thruputs‹ incorporate the idea that while a component of course usually has a primary function and thus a primary data output, the user might still be interested in the unmodified data. For example, a user might want to scale down a high resolution video stream for efficiently finding the rough position of a large object, but then crop the bounding rectangle of that object from the original material for further, more detailed analysis. For filter types that copy the data while processing, Thruputs can be provided without requiring any additional processing power, thus they are implemented in the base classes that come with Warsaw (namely, *Converter*, *Transverter*, *Mixer* and *Inserter*).

›Bypass mode‹ enables the user to temporarily turn off a certain component. While of course the processing graph could be modified to insert or delete a certain element, this usually involves at least pausing the processing for a noticeable period of time. When in bypass mode, a component will not actually do any processing, but pass the input buffers unmodified to its output. Bypass mode is implemented for all processing models that can support the idea: *Filter*, *Converter* and *Inserter*. In case of a Converter in bypass mode, on receiving of an input buffer it will check if the Thruput is connected to another element; if it is connected, the data will be copied and passed on to both Thruput and Output, if not, it will be routed straight to the Output.

18 On the accompanying CD-ROM

4.4.4. Data Types

Several basic and some more specific data types have been implemented to allow for an easier access to buffer data, while retaining compatibility with existing GStreamer plug-ins where possible.

Most mentioned video types exist in versions for fixed and variable sized streams. With a variable-sized video stream, every frame can have different dimensions. Width and height, as well as position in a possible original stream, are prepended to each buffer as 32bit integers. As this functionality is wrapped in the data type classes, components that don't care about a constant size can handle both variable and fixed video.

Some of the data type classes adopt the stream to proprietary data structures for some specific library (GdkPixbuf, IplImage). This is done by allocating a structure header once during format negotiation, and setting of the data pointer when a new buffer is received. The component can then directly use the structure to invoke respective functions of the integrated library.

Data Type	*Description*
RGB	3-Channel, 8-bit red/green/blue packed video
RGBA	4-Channel, 8-bit red/green/blue/alpha packed video
I420	3-Channel, 8-bit, 2x2 subsampled YUV video
Y800	8-bit grayscale video
NChannelCV	1 to 4-Channel, 8-bit generic video wrapped to IplImage structures for OpenCV.
generic video	n-Channel, 1/8/32-bit integer or 32-bit floating point video
generic CV	based on generic video, wrapped to IplImage structures for OpenCV
GdkPixbuf	based on RGBA, wrapped to GdkPixbuf structures
Point Data	Variable-width, 2 line, floating point for storing tracking points (the lines consisting of x and y, respectively)
Contour Data	Variable-width, 2-5 line, floating point for storing contour data (the lines consisting of x,y, and optionally first, second and third derivation, respectively)

Table 1: Data types currently implemented in Warsaw

4.4.5. Properties (filter parameters)

As most components in video processing not only act on the pure data, but incorporate a set of parameters controlling the algorithm, there has to be a way to set (and get) those parameters during setup of a processing pipeline, as well as at run-time. GStreamer elements use the more generic GObject *Properties* to define such parameters by their name, type, defaults, valid values (ranges/enumerations) and short description. Warsaw wraps this concept into C++ classes and thereby allows for very easy definition and manipulation of filter parameters. A range of property types has been implemented, as listed in Table 2.

Class name	*Description*
wFloatProperty	floating-point value with validity range
wDoubleProperty	double-precision floating-point value
wIntegerProperty	integer value with validity range
wBooleanProperty	boolean (true/false) value
wStringProperty	string value
wEnumProperty	Enumeration value with list of allowed values
wFloatArrayProperty	list of floating-point values
wPointerProperty	generic C pointer value

Table 2: Property types implemented in Warsaw

4.4.6. Components integrated or developed

Over the course of my diploma period, I have created nearly seventy GStreamer elements using the Warsaw library. The purpose of most should become obvious from the short one-line description below. The functionality of the more complex plugins will be described in chapter 5. For reasons of completeness, I am listing all created plugins here, grouped by the containing module (which is defined mostly by the libraries integrated with that module). Please consult the accompanying source code archive (and gst-inspect, if you have installed the plugins) for details.

4.4.6.1. cv

The *cv* module integrates functions from the [OpenCV] library, providing various basic and some more advanced facilities for analyzing video.

Component	*Status*	*Description*
adaptive-threshold	good	adaptive binary threshold
addup	fair	add up (and clip) all channels of rgb(a) video into grayscale
average	good	running average
blur	fair	smooth image
camshift	fair	CamShift algorithm
canny	good	Canny edge detection
contour	fair	finds the largest contour in binary image
contourpoints	exp.	find weighted curvature minima/maxima between zero crossings
curvature	fair	calculates L1 curvature for chain
difference	good	calculate pixelwise absolute difference
dilate	good	dilate image
disttrans	fair	distance transform
erode	good	erode image
flip	good	flip (mirror) image horizontally or vertically
hsv	good	converts RGB to HSV
matte	fair	extracts matte by alpha mask
mhi	good	calculates motion history image
moments	fair	calculate contour moments and derive centroid and principal axes
multiplys	good	multiply with scalar
pyrdown	fair	performs downsampling step of Gaussian pyramid decomposition
pyrup	fair	performs upsampling step of Gaussian pyramid decomposition
scalecontour	fair	scale/shift contour
setchannel	fair	set a channel of packed video
threshold	good	binary threshold

Table 3: Components in the ›cv‹ module

4.4.6.2. *rectify*

The components found in the *rectify* module implement the functionality for restoring the originally projected image from a distorted version as perceived by the camera (described in5.1).

Component	*Status*	*Description*
derectify	fair	reverse-rectify image based on stored calibration data
rectify	fair	rectify image based on stored calibration data
rectify-calibrate	fair	round-trip calibrate for rectification
rectify-calibrate-contrast	fair	round-trip calibrate contrast for rectication

Table 4: Components in the ›rectify‹ module

4.4.6.3. *libart and cairo*

These two modules provide components for rendering data onto a video stream, by utilizing functions from the [cairographics] and [libart] libraries.

Component	*Status*	*Description*
checkerboard	fair	draw a checkerboard pattern
circle	fair	draw a circle
color	fair	produce a solid color image
plot	fair	plot a float list
scanline	fair	draw a moving scanline
drawcontour	fair	render contour with cairo
drawpoints	fair	render points with cairo
text	good	render a string with cairo

Table 5: Components in the ›libart‹ and ›cairo‹ modules

4.4.6.4. *buffer*

The *buffer* module contains three components for handling of in-memory loop-buffers, as described in 5.5.3.

Component	*Status*	*Description*
buffer	fair	store into a loop buffer
delay	fair	delay using a loop buffer
player	fair	play from a loop buffer

Table 6: Components in the ›buffer‹ module

4.4.6.5.pixbuf

Some simple compositing methods are implemented in the *pixbuf* module, utilizing the GdkPixbuf library contained in [GTK]. Compare section5.5.2.

Component	*Status*	*Description*
copy	good	crop a part of image as GdkPixbuf
fixate	fair	scale a variable GdkPixbuf to a fixed size
imagesrc	fair	load images from disk
paste	fair	paste a GdkPixbuf on image

Table 7: Components in the›pixbuf‹ module

4.4.6.6.Miscellaneous components

Finally, components that implement some function directly, without using an external library, are placed in the *misc* module. Table 8 also contains some components for recording and playing back video using [FFMpeg], and the [mesapaste] component using the [mesa] OpenGL renderer for simple alpha compositing.

Component	*Module*	*Status*	*Description*
ffplay	ffmpeg	good	simple video player using ffmpeg
ffrecord	ffmpeg	good	simple video recorder using ffmpeg
mesapaste	gl	fair	compositing using mesa
i420split	y800	good	split I420 into three Y800
rgbasplit	y800	fair	split RGBA into four Y800
rgbsplit	y800	fair	split RGB into three Y800
invert	misc	good	invert image
move	misc	good	move position of variable video
setalpha	misc	fair	set alpha value to a fixed value
settle	misc	good	wait until video has settled to black
trackpoints	misc	fair	naive point tracker
tracker	tracker	good	simple sequential scan labeling blob tracker

Table 8: Miscellaneous components

5. Application

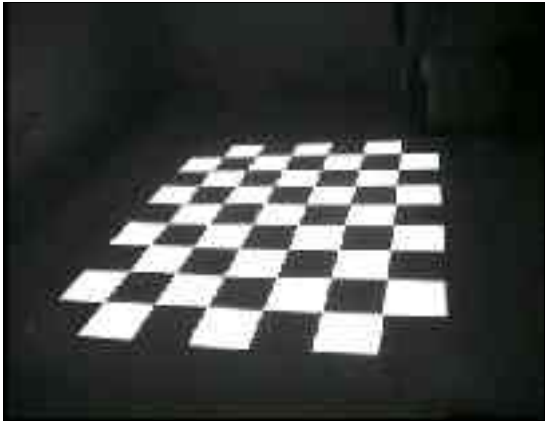

Image 6: Original image perceived by the camera, as used for the demonstrations of rectification below.

Multiple methods of tracking a human body in space have been developed[19]. I am applying only vision-based techniques, both because of their limited hardware requirements (they can be performed using only commodity hardware) and because I expect techniques based on purely visual properties to progress strongly due to further increases in processing power of standard workstations and new software algorithms that are being developed.

It should be noted that I have developed the rectification algorithm (5.1), some of the contour processing (5.4) and the buffering methods (5.5.3) by myself, while the algorithms mentioned in 5.2 and 5.3, and the method of finding contours (5.4) are based on the OpenCV library.

19 Compare [Paradiso97]

5.1. Rectification

One of the requirements of the performance setup (6.2.1) is to be able to analyze performer motion in a stage scene that is lit by a single frontal video projector. To allow calculating a meaningful difference between the image the computer projects unto the scene and the image perceived by the (fixed) camera watching the scene, I have to deal with the spatial distortion imposed by the positions of camera, projector and screen.

I have developed a simple rectification method to restore the original planar production image from its distorted appearance on the camera image. In the experiments throughout the project, this method prove to be relatively exact, and robust to even extreme camera angles. It integrates all distortions imposed by position and properties of the involved devices by applying a very high-level ›round-trip‹-calibration method.

5.1.1. Displacement Maps

The basis for the rectification method is the concept of a two-dimensional displacement map, where for each output pixel the originating position in the original image is being stored. Displacement maps are commonly used for applying spatial distortions to an image, for ›distortion mirror‹ effects or in audio visualization, as they provide an efficient means to apply a pre-calculated distortion to a video stream.

5.1.2. Calibration

In the calibration process, the display device is fed with a simple white rectangle that moves along the screen on a rectangular matrix. The camera signal is analyzed as to where that rectangle appears on the final image, and the relation between *produced* and *received* position is stored to produce a low-resolution displacement map.

Furthermore, the borders along this displacement map are extrapolated to accommodate for the fact that we cannot detect the rectangle's position reliably near the image borders. It is then scaled to the size of our production image using bicubic interpolation.

5.1.3. Forward

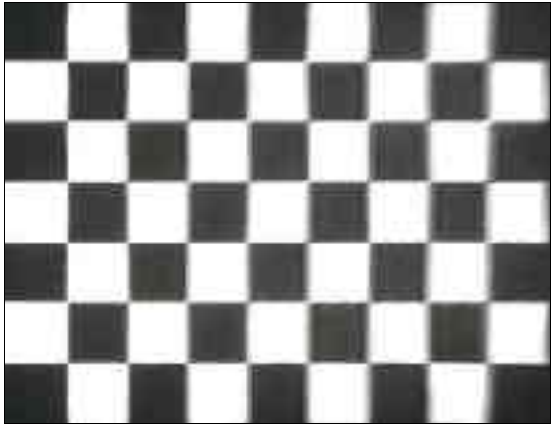

Image 7: Rectified version calculated from camera image

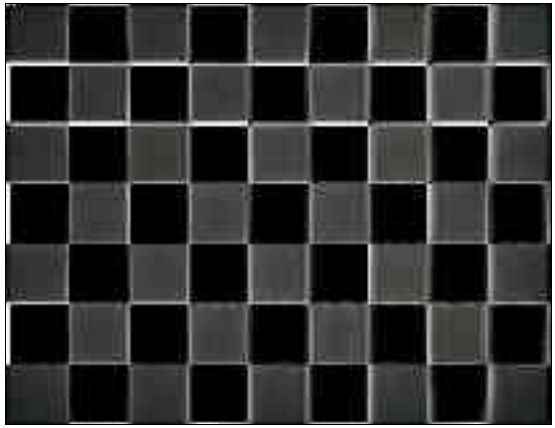

Image 8: Image 7 compared to the original pattern

Applying the resulting displacement map to the camera signal produces a *rectified* image that resembles our original production image (Image 7), given that no object interferes with the feedback loop. Thus, the rectified image can be compared to the original image (Image 8) and, given some tolerance to noise due to imprecision of the algorithm and video signal transmission, an object placed in the projector beam and its shadow can be made out.

5.1.4. Backward

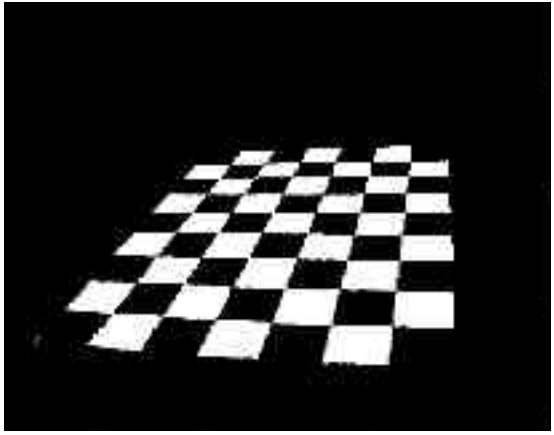

Image 9: Derectified version of the checkerboard pattern

Image 10: Image 9 compared to camera image

Once the rectification is calibrated, it can also be reverted to compare a distorted version of the original image with the camera signal (Image 9 and 10). This way, we can also detect parts of the object that are lit by the projector beam but do not appear in front of the screen from the camera's perspective. As the problem of inverting the displacement map clearly exceeded my mathematical understanding, the reverse (de-)rectification remains imprecise.

5.1.5. Contrast

In addition to the heavy spatial distortion, the projector/camera combination imposes another transformation on the original image: the colors and brightness are modified. I have introduced another calibration step (contrast calibration) to compensate at least a little bit for that fact. The contrast calibration routine finds the resulting brightness values for a black and a white version of each pixel projected and stores these values in a *luminance map* for the rectification process to accommodate for the shift in contrast. While that routine is again rather primitive and imprecise, and subject to brightness spill from neighboring pixels, the resulting image clearly improved.

5.2. Binary Presence Image

The first step of basic computer vision is often to calculate a binary image representing the probability of some object being present in the signal, where white pixels represent *presence* and black pixels *absence* of an object. Given a fixed camera, no (or little) occlusions and relatively stable lighting conditions, the calculation of such an image is quite simple.

5.2.1. Average/Difference/Threshold

The basic idea is to have an image of the (object-less) scene background, and compare the current frame of the video signal to that image pixel-by-pixel. The absolute difference between two pixels on the same position represents the likelihood of an object occupying that pixel.

Image 11: Running average of the video stream used in the examples

To get the background image, one could shoot a single frame of the scene when no objects (or people) are inside the picture and use that as a reference frame. This method is highly susceptible to changing lighting conditions and even very small changes in camera position, though, so a better method is to use a running average of the video stream to calculate a hypothetical background image (Image 11). While that method is vulnerable to people that don't move much (they will gradually fade into the background image, and once they move away will leave a false positive behind), an acceptable compromise can usually be found.

Once an image of the background is available, we can compare it to the current video frame to produce a greyscale probability image (Image 12), which can subsequently be thresholded to give the binary image we set out for (Image 13).

5.2.2. Shadow

In the specific case of the echo & narziss performance setup (see 6.2.1), I use an even simpler method to find a silhouette image of the performer: a camera pointed at the screen from a very high angle and rectified using the process described in 5.1 will produce an image of our original projection with the shadow of the performer clearly visible. Given that the performer doesn't move too close to the screen and the image is bright enough in all areas, calculating the binary presence image is a mere matter of an inverse threshold.

Image 12: Difference of background and current image

Image 13: Thresholded difference image

5.2.3. Difference to known projection image

Again in the specific case of my performance setup, I can apply another method to calculate the binary image, namely by comparing the camera signal to a de-rectified (see 5.1.4) version of the image that was projected onto the scene, or likewise, the original version of the projected image to a rectified (see 5.1) version of the camera signal. The drawback with this method is that it is computationally relatively expensive, and results in an image where both the performer herself and her shadow are marked. To avoid detection of the performer's shadow with this method, it can be used with the frontal camera and use the silhouette detected by the angular camera to darken those pixels in the original image that will be occluded by the performer's shadow, before comparing it to the perceived camera image.

5.3. Further pixel-based analysis

5.3.1. Image Moments

Image 14: Centroid, principal axes and extents (scaled) marked

Once the video stream is processed to produce a binary presence image like described above, we can calculate *image moments*, global statistical values about the analyzed picture that describe some basic features of the perceived object, namely its *centroid* (or ›center of gravity‹), *area*, *global orientation* and *extents* along the principal axes. These are significant, traceable parameters that allow for a complex coupling with sound and image production.

5.3.2. Motion History Image

Image 15: An example MHI

A *Motion History Image* (MHI) is a floating-point image that is continuously updated from a binary image. All pixels that are positive in the binary image will be set to the current ›timestamp‹ value in the MHI. Clipping a range of values from the MHI and scaling it to 8-bit values gives a grayscale image like Image 15.

5.3.3. Edge Detection

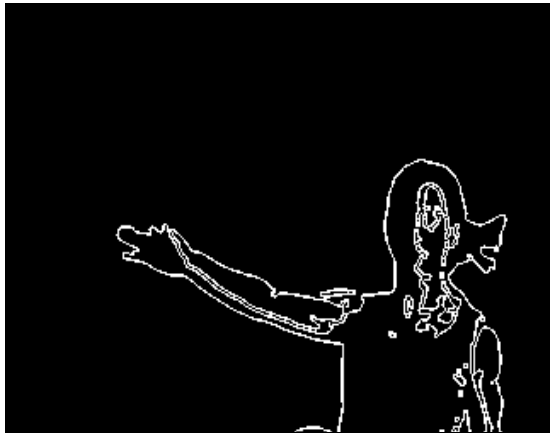

Image 16: A result image produced by the Canny edge detector

Various methods of finding edges in a (binary) image exist. Image 16 shows the result of the Canny edge detector applied to a binary image of the sample sequence. Some ›noise‹ (eg., tracing of shadow lines along the arm in the example image) can be reduced by *dilating* and *eroding* the picture before running the edge detection to reduce holes in the silhouette. In the resulting image we can easily find connected components and contours, as described below.

5.4. Contour

Image 17: Video frame with detected contour of performer

All vision algorithms described so far process the video stream in its digital representation as a two-dimensional pixel array. I made a (successful) attempt to exceed that level with the detection and processing of *contours*.

Starting from a binary image, the contour finding method implemented in OpenCV applies an edge detection algorithm, and scans the image left-right, top-bottom for the first non-zero pixel. From there, all connected pixels are followed to produce a path describing the connected component. On the way, edge pixels are zeroed, so a further run finds another component, until the edge image consists only of zero pixels.

From the resulting list of contours detected, I choose the largest for further processing and disregard the rest (I am expecting just a single human body in the image). The data describing this contour is stored in a variably-width floating-point image with a height of exactly two lines, as shown in Table 9:

x_1	x_2	…	x_n
y_1	y_2	…	y_n

Table 9: Layout of a contour image

The resulting image can be manipulated using the same components that process normal images (although the effects will of course be different). For example, Image 18 shows the results of a gaussian blur applied to the contour image with horizontal blur levels of 1, 10, 30, 50, 75 and 100, respectively.

Image 18: Gaussian blur applied to a contour image

5.4.1. Curvature

Image 19: Curvature along the performer's contour

An interesting property about contours is that of its *curvature*. There are multiple definitions of curvature[20], I calculate a first derivation of the contour by calculating the angles between two successive contour points, then derive the curvature by subtracting each pair of adjacent angles. Both derivations are appended to the bottom of the contour image, to produce an image layout as depicted in Table 10.

x_1	x_2	...	x_n
y_1	y_2	...	y_n
θ_1	θ_2	...	θ_n
c_1	c_2	...	c_n

Table 10: Layout of a contour image with curvature data

5.4.2. Local maxima of absolute curvature

I use the calculated curvature to derive a set of *points of interest* for control of the audio production. Originally, I thought that cutting the contour in parts (as to separate torso and limbs of the performer) would be trivial, but sadly, it is not. After

20 See [Fisher04]

reviewing some current research on parsing contours[21], I resorted to a much simpler approach that still produces traceable results: I find the weighted local maximums and minimums between zero crossings of the curvature plot. Considering Image 20, three maximums and three minimums clearly stand out. The first minimum and the last maximum are disregarded, as the curve does not cross zero before, respectively after the maximum.

Image 20: Plot of the smoothed curvature along the primary contour of Image 13

Calculating an average over the area of the hub (or dent, respectively) gives a weight of the detected interesting point. When I mark the points with rectangles representing their weight on the original contour, I get the image depicted in Image 21.

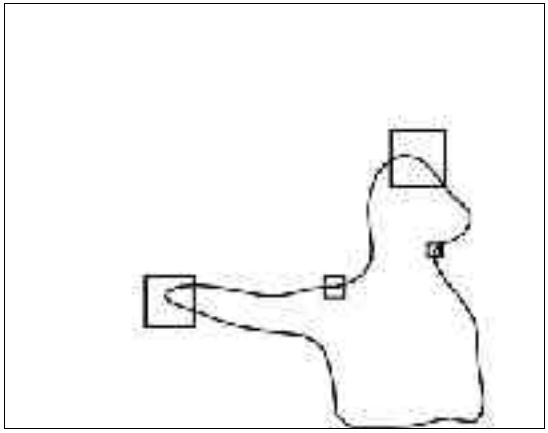

Image 21: ›Points of Interest‹ marked on the original contour

21 Compare [Azarbayejani96]

5.5. Synthesis

5.5.1. cairographics

After initial experiments with libart[22], I later switched to the cairographics library[23] for drawing vector graphics and text onto the video stream. Cairo seems to evolve to a general-purpose two-dimensional drawing library, very much like what OpenGL is for three-dimensional graphics. While I'm currently using only the internal software renderer, the development of various back-ends for cairo (OpenGL, PostScript, XRender) is in progress, which gives a promising perspective for the future. Indeed, cairo is being considered as integration as the low-level 2D rendering library for both major free desktop environments (GNOME and KDE).

Cairo already gives beautiful results with acceptable performance. The contours and moments information shown in the Illustrations above have been rendered using a few simple components integrating cairo[24], and it will be used intensively in the second act of the performance (compare 6.2.3.2).

5.5.2. OpenGL and GdkPixbuf

In addition to rendering with cairo, some initial efforts of using both OpenGL and the GdkPixbuf library have been undertaken. Especially the use of OpenGL promises interesting results in the future, yet for the use in this project, is limited to using the Mesa software-rendering library for simple alpha compositing (in the *mesapaste* component.

The GdkPixbuf[25] library has been tried out for some simple operations (matte extraction, alpha compositing and scaling), but as it doesn't support full alpha transparency yet and is also relatively slow, for most of these better variants have been found later (matte extraction is done using functions of the OpenCV library; alpha compositing with Mesa; for

22 [libart]

23 [cairographics]

24 Obviously, as these are rendered as bitmap graphics, they don't turn out as beautiful in print. Using the PostScript back-end of cairo would have been appropriate for the printed versions, but was not a reachable goal within the given timeframe.

25 Part of [GTK]

scaling of variable video, in the *fixate* component, GdkPixbuf is still used but will be replaced by more efficient algorithms in the future). As an exception, I use GdkPixbuf for loading of compressed image formats, as it supports a wide variety of those (*imagesrc* component).

5.5.3. Buffering in RAM

A large set of possibilities for interesting image effects is opened up when video is not only available from compressed format on the hard disk, or as a live stream from a camera, but buffered in RAM in a raw (uncompressed) representation, as access to video stored in such a way is very fast. Two basic components for doing so have been developed:*buffer* and *player*. Both access the same structure that manages a loop-buffer of frames, where the *buffer* component stores every frame it receives, and the *player* re-inserts these frames into the processing pipeline according to its own speed and direction. Multiple players can be connected to the same memory buffer, and thus they can feed other compositing elements with different points of time in the same video stream. Some ad-hoc effects based on buffering video in RAM will be presented during the third act of the performance (6.2.3.3), mostly by varying the playback speed of two *player* components feeding a *difference* element. They will, for example, amplify and visualize the variation of two iterations of the same move performed by the dancer.

5.6. Outlook

Obviously, only a small subset of both the algorithms found in the OpenCV library and computer vision research in general could be integrated into Warsaw within the limited time of the project period. [Intel01] and [Fisher04] describe a wealth of techniques that promise further interesting results. [Dimitrov] presents an algorithm for finding skeletal graphs from binary images that could provide interesting data also within a near-real-time context. [Singh99] proposes the »short-cut rule« for parsing silhouettes that could imitate the human ability to quickly grasp important features of two-dimensional contours. Finally, many techniques are being developed for deriving depth information from stereo cameras, for example the »spfinder« system presented by [Azarbayejani96].

6. Performance

6.1. Theme

6.1.1. Narrative

I have chosen the ancient Greek legend of Narcissus and Echo as the basic narrative to interpret within this project. I will sketch out the story based on Ovid's version[26].

Jupiter, the highest of gods, orders the forest nymph Echo, who is blessed with the gift of conversation, to distract his wife while he is ›visiting‹ the nymphs. Indeed, Echo succeeds in holding back Juno from catching her husband in flagranti, by engaging her in a debate about the proper way of organizing a wedding. When Juno finds out what's going on, she is understandably enraged, and punishes Echo with the curse that makes her name known to us today: from then on, she shall only be able to repeat the last words spoken to her.

Like all other nymphs, Echo is in love with Narcissus. She follows him like a shadow when he is hunting in the forests, waiting for him to speak a word so she can reply and try to seduce him. When

26 [Ovid]

one day Narcissus lost his friends during a hunt, her chance has come. Alone in the forest, Narcissus, looking for his friends, asks »Is anybody here?« and »Here.« replies Echo. Narcissus is confused, as he cannot make out the source of that reply»Come!« he calls, and»Come!« answers Echo.»Why do you avoid me?«, Narcissus requests, and so does Echo. »Let us be together!«[27] he calls, and, happily, Echo replies with the same. Encouraged by these last words, Echo leaves her hide and embraces Narcissus, who is disgusted by such stormy passion, and states:»Leave me be, wench! I would rather die than let you have power over me!«. And Echo, disappointedly, replies nothing but »you have power over me!« She retreats to the mountains where she fades to stone until only her voice remains.

Not much later, Narcissus finds his own curse in the form of a beautiful pool, where he falls in love with his own reflection. He does not realize it is only an image of himself he sees in the water, and gets confused because that other being seems to reply to his smiles and flirts, yet evades whenever he tries to embrace or kiss it. So at some point he understands his error, and in a surge of desperation, beats his chest until he dies. His last words,»Oh, boy, whom I loved in vain. Farewell!«, are repeated by the now disembodied Echo. The nymphs mourning over the death of narcissus is also echoed, as they look for his body but only find a yellow-white flower which from then on carries his name.

6.1.2. Interpretation

Unlike most interpretations of that narrative (of which there are many to find[28]), my focus is not so much on the character of Narcissus, but on that of Echo. It becomes immediately clear that Echo not merely repeats the last words uttered to her, but by choosing exactly which words, enters a dialog. Even with her curse, she retains her old gift of being able to converse with anybody about anything. The tale is not only about what Freud later dubbed Narcism, the blind self-love, but just as much about the variation inherent in any repetition, about the changes introduced with every iteration. Echo is a mirror of

27 Ovid uses the nicely ambivalent »coeamus« here, which translates to »fit together; have sexual intercourse; gather; meet; assemble; unite«

28 [Orlowsky92]

sorts, but a magic mirror that will exaggerate and interpret any object placed in front of it.

Narcissus, at first, regards his own reflection as the image of an external entity. The distortion and variation imposed by the noise of the medium (the distortion caused by movement of the water surface, or the modification of the voice by the reflecting surfaces) cause the subject to misinterpret the true source of his perception and regard the reflection of his own actions as answers from a different being.

This is analogous to a behavioral pattern to be found with current user interfaces[29]: people seem to regard the computer as a person, a being with its own free will. Whenever ›it doesn't do as we tell it to‹, the computer is attributed a ›bad day‹, or even ferocity. Rarely do we realize how precisely and stubbornly the computer interprets our actions, and that an error is only very rarely caused by a real bug in the software (and never by ferocity on the computer's side), but much more often by a misconception on the user's side, a misunderstanding between the user and the interface designer.[30]

To turn things around, Narcissus is caught in a feedback loop like a computer stuck in an infinite processing loop: the object of his love loves him back, but yet the love cannot be fulfilled. »A novel wish for a lover: I wish, what I love, would not be with me.«[31] Like the classic video feedback loop of a camera pointed at a screen displaying the same camera's signal, all ›error‹, all noise inherent in the signal transmission is amplified and exaggerated until the signal itself is completely lost. In the case of Narcissus, his stigma of being incapable of love to anyone but himself, is amplified and amplified again until it becomes overpowering, until the signal itself disappears and only the error remains.

6.1.3. Representation

Following this interpretation of the myth, I chose to unify the ›characters‹ of Echo and the pool in one entity: the computer systems used in the per-

29 [Nassoo]

30 To be fair, we have to add that these misconceptions are mostly caused by bad interface design, and not by any alleged stupidity on the user's side.

31 Narcissus in [Ovid]

formance. The human performer will act the role of narcissus, the human being confronted with his own reflection. All audio and video material used in the performance will originate from that performer. The software systems described above will interpret the performers actions in visuals and sound, project these interpretations back onto the performer and start over again.

Yet, as I have described, Echo is not a 'stupid' repeater: she still has her own free will, and inserts her disembodied being into the loop. This is where the performers not present on the stage come into play. First, we had of course our artistic influences during the development of the systems itself, by using the information-processing abilities of the computers to reinterpret the live signal[32]. Furthermore, we will actively adjust parameters and function of the systems during the performance itself. We'll thus be doing just what Echo does in the classic narrative, when she clearly has a choice as to which last words she will repeat, and thereby enters a dialog.

32 After all, the most simple realization of my interpretation could have been achieved with a simple audiovisual feedback loop.

6.2. Realization

6.2.1. Technical Setup

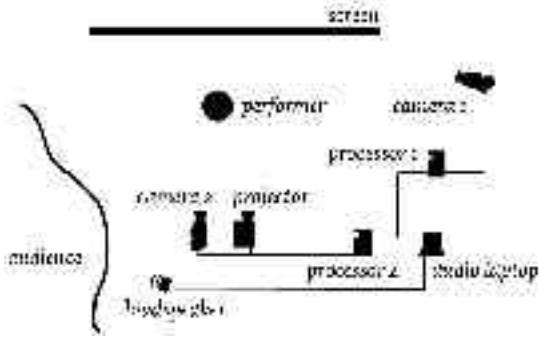

Image 22: Overview of the technical performance setup

After juggling with various possibilities for the technical setup of the performance, I decided to go for a relatively simple combination of a frontal projection, two cameras and two processors, as shown in Image 22. Camera one, watching the screen from a high angle as to avoid direct exposure of the performer, delivers its signal to the first computer that will perform the basic analysis parts of the setup.

It will derive a binary image displaying only the performer's shadow, as described in 5.2.2, and calculate image moments (5.3.1) and contour (5.4) to send those to both the musicians for audio synthesis and to the second computer for incorporation into the image generation. I will modify live parameters of the synthesis and analysis on the two processors with controls (4.3.3) from my laptop.

6.2.2. Participants

I have invited three artists for cooperation within this project. Alexandra Janeva works as a professional dancer and choreographer mainly in France, Croatia and Switzerland. The experience of watching one of her performances in Zagreb in 2001 has triggered my wish to increase and intensify my cooperation with professional dancers, so she became my obvious first choice for involvement into this project.

Oliver Macklott and Astrid Schwarz have both studied electroacoustic music at the University of Music in Vienna. Together they perform in various concert situations and manage to create alluring listening experiences with their unique use of new audio technologies. Some early experiments in exchanging data to synchronize video analysis with audio production and details of that audio production with video generation have proven promising, so they also became an obvious choice for integration within this project, to continue and extend those early experiments and hopefully together approach a level of exchange that is not only fertile to us, but also able to inform the audience about the possibilities of such coupling.

6.2.3. Dramaturgy

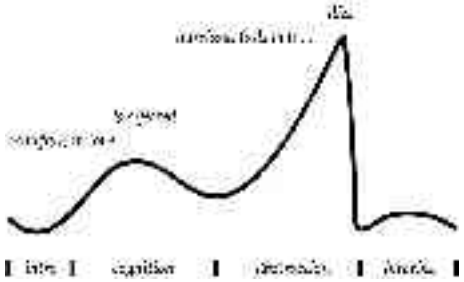

Image 23: Dramatic composition

In terms of choreography and general composition, the actual event itself will remain necessarily

improvised[33]. Nevertheless, I have, in dialog with musicians and performer, developed a simple dramaturgy to give the involved artists (including myself) a ›canvas‹ and a means of general orientation for the live play. This dramaturgyis divided into four major parts, as depicted in Image 23. It's description here remains vague, as details of the realization are subject to improvisation and rehearsal shortly before the performance.

6.2.3.1. Introduction

At the start of the performance I present a short introduction to the storyline in textual form on the projected image. It introduces Echo's background and the story of her curse.

6.2.3.2. Cognition

The second act displays the encounter of Echo and Narcissus, where Echo falls in love and is denied. Narcissus (the dancer) enters the stage, which is plainly lit in a natural green, to represent the forest setting, making his shadow clearly visible. Echo appears in the form of initially subtle and then increasingly complex silhouette representations of the dancer (Echo chasing Narcissus ›like a shadow‹) that interfere with the real shadow. Technical displays of the analysis are projected onto and next to the performer, approximating her outline at first roughly then with increasing precision.

The coupling of performer motion to sound should become obvious, and with increasing display of analysis data, peaks in a directly perceivable interaction. While initially interested, confused and curious, Narcissus strongly rejects Echo's approach at the peak of this act, causing her to retreat (images become less clear, finally disappear into a plain color again).

6.2.3.3. Abstraction

The third act of the performance tells the part of the story where narcissus is at the pool. The shift from the embodied Echo involved in audible conversation to the lifeless water reflecting the visual image of Narcissus is exemplified by the fact that visuals now

33 This is not a bad thing, as one of the basic ideas driving this project was the development of a generic toolkit that allows for just this kind of improvisation.

become less stylized and directly use images of the performer itself.

Initially, these are subtle and evasive; Narcissus falls in love with his image without realizing it is his own reflection; the dancer dances with herself. As Narcissus starts to realize what happened, the images of the performer are projected back onto herself, things get increasingly complex and feedback-like, representing Narcissus' struggle with himself. Analysis data becomes less precise in this setting, so does their coupling to sound, and the music, like the images, becomes increasingly chaotic. All involved parts culminate in a furious feedback until all signal is lost and only white noise remains. Narcissus dies.

6.2.3.4. Imprint

The last act uses material gathered during the second and third part, to replay the whole performance in reverse order without the performer interacting, a repetition and variation of the story, a quiet, calm epilogue, representing the nymph's mourning and Echo's echo of the same, and the futile search for Narcissus' body.

7. Summary

7.1. A Generic Toolkit

As described, I have developed two libraries, Warsaw and Pakt, that interface with the Open-Source media processing platform ›GStreamer‹ from two angles: Pakt integrates GStreamer and other GObject-based libraries to allow complex interaction with a running processing pipeline on a remote computer; Warsaw allows easy integration of signal processing libraries into GStreamer and development of new algorithms within the framework. Additionally, an effort has been taken to integrate a number of functions from OpenCV, cairographics and other freely available libraries into that framework.

Together, these systems comprise a flexible and extensible toolkit that allows for complex setups enabling performances that integrate image analysis and synthesis, communication with external processing systems and live interaction from multiple human performers.

Other applications of the same tools could include interactive and reactive art installations, VJ performances, and last but not least research in the field of human-computer interaction.

7.2. Improvisation and Performance

One of the focal points of the toolkit development was to allow improvisation on all levels: While usually dance performances involving computers require a lot of rehearsal and practice from both performer and engineers, the methods involved in this project are relatively robust to changing conditions and allow for free improvisation on the dancer's side.

Additionally, the developed components empower me to involve directly into the performance using various controls, and thus take part in the improvisation. Communication with the audio systems has been extended to enable complex coupling of video analysis and synthesis with audio production. While the actual set of parameters communicated is still relatively limited, the ground has been prepared for increasing cooperation.

Finally, the project involved various details of preparing and managing a contemporary dance performance involving three external artists and a significant amount of technology.

7.3. Freedom and Constraints

I have enjoyed very much the freedom to work on a set of my own ideas for such a long period of time, and I think I have shown that the lack of detailed a-priori definition of the expected results does not necessarily mean that a work is less focused or intense.

As within most projects, the main struggles were not to be found with the development of a vision, but with its realization. I think I have extended my tools to a degree where

1. they allow me to continue development in the area of performative and installation arts with a promising perspective, and
2. it makes sense to publish these tools as Open-Source software for application by others.

Apart from the technical results of this project, there is still one experiment whose outcome is yet to be seen: Does a dance performance on this experimental level have the ability to *enrich* both participants and audience?

7.4. Future Directions

To me, there is not much question about how to continue in the direction suggested by the project. First, I hope I can still work on projects like this even when I have to care about my income myself in a commercial background. Models for my own professional practice and private live that enable this will have to be developed. Second, the developed tools are all still in a prototypical stage, and need further stabilization, extension and experimentation. Many promising computer vision and graphic synthesis techniques remain unused.

One of the most important lessons learned within this project though, is that my work seems much more rewarding if I open it up to more intense cooperation. The work with the involved participants was good, but still rather limited, and should be intensified. Additionally, I hope to receive important feedback from others using the developed software for implementing their own ideas.

8. References

8.1. Pakt/Warsaw dependencies

[cairographics] Cairo vector graphics library, Version 0.1.22 (MIT License): http://www.cairographics.org/

[gnet] Gnet network library, Version 2.0.5 (GNU LGPL): http://www.gnetlibrary.org/

[gob2]The GObject Builder, Version 2.0.6 (GNU GPL): http://www.5z.com/jirka/gob.html

[GStreamer] GStreamer Open Source Multimedia Framework and associated plug-ins package, Version 0.8.0 (GNU LGPL): http://gstreamer.freedesktop.org/

[GTK], [GObject] GTK+ widget toolkit and associated glib/GObject support libraries, Versions 2.4.1 (GNU LGPL): http://www.gtk.org/

[libart] libart vector graphics library, Version 2.3.14 (GNU LGPL): http://www.levien.com/libart/

[libgnomecanvas] GNOME2 Canvas Library, Version 2.0.5 (GNU GPL): ftp://ftp.gnome.org/pub/gnome/sources/libgnomecanvas/2.0/

[libxml2] libxml2 XML/HTML processing library, Version 2.6.3 (MIT License): http://xmlsoft.org/

[libxslt] The XSLT C library for GNOME, Version 1.1.1 (MIT License): http://xmlsoft.org/XSLT/

[mesa] Mesa3D Software Rendering Library, as part of XFree 4.3.0 (XFree copyright, MIT-style license): http://mesa3d.sourceforge.net/

[OpenCV] Intel Open Source Computer Vision Library, Version 0.9.5 (Proprietary Intel Open Source License): http://www.intel.com/research/mrl/research/opencv/

[spidermonkey] SpiderMonkey JavScript-C engine, Version 1.5 (Mozilla Public License/GNU LGPL): http://www.mozilla.org/js/spidermonkey/

8.2. References to books and papers

[Azarbayejani96] Azarbayejani, Ali, Wren, Christopher, Pentland, Alex (1996): Real-Time 3-D Tracking of the Human Body [Electronic version]. M.I.T. Media Laboratory Perceptual Computing Section Technical Report No. 374, as appearing in Proceedings of IMAGE'COM 96, Bordeaux, France, May 1996.

[Dimitrov] Dimitrov, Pavel, Phillips, Carlos, Siddiqi, Kaleem (?): Robust and Efficient Skeletal Graphs [Electronic version]. Retrieved January 28, 2004 from http://www.cim.mcgill.ca/~shape/publications/CVPR00.ps.gz

[Fisher04] Fisher, Robert B (Ed.) (2004): CVonline: The Evolving, Distributed, Non-Proprietary, On-Line Compendium of Computer Vision. Retrieved January 19, 2004 from University of Edinburgh, School of Informatics Web site: http://homepages.inf.ed.ac.uk/rbf/CVonline/

[Golovchinsky95] Golovchinsky, Gene (1995): Subverting Structure: Data-driven Diagram Generation. Retrieved from: http://www.imedia.mie.utoronto.ca/people/golovch/publications/vis95/toc.html

[Intel01] Intel Corporation (2001): Open Source Computer Vision Library: Reference Manual. Retrieved November 21, 2003 from:

http://opencvlibrary.sourceforge.net/

[Nassoo] Nass, Cliff (2000): How People Treat Interfaces Like People: Social Psychology and Design. University of Washington Video Lecture, as retrieved from http://www.uwtv.org/programs/displayevent.asp?rid=602

[Orlowsky92] Orlowsky, Ursula, Orlowsky, Rebecca (1992): Narziß und Narzißmus im Spiegel von Literatur, Bildender Kunst und Psychoanalyse: vom Mythos zur leeren Selbstinszenierung. München, Fink.

[Ovid] Publius Ovidius Naso: Metamorphosen. Übertragen und herausgegeben von Erich Rösch. 11. Aufl., München und Zürich, 1988. Nach: [Orlowsky92].

[Paradiso97] Paradiso, J.A., Sparacino, F. (1997): Optical Tracking for Music and Dance Performance [Electronic version]. Presented at the Fourth Conference on Optical 3D Measurement Techniques, ETH Zürich, September 1997.

[Raymondoo] Raymond, Eric S. (2000): The Cathedral and the Bazaar [Electronic version]. Retrieved October 4, 2003 from http://www.catb.org/~esr/writings/cathedral-bazaar/cathedral-bazaar/

[Singh99] Singh, Manish, Seyranian, Gregory D., Hoffman, Donald D. (1999): Parsing silhouettes: The short-cut rule [Electronic version]. Perception and Psychophysics, 61: 636-660.

[Wilde] Wilde, Oscar: The Disciple. In: The Oxford Anthology of English Literature. Vol. II. 1800 to the Present. General Editors: Frank Kermode and John Hollander. New York, London, Toronto: 1973. From [Orlowsky92].

8.3. Web Links

[Combustion] http://www4.discreet.com/combustion/

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

[EyeCon] http://eyecon.palindrome.de/

[FFMpeg] http://ffmpeg.sourceforge.net/

[Houdini] http://www.sidefx.com/products/houdini/index.html

[Isadora] http://www.troikatronix.com/isadora.html

[Jitter] http://www.cycling74.com/products/jitter.html

[Keyworks] http://www.keyworx.com/

[LGPL] http://www.fsf.org/licenses/lgpl

[libdm] http://www.sgi.com/software/irix/tools/digitalmedia.html

[MAS] http://www.mediaapplicationserver.net/

[MediaKit] http://open-beos.sourceforge.net/tms/team.php?id=5

[Palindrome] http://palindrome.de/

[Reaktor] http://www.nativeinstruments.de/index.php?reaktor4_us

[Rockeby] http://homepage.mac.com/davidrokeby/vns.html

[Sony] http://www.eyetoy.com/

[TroikaRanch] http://www.troikaranch.org/