GABRIEL FINCH

LiVES: LiVES is a Video Editing System

RECIFE-PE – JULHO/2013.

UNIVERSIDADE FEDERAL RURAL DE PERNAMBUCO

PRÓ-REITORIA DE PESQUISA E PÓS-GRADUAÇÃO

PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA APLICADA

LiVES: LiVES is a Video Editing System

Dissertação apresentada ao Programa de Pós-Graduação em Informática Aplicada como exigência parcial à obtenção do título de Mestre.

Área de Concentração: Engenharia de Software

Orientador: Prof. Dr. Giordano Ribeiro Eulalio Cabral

RECIFE-PE – JULHO/2013.

Ficha Catalográfica

F492L Finch, Gabriel
LiVES: LiVES is a video editing system / Gabriel Finch.
-- Recife, 2013.
132 f.

Orientador (a): Giordano Cabral.
Dissertação (Mestrado em Informática Aplicada) – Universidade Federal Rural de Pernambuco, Departamento de Estatísticas e Informática, Recife, 2013.
Inclui referências e apêndice.

1. Software - Desenvolvimento 2. Prototipagem
3. Multimídia 4. Usuários de computador 5. Vídeo digital
I. Cabral, Giordano, orientador II. Título

CDD 005.1

UNIVERSIDADE FEDERAL RURAL DE PERNAMBUCO

PRÓ-REITORIA DE PESQUISA E PÓS-GRADUAÇÃO

PROGRAMA DE PÓS-GRADUAÇÃO EM BIOMETRIA E ESTATÍSTICA APLICADA

LIVES: LIVES is a Video Editing System

GABRIEL FINCH

Dissertação julgada adequada para obtenção do título de Mestre em Informática Aplicada, defendida e aprovada por unanimidade em 30/07/2013 pela Banca Examinadora.

Orientador:

[illegible]

Prof. Dr. Giordano Ribeiro Eulalio Cabral
Universidade Federal Rural de Pernambuco

Banca Examinadora:

[illegible]

Prof. Dr. Tiago Alessandro Espínola Ferreira
Universidade Federal Rural de Pernambuco

[illegible]

Prof. Dr. Jones Oliveira de Albuquerque
Universidade Federal Rural de Pernambuco

[illegible]

Prof. Dr. Geber Lisboa Ramalho
Universidade Federal de Pernambuco

ACKNOWLEDGEMENTS

The author would like to thank:
The staff and students at UFRPE.
All the LiVES users and contributors.
My family.

and the following, who have helped along the way:
Niels Elburg, Denis "Jaromil" Rojo, Tom Schouten, Andraz Tori,
Silvano "Kysucix" Galliani, Kentaro Fukuchi, Dr. Jun Iio, Oyvind Kolas,
Carlo Prelz, Yves Degoyon, Lady Xname, timesup.org, LinuxFund, VJ Pixel,
estudiolivre, mediasana, Felipe Machado, elphel.com.

RESUMO

Relativamente pouca pesquisa científica tem sido executado até à data atinente aos requisitos dos usuários de aplicativos de processamento de vídeo. Nesta dissertação, apresentamos um novo termo "Experimental VJ", e examinamos os requisitos de software para essa classe de usuário, derivados de uma variedade de fontes. Por meios desses requisitos, definimos os atributos que seria necessário um programa criado para satisfazer essas demandas possuir. Nós nos concentramos em uma ferramenta em particular - ou seja, LiVES - e mostramos como ele foi projetado e desenvolvido para atender a essas necessidades. Detalhes do desenvolvimento do LiVES e de sua arquitetura estão incluídos. Após isso, nós fornecemos algumas validações que o aplicativo LiVES está conseguindo em seus objetivos.

Palavras-chaves: Requisitos de usuários. Multimédia. Vídeo. Desenvolvimento de software. Prototipagem.

ABSTRACT

Relatively little scientific research has been performed to date regarding the requirements of users of video processing applications. In this dissertation, we introduce a new term “Experimental VJ”, and examine the software requirements for this class of user, deriving this from a variety of sources. From these requirements we infer the set of features which a program designed to satisfy them should possess. We focus on one tool in particular – namely, LiVES - and show how it has been designed and developed to fulfill these needs. Details of the development of LiVES and its architecture are included. Following this, we provide some validation that the LiVES application is succeeding in its aims.

Keywords: User requirements. Multimedia. Video. Software development. Prototyping.

List of Figures

List of Tables

Table of Contents

1 **INTRODUCTION**

Computer video applications are an increasingly important area of modern technology. Sherman (2008) wrote:

> *As video technology spun off from television, the mission was clearly one of complete decentralisation. Forty years later, video technology is everywhere. Video is now a medium unto itself, a completely decentralised digital, electronic audio-visual technology of tremendous utility and power. Video gear is portable, increasingly impressive in its performance, and it still packs the wallop of instant replay. As Marshall McLuhan said, the instant replay was the greatest invention of the twentieth century.*
>
> *Video in 2008 is not the exclusive medium of technicians or specialists or journalists or artists – it is the people's medium. The potential of video as a decentralised communications tool for the masses has been realised, and the twenty-first century will be remembered as the video age. Surveillance and counter-surveillance aside, video is the vernacular form of the era – it is the common and everyday way that people communicate. Video is the way people place themselves at events and describe what happened. In existential terms, video has become every person's POV (point of view). It is an instrument for framing existence and identity.*
>
> *There are currently camcorders in twenty per cent of households in North America. As digital still cameras and camera-phones are engineered to shoot better video, video will become completely ubiquitous. People have stories to tell, and images and sounds to capture in video. Television journalism is far too narrow in its perspective. We desperately need more POVs. Webcams and videophones, video-blogs (vlogs) and video-podcasting will fuel a twenty-first- century tidal wave of vernacular video.*
>
> *(SHERMAN, 2008, p. 161).*

The concept of video as "the people's medium" is even more relevant today. Smartphone technology has thrust a camera into the hands of a huge and growing segment of the population. Laptops and video games consoles increasingly come equipped with cameras as standard.

The way we work with video and view the results are also changing. Sherman again predicted:

> *Displayed recordings will continue to diminish in duration, as television time, compressed by the demands of advertising, has socially engineered shorter and shorter attention spans. Videophone transmissions, initially limited by bandwidth, will radically shorten video clips. The use of canned music will prevail. Look at advertising. Short, efficient messages, post- conceptual campaigns, are sold on the back of hit music. Recombinant work will be more and more common. Sampling and*

> *the repeat structures of pop music will be emulated in the repetitive ‘deconstruction’ of popular culture. Collage, montage and the quick-and-dirty efficiency of recombinant forms are driven by the romantic, Robin Hood-like efforts of the copyleft movement. Real-time, on-the-fly voiceovers will replace scripted narratives. Personal, on-site journalism and video diaries will proliferate. On-screen text will be visually dynamic, but semantically crude.*
>
> *(SHERMAN, 2008, p.161).*

If we examine the market for video tools, we may note that there is something of a lack of tools which empower individuals and groups to work comfortably to achieve their aims. Manovich (2001) explains what drives these goals:

> *If we are surrounded by highly dense information surfaces, from city streets to Web pages, it is appropriate to expect from cinema a similar logic. In similar fashion, we may think of spatial montage as reflecting another contemporary daily experience - working with a number of different applications on a computer at once. If we are now used to switching our attention rapidly from one program to another, from one set of windows and commands to another, we may find multiple streams of audio-visual information presented simultaneously, more satisfying than the single stream of traditional cinema.*
>
> *(MANOVICH, 2001, p. 328).*

Although Manovich talks here about cinema, the same factors may apply to amateur video. Sherman again describes what such video might look like:

> *Slow motion and accelerated image streams will be overused, ironically breaking the real-time-and-space edge of straight, unaltered video. Digital effects will be used to glue disconnected scenes together; paint programs and negative filters will be used to denote psychological terrain. Notions of the sub- or unconscious will be objectified and obscured as ‘quick and dirty’ surrealism dominates the ‘creative use’ of video.*
>
> *(SHERMAN, 2008, p. 162).*

If these are the kind of effects that a certain type of user might wish to create, how can we classify this type of user ? What are their specific requirements, and how can we satisfy them ? In this dissertation we examine these questions and propose a solution.

There are a great many software applications which exist that are directed towards processing video. However, none of these applications completely satisfy the requirements for a large segment of the market. If we agree with the statements of Sherman and Manovich above, then each of these tools falls short of meeting the needs of those who would work with modern methods of video production.

It is important to have a detailed specification for this target audience – those who wish to work in new ways with video, and whose needs are constantly mutating. In this dissertation we present an application called "LiVES"[1] which can be shown to satisfy this changing specification very well.

Development of the Free Software project, LiVES began in 2002, and it has been updated continuously since then. There have been more than 350,000 downloads of the source code to date (not counting binary versions).

Estimates of the cost to produce this software commercially have arrived at a figure of almost $4,000,000 USD, as will be discussed later. Other estimates can be made to show a figure of over 10,000 users. LiVES has also received some excellent reviews in the media

In **Chapter 2**, we will examine the state of the art for video applications, which can be roughly divided into the categories of VJ applications, video editing applications, and video programming environments. Here, we present a brief overview with examples of some instances of each type of application.

In **Chapter 3**, we discuss in detail the problems faced in (identifying our target audience), profiling their needs, and developing a system to satisfy them.

Chapter 4 describes our methodology for resolving the issues which were identified in the preceding chapter.

Chapter 5 deals with the concept of the LiVES application, the technical and non-technical requirements, and the challenges faced in developing such software.

Following this, **Chapter 6** explains the history and philosophy of LiVES, along with a look at the functional and technical architecture of the application.

In **Chapter 7**, we present the results of our study, to see how well LiVES fulfills the needs of our target audience.

The remainder of this dissertation consists of our conclusions, followed by a list of references, and appendices.

[1]FINCH, G. LiVES website. Available at: <http://lives.sourceforge.net>. Accessed: 13 Jun. 2013.

It should be noted that Appendix A contains a list of definitions for some technical terms (MIDI, OSC, and so on) used in this dissertation, to which the reader may wish to refer.

2 THE PROBLEM AND REQUIREMENTS FOR ITS SOLUTION

Throughout this dissertation we use a term of our own invention: “Experimental VJ”, which is not generally found in the literature. By this, we mean an individual, or a group collaborating together, from a particular subset of VJs – those who are interested in novel and experimental techniques, pushing forward the boundaries of this evolving art-form.

Experimental VJs have various requirements which can be different from individual to individual (or group to group) and can change over time. These requirements are therefore extremely dynamic.

There exist distinct categories of software which attend to these requirements. As mentioned in the preceding chapter, the vast majority of video software falls into the three categories of VJ applications, video editing applications, and video programming environments.

By itself, a program which fits within one of these categories may be insufficient to fulfill the requirements of an Experimental VJ. From a user's point of view, it would seem advantageous to combine these areas in a single system.

As an example, if a person working mostly with a VJ application wanted to prepare material (clips, camera footage, animation sequences) prior to presenting it, they would generally use a program such as Premiere, After Effects or Final Cut to do this, then use the VJ tool to present that material[2], since most, if not all VJ applications lack advanced or user friendly editing features.

Should that person then wish to control their presentation programmatically, they would likely be required to use a third tool, for example Pure Data, or Isadora.

In Chapter 4, we examine more closely the specific features which are lacking from example programs for each of the traditional categories of video programs.

2.1 ADVANTAGES

[2]WIKIPEDIA, VJing, Common Technical Setups. Available at: <http://en.wikipedia.org/wiki/VJing#Common_technical_setups>. Accessed: 13 Jun. 2013.

A single system which combined these areas would simplify the workflow for users in many instances, due to the need to employ only one application, rather than make use of multiple applications. This may have the additional positive effects of reducing the amount of processing required, and increasing the quality of the resulting output - since it may eliminate the need for conversions between formats, frame sizes, frame rates and so on which may be required otherwise when transferring material from one application to another. The time savings due to this may also be a factor. A VJ will often arrive for a performance, only to be handed some material (clips, camera footage and so on), which they are asked to incorporate in the performance. If this material needs some preparation, then it would be useful and convenient if this could be done entirely within the same application which they will use for performance – so that the results can be tested straight away. If preparation time is very limited, as is often the case, then it would be advantageous not to have to export/import, re-encode and arrange clips, then test to see how they fit with the rest of the material.

A second advantage for users may be that it would allow them to experiment and make greater use of their creative talents by exploring other areas of video processing. For example, anecdotally, various users who are focused mostly on non-linear editing have expressed that they find the use of VJ applications as "fun and stimulating for the imagination". On the other hand, those users who are more focused on real-time processing, would have access to the tools they need to both prepare their material, and also to record and possibly encode their performances, by making use of the non-linear editing features within such a system.

When talking about traditional video editing applications, Miles (2008) opines:

> *These systems, just as with word processing, offer all the advantages of the digital for the production of content, but remove them for the user at the point of publication. For example, while using a word processor it is trivial to move text, annotate it (with voice, image or other text), change fonts, resize the screen and so on. But as a word processor all of these tools are actually directed towards getting those words on paper (hence pagination, page numbering and so on). Once on paper, all of those functions just listed (and many others) are gone. It is exactly the same with video, where similarly the video work is malleable and fluid in quite extraordinary ways while being edited, but once committed to publication these features are removed – it becomes resolutely and immutably flat. This is what I have, elsewhere, described as the distinction between hard and softvideo, where in softvideo it is possible to imagine a video architecture and practice that is able to retain this granularity after publication, where videos can be created that consist of shots that no longer have a canonical sequence. The multiplicity of possible relations between shots, which granularity affords, can then be preserved and made available to the user or viewer as a material property of the completed video text.*
>
> *(MILES, 2008, p. 224).*

According to Miles' definitions, such a system as we are discussing would combine the best features of “hard” and “soft” video.

In addition to the advantages touched on for users, such a system would also have benefits from a development point of view. Much of the code base of applications in each of the categories denoted above would, by necessity, be very similar. For example, in a VJ application, source frames are read from the input clips, effects may be applied, and the output is displayed. For a video editing application, the process would likely be similar, except that each output frame is stored as a file. In a VJ application, obtaining an input frame, applying effects, and display must be highly optimised to function in real time. Optimisations here would also be beneficial to a video editing program, since it would translate into faster rendering, and due to the fact that even in a non-linear editing application, there is still the need to preview the end result and play it back in real-time. For a video programming environment, again, the same factors apply, except that in this case the control may come from outside the application, via a network interface rather than through human interaction with its graphical interface or keyboard. Improvements to the multitrack video editing side could well translate into a more feature rich, responsive and optimised real-time recording system.

The objective of this research, therefore, is to develop an application of this type.

2.2 REQUIREMENTS

It is useful at this point to examine the requirements for an application both in terms of specific features and more general attributes. Regarding specific features for our target set of users, data for their requirements are difficult to ascertain, as almost no formal study has been carried out in this area. In order to compile such a list, the author carried out the following procedure:

- **Data Collection**

The author collected data relating to the principal applications in the area of video processing, making use of online research and other sources. Material was gathered by means of Google searches, specialised forums, chats with application developers and online book research. From this, a list of representative software packages was selected. Applications which are no longer in active development were discarded from the list.

- **Benchmarking**

For each of the selected applications, the features and capabilities were noted, and a cross-reference of comparative features versus applications was created. The features were discovered by means of the application websites, reviews and blogs, enquiries made to users and in some cases observing a demo of the software running.

- **Selection of Features / Requirements**

The author derived a list of required features for Experimental VJs, based on reading the literature, discussions with specialists (for example at conferences and seminars), feedback from users, and his own personal experience as an Experimental VJ. The importance of the selected features is validated in Chapter 7.

With regard to **non-specific** features, this is discussed in more detail in section 5.2. The main elements are feature completeness, flexibility of design and implementation, a high level of performance/responsiveness, be easy to use and intuitive, and to demonstrate stability of operation.

In our results we show that LiVES does indeed meet all of these requirements, providing and in many cases expanding on the basic list of specific requirements, and scoring highly when rated on non-specific elements.

3 **AN OVERVIEW OF VIDEO PROCESSING APPLICATIONS**

The vast majority of video processing applications fall into two major categories and one minor category. The major categories are VJ applications (for working with real-time video) and video editing applications (for working with non-linear video). The minor category contains tools which can control video processing (playback, conversion, display, etc) programmatically. We will look at each of these areas in turn, focusing on popular non-academic works (since very few of these types of applications have been developed purely for academic research).

3.1 VJ APPLICATIONS

VJ (Video Jockey) applications are generally used to provide real time video for shows and other events; frequently the video is synchronised with music. The video inputs may come from various sources including pre-recorded clips, camera inputs, generated video and text overlays. Effects are often applied to the images and streams may be mixed together in a variety of ways.

The earliest VJ setups used purely analogue video, mixing television streams via a video mixing box. Most modern equipment is now completely digital.

One of the few academic works which relates to this area, Jácome (2007), presents an overview of the history of visual performance - from its conception in ancient times, when for example, firework displays and shadow theatre were popular forms of visual entertainment - up to the present day.

In the twentieth century there was a key development in the field of visual entertainment. The advent of the home computer for the first time placed the ability to create interesting and entertaining digital video effects in the hands of the everyday user. One of the first tools to take advantage of this was Video Toaster[3] which was first released in 1990.

[3]A look at the Video Toaster for the Amiga computer, 1990. Video clip. Available at: <http://www.youtube.com/watch?v=zyGCYoZ5Nlk>. Accessed: 13 Jun. 2013.

> *The NewTek Video Toaster was a combination of hardware and software for the editing and production of standard-definition video in NTSC, PAL, and resolution-independent formats...It comprises various tools for video switching, chroma-keying, character generation, animation, and image manipulation*.[4]

The earliest versions were produced for the popular and affordable Commodore Amiga home computer; the Video Toaster product sold for $2399. Due to the relatively low cost and wide range of features, the product was considered revolutionary at the time.

In 1989, Fujitsu began producing a computer for the Japanese market called “FM Towns”[5]. The machine was equipped with graphical capabilities which were extremely advanced for the time – it supported 24 bit colour at a resolution of up to 1024x1024 pixels, and in addition had a unique feature: the ability to overlay two different display modes simultaneously. Experience with this machine during the 1990s led to some new developments in the field of accessible video manipulation – as an example, the hardware inspired Kentaro Fukuchi to begin creating the program effecTV[6] (FUKUCHI; MERTENS, 2004). The effecTV program was later ported to the Linux operating system, released as open source, and further developed. Due to the availability of the code and the clever programming of the effects, these effects have in turn been adapted for many other systems.

These innovations and others contributed to a growing popularity of D-I-Y video performance, using off-the-shelf hardware and homegrown software.

In parallel with this, the 1980s and 1990s witnessed a new movement: the so-called “demoscene”. Participants within the demoscene sometimes worked alone, but more frequently in groups; but whatever the case, the goal was always the same – to produce ever more spectacular “demos”, demonstrations of skill at programming; creating programs which produced entertaining video and audio, using whatever hardware was at hand. The demoscene helped to further popularise the idea that audio, and in particular, video, could serve a purpose for entertainment – not only in the traditional cinematic sense, but rather by playing with the parameters of the video itself – overlaying one video clip on another, warping the video “screen”, altering the viewer's perspective, changing or rotating the colours and so on. It is

[4]WIKIPEDIA, Video Toaster. Available at: <http://en.wikipedia.org/wiki/Video_Toaster>. Accessed: 13 Jun. 2013.
[5]ONLINE article about FM Towns. Available at: <http://www.giantbomb.com/fm-towns/3045-108/>. Accessed: 13 Jun. 2013.
[6]ABOUT EffecTV. Available at: <http://fukuchi.org/research/effectv/>. Accessed: 13 Jun. 2013.

these ideas which have inspired modern VJs, and the developers of software of which they make use.

Most people are familiar with the term DJ – Disk Jockey, a person who plays and mixes music for performance events. A VJ – *Video Jockey* - is the visual equivalent of this – a VJ will play and mix visual clips and effects. In general, this is done to accompany music – played either by a DJ or by live musicians. Some VJs also operate as a DJ – providing both visual effects and audio, although this is not always the case.

Over time VJs have developed some standard techniques for providing interesting and entertaining results. These techniques can include:

- Playing video at varying rates – slow motion, fast motion, reverse playback can all be used.
- "Scratching" or jumping the video frames around – scratching is analogous to the technique of the same name used by DJs, meaning rapid backwards and forwards movements of the source(s), in this case video frames. Jumping the video frames around can also be used – for example one playback technique offered by some video tools is "nervous" mode, whereby the frames are skipped around by a few frames ahead or a few frames backwards in a random fashion.
- Mixing or overlaying two or more video sources may also be done. A variety of methods can be used here – the screen may be divided into geometrically distinct areas with each area showing a different video source. Alternately the video sources can be mixed, either by averaging the colours, or by using some other function – multiplying, dividing, or subtracting for example. In other cases some areas of one clip may be made transparent so that areas from a clip "behind" it are visible there.
- Applying one or more visual effects to the video – for example, altering the colours or mirroring one part of the screen to another. Effects are used very commonly by VJs to increase the visual attraction of the images, to create strobe-like effects and so on.
- Overlaying text or other symbols on the video.

All of these techniques may be combined together or may be used individually. Often the VJ attempts to synchronise changes in the video to coincide with music which is playing, in order to create a qualitatively more harmonious sensation for the audience.

Video may come from a variety of sources – some from pre-recorded material, some from video cameras, and other material may be generated dynamically. Quite often, the VJ tries as much as possible to make the source material relevant to the music being accompanied. This can affect the choice of the source material. The VJ can even expand on the themes of the music, thus enhancing the experience for the audience and increasing the appreciation of the musicians. For more abstract music, the emphasis may fall more on dynamically generated material, and synchronising the video changes in time with the beats of the sound may be more important here. In other cases the video may be the main attraction – reversing the role for the music, or there may be no audio at all.

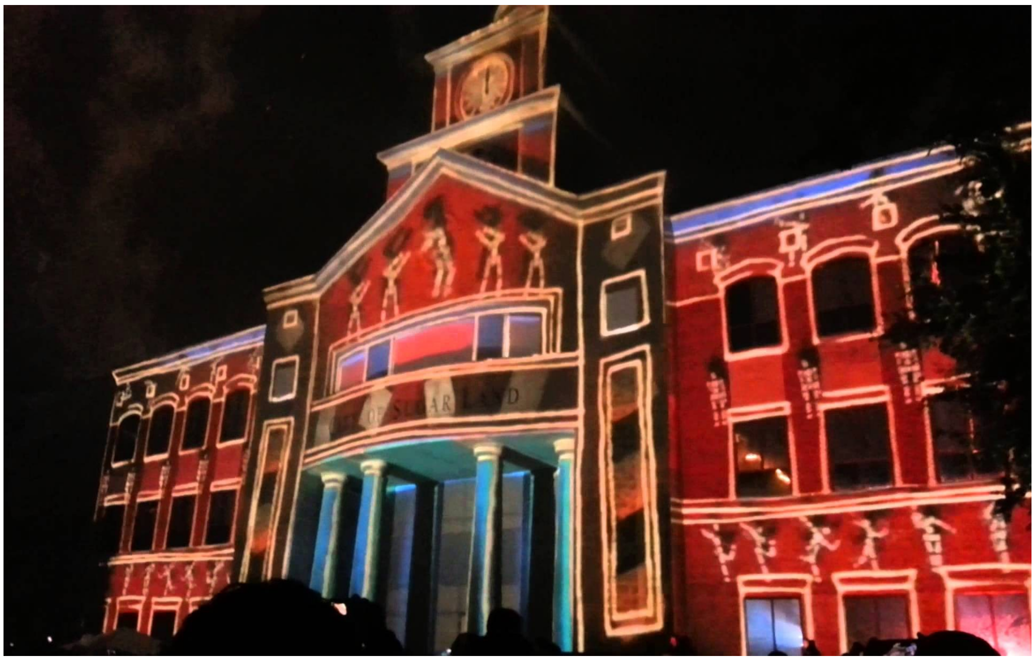
Comparing VJs to DJs, Spinrad (2005) explains:

> *But there's another side of DJing, its cultural collage—selecting, mixing, and juxtaposing music and other sounds to draw parallels and distinctions among the sampled components. And for this creative intent, VJing blows DJing away completely. The VJ's palette is far broader, encompassing any imagery at all, recognizable or obscure, abstract or representational, iconic or ambiguous. The elements can also be combined more freely, anything mixed with anything, without worrying about key, or beats per minute, or whether they sound good together. The dissonances a VJ creates are cognitive, not musical.*
>
> *(SPINRAD, 2005, p. 14).*

The field in which VJs operate is a very competitive and dynamic one. In order to compete more effectively and to provide a better experience for the audience, new techniques are constantly being developed. For example, it is usual for VJs to project onto flat projector screens; however many VJs have experimented by changing this, for example, projecting on water screens produced by high pressure jets. This produces the effect of a screen floating in mid-air, and it can be viewed equally well from both front and back.

A recent development in this field is the technique of **video mapping**. This technique is frequently used to project onto the surface of buildings outdoors (Figure 1), but it may also be employed indoors. Here the projection surface is not a flat screen but maybe bumpy, and there may be features such as doors, windows and balconies visible. Video mapping takes advantage of this by enabling the VJ to make use of the existing features as part of the projection. The VJ first maps the projection surface, noting the features and their positions and sizes, as well as the overall shape of the target. These features are then incorporated into the video projection - for instance a different image could be projected around the windows versus the rest of the building.

Figure 1: An example of video mapping projection



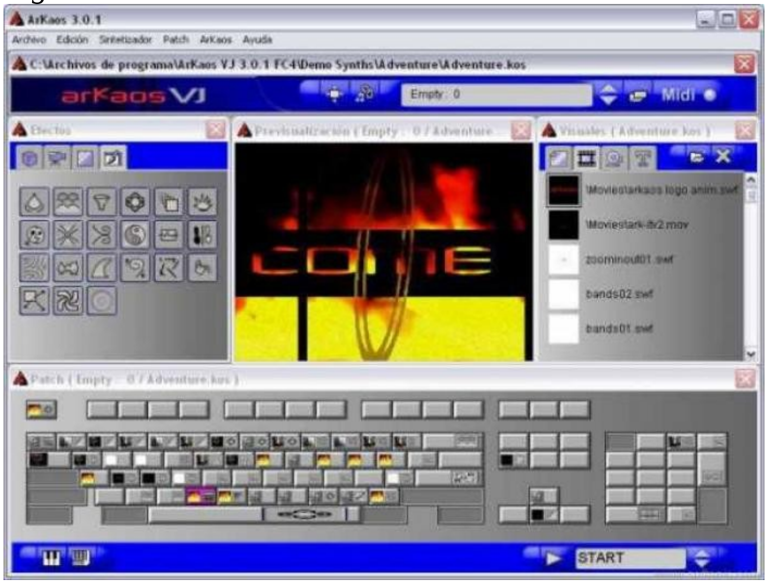
Source: youtube.com

In view of this changing environment, a great deal of experimentation is involved in the art of VJing. The term “Experimental VJ” could be applied to a great many individuals and groups operating within this field. VJ activities are by their very nature, experimental. This differs from more conventional computer activities, such as word processing or sending email, where the requirements are well established, and not subject to rapid change. We will return to this topic later on.

3.1.1 **Example applications**

Today, there are a few programs designed specifically for the VJ market – here we will look at three examples of these - Arkaos, Resolume, and the free software program Veejay.

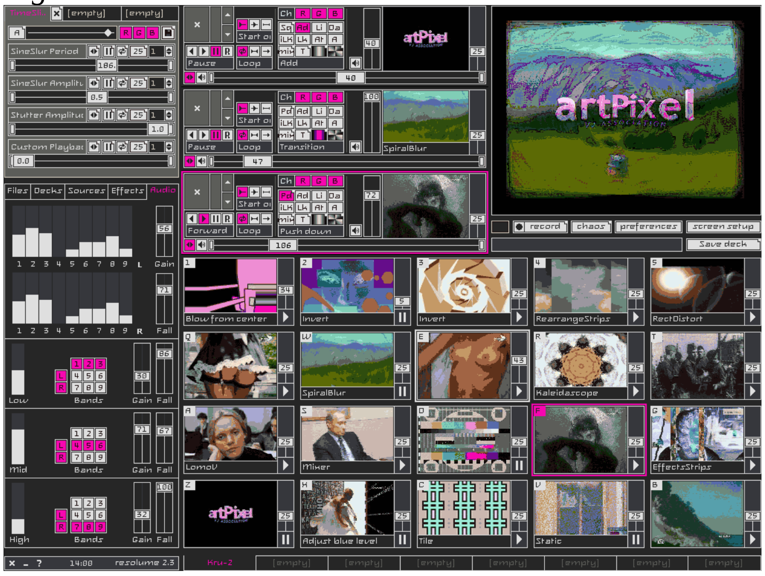
Arkaos[7]

Arkaos GrandVJ (Figure 2) is a well known professional VJ application. It supports features such as MIDI mapping, OSC control, video generators and multiple monitors. It runs under Windows and Mac OS X, and has a price of $399 USD.	*Figure 2: Arkaos GrandVJ*  ArKaos 3.0.1 Archivo Edición Sintetizador Patch ArKaos Ayuda C:\Archivos de programa\ArKaos VJ 3.0.1 FC4\Demo Synths\Adventure\Adventure.kos arKaos VJ Empty: 0 Midi Efectos Previsualización (Empty : 0 / Adventure Visuales (Adventure.kos) \Movies\arkaos logo anim.swf \Movies\ark-itv2.mov zoominout01.swf bands02.swf bands01.swf Patch (Empty : 0 / Adventure.kos) START *Source: softonic.com.br*

The basic idea in Arkaos is the use of *cells*. A cell can be mapped to a video clip, camera input, or an effect, and then subsequently linked to a key on the computer keyboard. Cells can also be assigned an area on the screen in terms of x and y coordinates. In “mixer mode”, Arkaos allows up to 8 layers of video to be combined onto a single screen. In “synth mode”, the cells can be played back by pressing various keys on the keyboard. Arkaos also has a MIDI (Appendix A – Definitions) learner interface which can be used to link keys on the keyboard to the knobs, sliders, keys and other controls on a MIDI controller. Effects can be driven by audio input, and it is possible to control the application using OSC (Appendix A – Definitions).

Resolume[8]

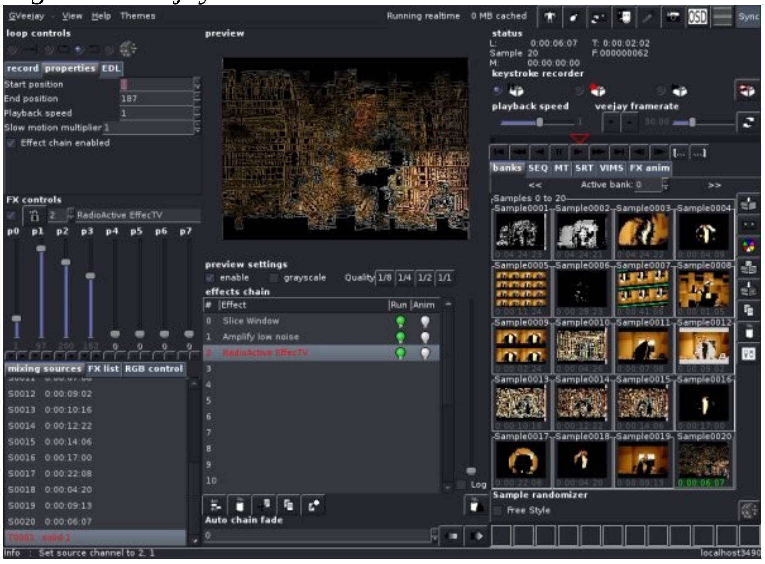
[7]ABOUT Arkaos GrandVJ. Available at: <http://vj-dj.arkaos.net/grandvj/about>. Accessed: 13 Jun. 2013.
[8]RESOLUME website. Available at: <http://resolume.com/>. Accessed: 13 Jun. 2013.

Resolume Avenue VJ software (Figure 3) is another popular VJ tool. Like Arkaos, it runs on Windows and Mac OSX, but is not supported on Linux. It has a price tag of EUR 299.00 for a single computer.	*Figure 3: Resolume*  *Source: youtube.com*

Like Arkaos, Resolume can mix multiple layers of video, it can be controlled by a MIDI controller, and via OSC. Resolume has a fairly limited set of input format options: Quicktime or AVI video formats, and PNG and JPEG images. However, this may be sufficient for some VJs.

Veejay[9]

[9]VEEJAY website. Available at: <http://www.veejayhq.net/>. Accessed: 13 Jun. 2013.

Veejay (Figure 4) is a real-time video sequencer and effects processor. It runs under the Linux operating system, and is Free / Open Source software, and is available at no cost. It supports features like OSC control, integration with Jack audio, and performance recording. The latest version of Veejay is 1.5.8 which was released in 2011.	*Figure 4: Veejay*  *Source: linuxlinks.com*

It is important to note that all of these applications share the same key features, and follow the similar paradigms of usage: a set of video clips and other sources (e.g. cameras) is imported, and these are commonly displayed as thumbnails within the interface. The user can select between the clip thumbnails, and also apply real-time effects. The output of the clips and effects is displayed, generally on a second monitor (or projector). Some programs also allow the operator's performance to be recorded.

3.2 VIDEO EDITING APPLICATIONS

The second category of video processing applications is *video editing*. Here the goal is not necessarily to display the output in real time but rather to produce one or more encoded clips. A timeline is used to lay out the pieces which will be joined together to produce the finished result. Effects and transitions may also be applied, but here there is less need for the effects to operate purely in real time.

The earliest editing systems were video tape based and hugely expensive:

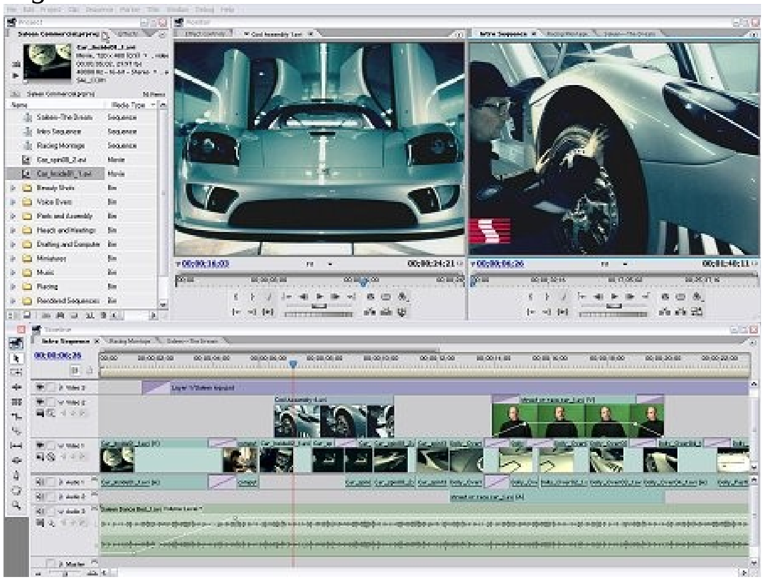
The 2" Quadraplex system cost so much that many television production facilities could only afford a single unit and editing was a highly involved process requiring special training.[10]

The advent of home computers again put these kinds of tools in the hands of ordinary users, without the need for any specialised hardware, and today there are dozens of applications dedicated to this field of use.

3.2.1 **Example applications**

Below we list some examples of video editing applications, although many more exist.

Adobe Premiere[11]

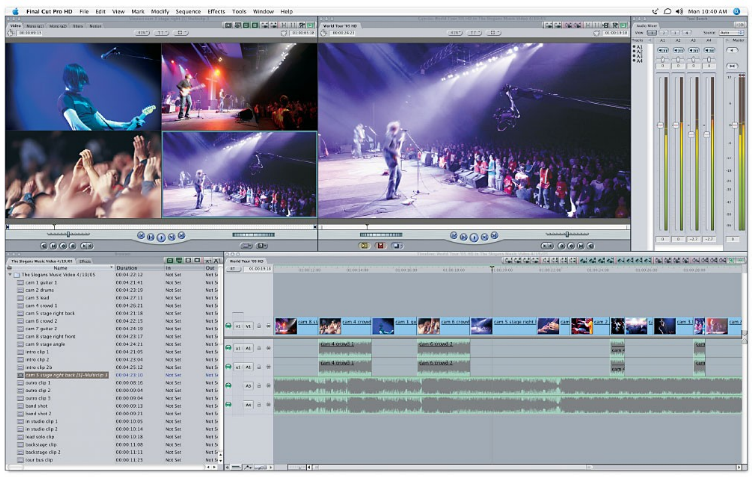
Adobe Premiere (Figure 5) is a popular commercial video editing application. It has advanced features such as a multitrack timeline, subtitling, and broad format support. It runs on Windows and Mac OSX. It is available only on a subscription basis, which costs $19.99 per month for the basic plan.	*Figure 5: Adobe Premiere*  *Source: adobetrainingni.com*

The building blocks of movies in Premiere Pro are *sequences*, which appear as tabs on

[10]WIKIPEDIA, Video Editing. Available at: <http://en.wikipedia.org/wiki/Video_editing>. Accessed: 13 Jun. 2013.
[11]PRODUCT page for Adobe Premiere. Available at: <http://www.adobe.com/products/premiere.html>. Accessed: 13 Jun. 2013.

the timeline. These sequences can be moved around, adjusted and trimmed, before the final mix is rendered. As with almost all video editing applications, the timeline is arranged horizontally in terms of time, with a series of tracks arranged vertically. During playback, the play cursor moves along the timeline, and depending on the sequence of effects applied, one or more of the clips at that point in the timeline is shown in the preview window. In addition to video tracks there are audio tracks – the audio may be either attached to a particular video clip, or it may be set as a separate backing audio track. Once the layout on the timeline is ready, the layout can be rendered (written to actual frames) and encoded to create a completed video file.

Final Cut[12]

Final Cut (Figure 6) is another popular video editing application. Like Premiere, it has an advanced multitrack editing interface, but does not appear to support individual clip editing. The program runs exclusively under Mac OSX, and has a price tag of $299.99.	*Figure 6: Final Cut Pro*  *Source: alltechnews.com*

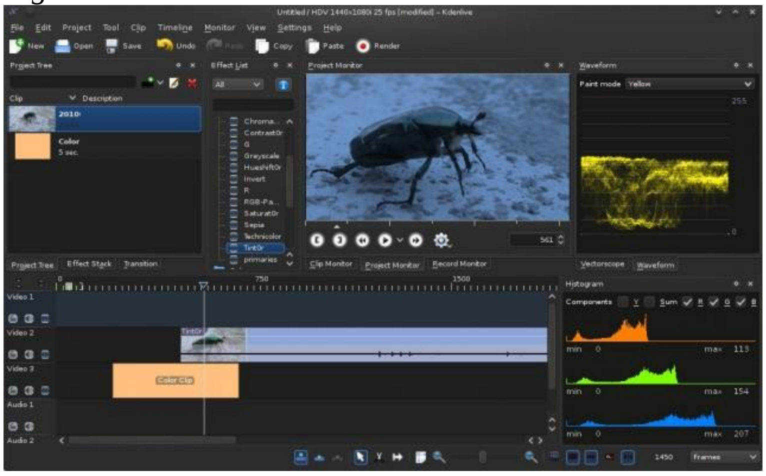
Final Cut Pro (FCP) has extensive media management features. Source clips can be arranged by metadata, for example – number of people in the shot, long, medium or short range, and so on. Keywords can also be applied to clips, or parts of clips, even before importing them, so it is a good tool for managing a media collection. These features are mainly important for those working in professional video editing studios, who may have many thousands of source clips to manage.

[12]PRODUCT page for Final Cut Pro. Available at: <http://www.apple.com/finalcutpro/>. Accessed: 13 Jun. 2013.

FCP uses a so-called “magnetic timeline”, which means that when any clip is moved around on the timeline, other clips will automatically move out of the way, move to higher or lower tracks and so on. This is a useful feature in some instances, but in other cases it may get in the way of what the user wishes to achieve.

Performance of FCP is another of its touted features - owing to the fact that the program is a native Apple application, and thus can make comprehensive use of the machine hardware.

Kdenlive[13]

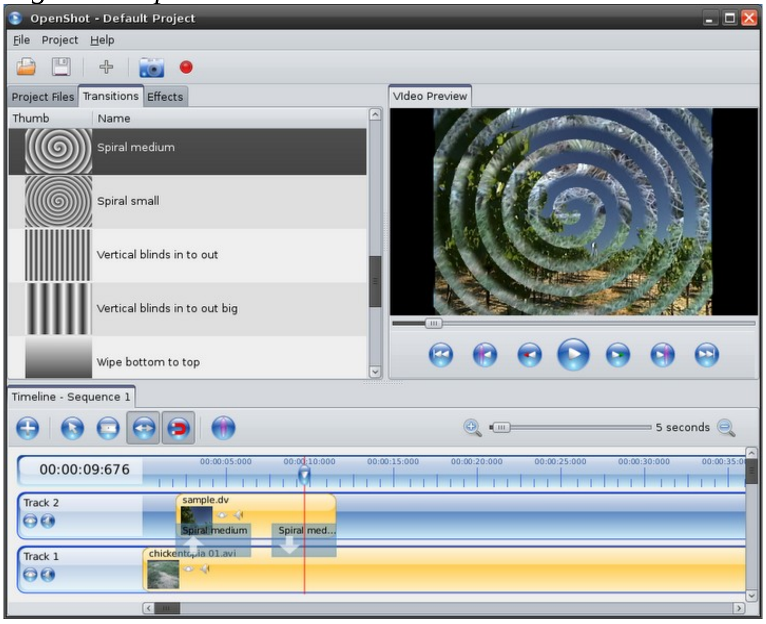
Kdenlive (Figure 7) is a Free Software video editing application for Linux and Mac OSX. It supports such features as camera inputs, frei0r effects compatibility, and a multitrack editing interface. It also has some basic clip editing facilities. It is available free of charge.	*Figure 7: Kdenlive*  *Source: ostree.org*

OpenShot[14]

[13]KDENLIVE features page. Available at: <http://www.kdenlive.org/features>. Accessed: 13 Jun. 2013.
[14]OPENSHOT website. Available at: <http://www.openshot.org>. Accessed: 13 Jun. 2013.

OpenShot (Figure 8) is another Free Software video editing application for Linux. It has a multitrack timeline, support for frei0r effects, and a fairly basic clip editor. It is available free of charge.

Figure 8: OpenShot



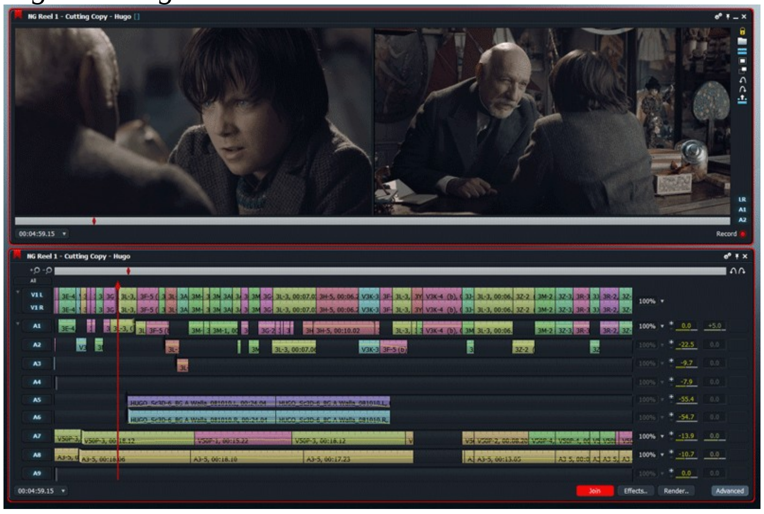
Source: ubuntugeek.com

Lightworks[15]

The Lightworks video editor (Figure 9) is available for Windows and Mac OSX, with a beta version available for Linux. At the present time, the source code is not available. It has a multitrack timeline, and support for dual monitors.

Lightworks Pro costs $60.

Figure 9: Lightworks



Source: editshare.com

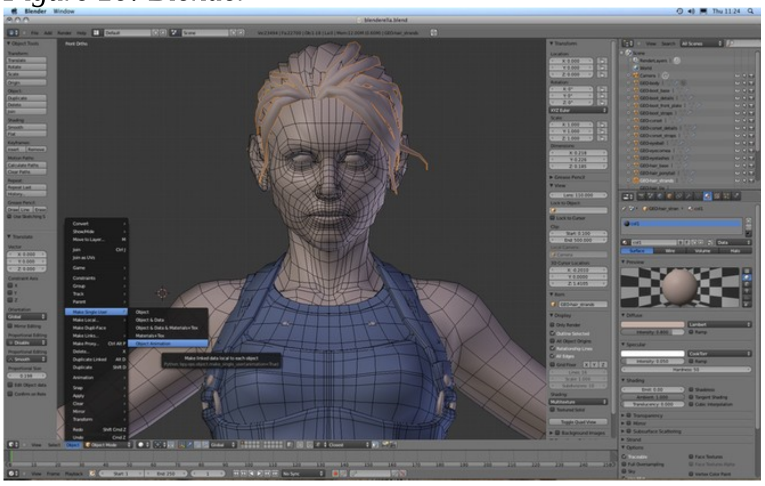
The applications mentioned above have largely similar features and paradigm of

[15]LIGHTWORKS website. Available at: <http://www.lwks.com/>. Accessed: 13 Jun. 2013.

usage. There is a timeline with multiple tracks of audio and video, onto which clips can be dragged or inserted. Effects and transitions between clips can also be placed on the timeline. The content of the timeline can be previewed and rendered to a new clip.

Blender[16], however, differs in its objectives and mode of use. Blender is actually a 3D rendering application which now has a multitrack timeline for rendering video. It has been used in several productions, for example: Elephants Dream[17], Sintel[18].

There is no mention of the video editing features on the Blender website - the features that are mentioned refer to it as a rendering/animation tool. Elsewhere, the video tool within Blender is referred to as the "Video Sequence Editor" (VSE). Rather than operating as a general purpose video editing tool, the VSE in Blender is designed specifically for use with the models which are generated within the application.

The Blender application (Figure 10) is available at no charge. The software will run on Windows, Mac OSX and Linux.	*Figure 10: Blender*  *Source: theguardian.com*

One advantage of Blender's Video Sequence Editor is its tight integration with the 3D rendering features of the software. For experienced users this means that it is easy to switch between modeling and movie production. The VSE follows the same interface design as the

[16]BLENDER website. Available at: <http://www.blender.org/>. Accessed: 13 Jun. 2013.
[17]ELEPHANTS dream. Video production. Available at: <http://www.elephantsdream.org>. Accessed: 13 Jun. 2013.
[18]SINTEL. Video production. Available at: <http://www.youtube.com/watch?v=eRsGyueVLvQ>. Accessed: 13 Jun. 2013.

rest of the Blender application (Figure 11), meaning that it would benefit mostly those who are already familiar with the other modes. Those users who are not familiar with Blender, however, may find it difficult to navigate around.

Figure 11: The Video Sequence Editor in Blender

Source: freesoftwaremagazine.com

3.3 VIDEO PROGRAMMING ENVIRONMENTS

Working with video programming environments is a still more recent development than working with video presentation or with video editing, since the tools needed for the former are totally reliant on computer technology. Such tools first appeared in the early demoscene days. Demoscene coders made extensive use of procedural video programming languages in order to fit within the hardware limitations of the time. Using these kinds of tools, the production, treatment and output of video may be controlled programmatically - perhaps by a script or by another application.

In addition, these types of tools may allow data to be processed, combine data with audiovisual material, or else extract data from audio or video. Some examples of this type of tool include Pure Data, Isadora, FreeJ, and the programming language named Processing.

Such tools are frequently used for artistic installations, but may have different uses – for example for static or dynamic image analysis.

3.3.1 **Example applications**

Again, there are quite a few applications which have been produced for this purpose. Here we list some of the more commonly recognised ones.

Pure Data[19] / MAX/MSP[20]

Pure Data (PD) is an Open Source visual programming language. It does not have a fixed graphical interface as such – rather, the user must connect together basic components (objects) in the interface to create what are termed “patches” (example, Figure 12). The program has a steep learning curve, generally requiring a good deal of study before one can go on to become proficient with it. A large number of components and patches have been created which allows for for a great deal of flexibility. However these components must first be studied, understood and connected together in specific ways.

[19]PURE Data website. Available at: <http://puredata.info/>. Accessed: 13 Jun. 2013.
[20]MAX/MSP website. Available at: <http://cycling74.com/>. Accessed: 13 Jun. 2013.

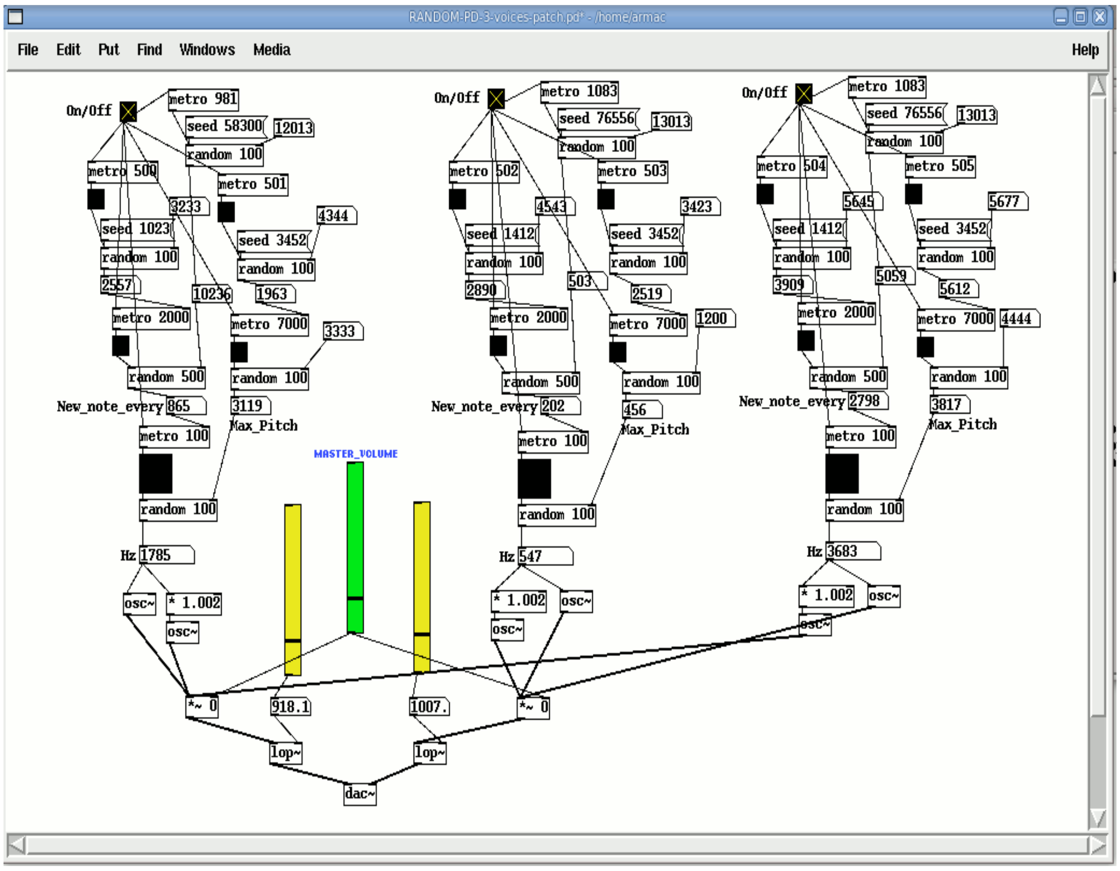


Figure 12: An example of a patch in Pure Data

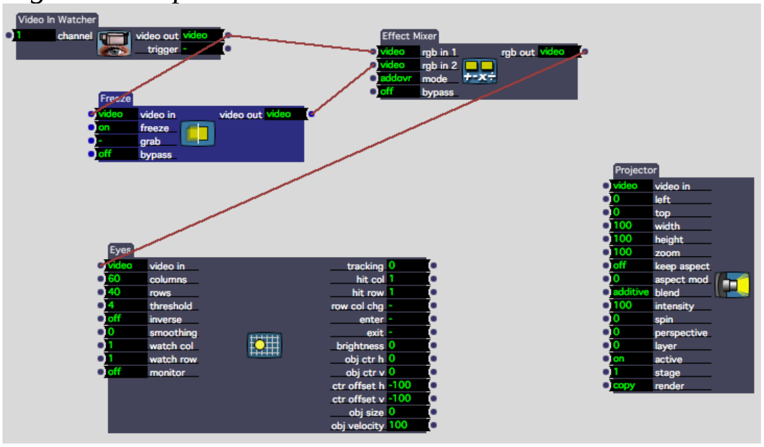
Source: noisefloor.org

The name “patch” is generally understood to be an analogy with early analogue audio synthesisers. The operator would modify the synth by connecting (or “patching”) the elements of the instrument together physically, using wires (“patch” cords).

PD was originally developed by Miller Puckette in the 1990s. It is based on his original MAX program (which in turn has now developed into MAX/MSP – a commercial version of the original MAX). PD is an example of what is known as a data-flow programming language. It is a programming language only in the sense that it can be used to construct applications. In a standard programming language, the flow of operation proceeds in one direction only – these languages can be represented one dimensionally. Visual programming languages take this a step further and add more dimensions to the program flow.

Video capability has been added to PD using a set of externals (libraries) referred to as GEM (Graphics Environment for Multimedia). GEM provides support for many objects, such as polygon graphics, lighting, texture mapping, image processing, and camera motion. Other similar extensions include PDP, PiDiPi, Framestein and GridFlow. An equivalent framework for MAX/MSP is Jitter.

Isadora[21]

Isadora (Figure 13) is a video processing environment with a graphical interface. Elements called "actors" can be arranged in patches in a similar fashion to Pure Data. Output can be displayed on up to six different monitors, and performances can be recorded and encoded to the Quicktime format. Isadora is available for a standard price of $350. It will run on Mac OSX or Windows.	*Figure 13: A patch in Isadora*  Video In Watcher channel video out trigger Effect Mixer rgb in 1 rgb in 2 mode bypass rgb out Freeze video in freeze grab bypass video out Projector video in left top width height zoom keep aspect aspect mod blend intensity spin perspective layer active stage render Eyes video in columns rows threshold inverse smoothing watch col watch row monitor tracking hit col hit row row col chg enter exit brightness obj ctr h obj ctr v ctr offset h ctr offset v obj size obj velocity *Source: vjskulpture.wordpress.com*

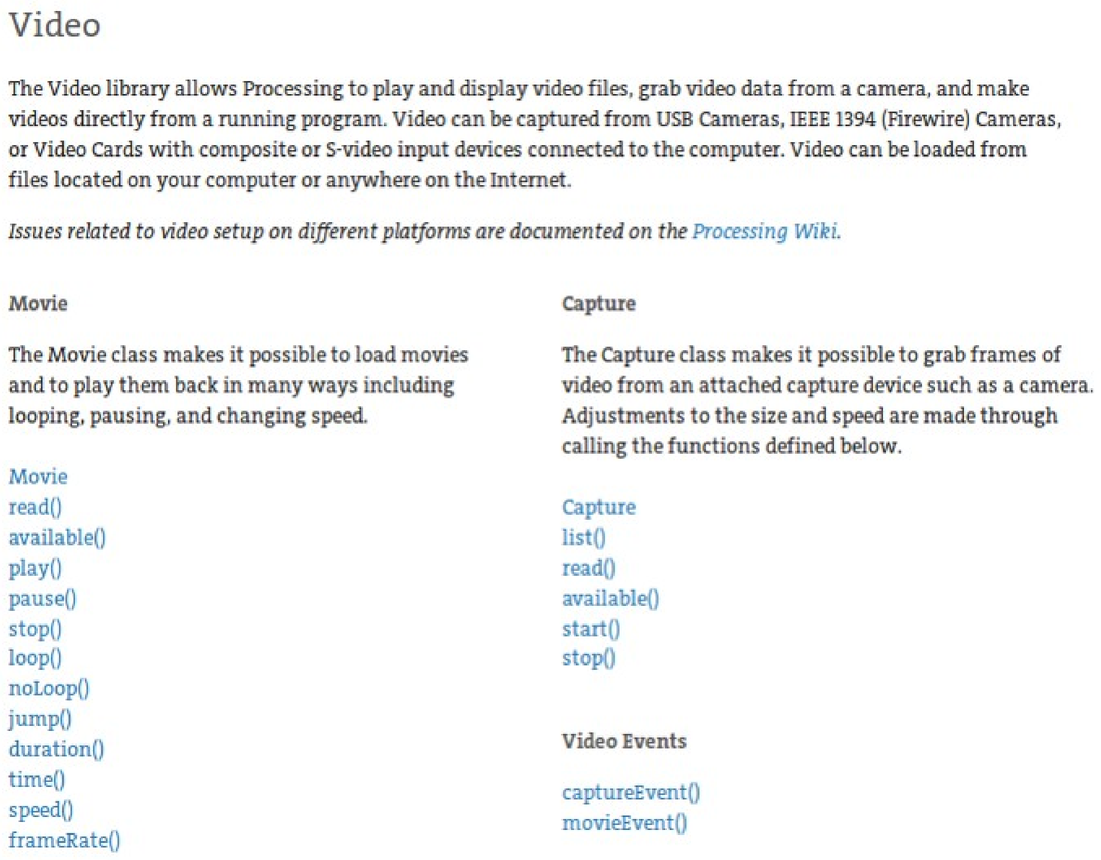
Isadora provides interactive control over digital media, with special emphasis on the real-time manipulation of digital video. An Isadora program is created by linking together graphically represented building blocks, each of which performs a specific function: playing or manipulating digital video, capturing live video, looking for MIDI input, controlling a DV (digital video format) camera, etc. By linking the modules together one can create complex interactive relationships that can be controlled in real time, either with the mouse and keyboard, or with external devices.

Notable features include the ability to composite numerous layers of video, a host of video effects possibilities, its ability to output to as many as six separate video projectors, and a powerful offering of input and output protocols. The latter (which includes OSC, MIDI, Serial, TCP/IP, and game controller devices) are essential for works that require real-time sensory input.

[21]ISADORA website. Available at: <http://troikatronix.com/>. Accessed: 13 Jun. 2013.

Processing[22]

Processing is a text based programming language which is often used for video programming, due to the inclusion of a dedicated video library[23] (Figure 14). Processing is free to download and use, and is open source. It is supported on the Windows, Mac OSX and Linux platforms.



Video

The Video library allows Processing to play and display video files, grab video data from a camera, and make videos directly from a running program. Video can be captured from USB Cameras, IEEE 1394 (Firewire) Cameras, or Video Cards with composite or S-video input devices connected to the computer. Video can be loaded from files located on your computer or anywhere on the Internet.

Issues related to video setup on different platforms are documented on the Processing Wiki.

Movie

The Movie class makes it possible to load movies and to play them back in many ways including looping, pausing, and changing speed.

Movie
read()
available()
play()
pause()
stop()
loop()
noLoop()
jump()
duration()
time()
speed()
frameRate()

Capture

The Capture class makes it possible to grab frames of video from an attached capture device such as a camera. Adjustments to the size and speed are made through calling the functions defined below.

Capture
list()
read()
available()
start()
stop()

Video Events

captureEvent()
movieEvent()

Figure 14: Video library API in Processing (screenshot from website)

Source: processing.org

FreeJ[24]

FreeJ is described by its creator as a “Free Vision Mixer”. It consists of a commandline based core, but a graphical interface exists for Mac OSX. FreeJ has an

[22]PROCESSING website. Available at: <http://www.processing.org>. Accessed: 13 Jun. 2013.
[23]Processing - video library API. Available at: <http://www.processing.org/reference/libraries/video/index.html>. Accessed: 13 Jun. 2013.
[24]FREEJ website. Available at: <http://freej.org/>. Accessed: 13 Jun. 2013.

interpreter which can run scripts written in JavaScript and it can also be used as a C++ library for inclusion in other programs. It is Free/Open Source software and will run on MacOSX and Linux. It can be downloaded and used free of charge. A sample JavaScript script for freeJ might appear as follows:

```
//simple script to test filter functionalities

img = new ImageLayer();
img.open("doc/ipernav.png");
img.activate(true);
img.start();
add_layer(img);
filt = new Filter("Distort0r");
img.add_filter(filt);

kbd = new KeyboardController();
register_controller(kbd);
kbd.released_q = function() { quit(); }
kbd.released_r = function() {
    if(reset("freej_equalizer.js")) {
        rem_controller(this);
        echo("reset ok");
    }
    return true;
}

bang = new TriggerController();
bang.frame = function() {

    a = 1/(rand()%200);
    b = 1/(rand()%200);
    filt.set_parameter("Frequency", a);
    filt.set_parameter("Amplitude", b);
}


register_controller(bang);
```

Using FreeJ one can overlay, mask, transform and filter multiple layers on the screen. There is no limit to the number of layers that can be mixed. Each layer can be video taken from different sources: movie files, webcams, TV cards, images, rendered text, flash animations, generated video, and effects from the *frei0r* (Appendix A – Definitions) plugin collection.

FreeJ has an asynchronous video rendering engine. It can be scripted using JavaScript syntax in an object oriented way, to control its operations through a procedural list of commands and actions.

3.4 ACADEMIC PROJECTS

Searches of Google were performed using keywords such as “user requirements video applications”. The author also searched on CiteSeerX[25] using keywords such as “video applications”, “user requirements video applications”. The lack of articles covering the area of user requirements for video applications indicates that very little scientific research has been devoted to this topic. For this reason, the majority of references for this dissertation consists of web documents, books and non-academic journals. However, a few academic projects have been produced in this area.

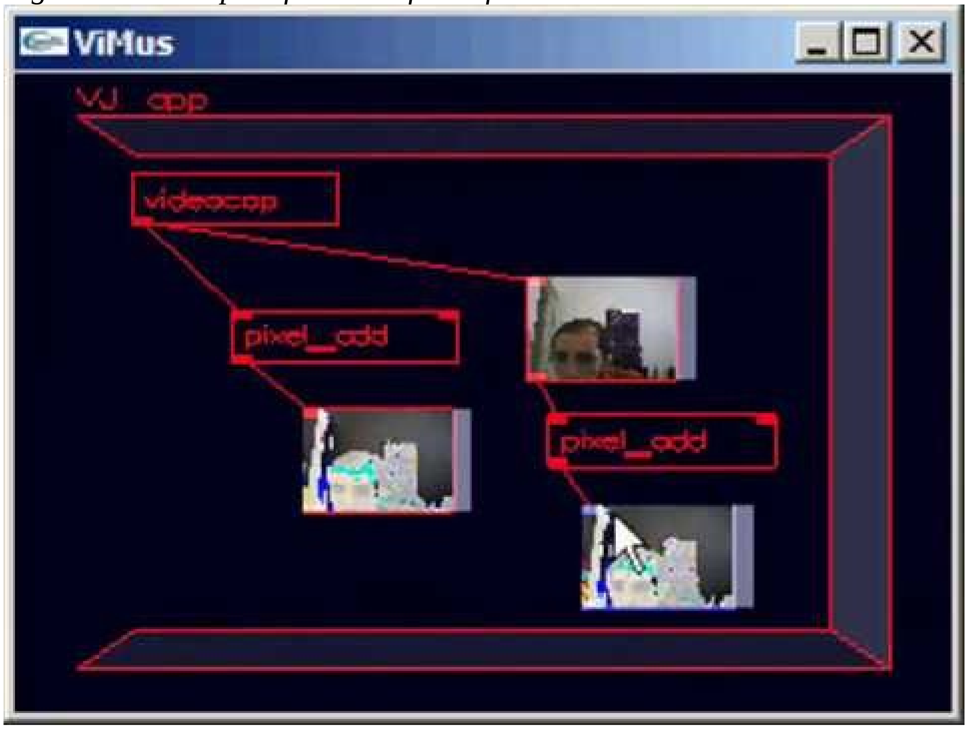
ViMus[26]

ViMus (short for Visual Music) is a project which was started by Jarbas Jácome in 2003, as part of a masters course in Computer Music/Computer Graphics.

ViMus is different in that it takes the two dimensional paradigm of most visual video programming environments such as PD and Isadora, and creates the illusion of a three dimensional interface (Figure 15). As well as setting out modules on a flat plane in the way these other frameworks allow, ViMus enables several of these layers to be stacked one on top of the other.

[25]CITESEERX website. Available at: <http://citeseerx.ist.psu.edu>. Accessed: 13 Jun. 2013.
[26]JÁCOME, J. Vimus. Available at: <http://jarbasjacome.wordpress.com/vimus/>. Accessed: 13 Jun. 2013.

Figure 15: Example of the interface of ViMus



Source: youtube.com

This allows far more flexibility in designing patches by providing a cleaner, clearer view for the user. The layers may be partitioned logically – for example, input layer, processing layer, and output layer. However, the main drawback is that most graphical interface toolkits are designed with the idea of a two dimensional paradigm – thus the development of the interface elements for a tool such as ViMus requires a great deal more effort. In addition, it may take some time for the user to become acquainted to working in this way, both in terms of control of the interface and conceptually.

The principle use of ViMus has been as a visual musical instrument, as the name suggests. Audio which is fed in to the system is analysed, and the results of the analysis are used to alter parameters of visual effects.

The sourcecode for ViMus was released in 2009 under a Free Software license. Despite this, the application remains little known.

OpenCV[27]

[27]OPENCV website. Available at: <http://opencv.org/>. Accessed: 13 Jun. 2013.

OpenCV (Open Source Computer Vision Library) is an open source computer vision and machine learning software library. OpenCV was built to provide a common infrastructure for computer vision applications and to accelerate the use of machine perception in the commercial products. Officially launched in 1999, the OpenCV project was initially an Intel Research initiative to advance CPU-intensive applications. The first alpha version of OpenCV was released to the public at the IEEE Conference on Computer Vision and Pattern Recognition in 2000. The main contributors to the project included a number of optimization experts in Intel Russia, as well as Intel’s Performance Library Team.

Application areas include: facial recognition, motion tracking, augmented reality, gesture recognition, and mobile robotics. It can also be used in machine learning software, for example for artificial neural networks, Bayes classification, and decision tree learning. OpenCV is licensed under the BSD license, and runs on several operating systems including Windows, Android, BSD, Linux and Mac OSX.

4 **METHODOLOGY**

In this section we explain the methodology used to develop an application in order to satisfy the key requirements of Experimental VJs.

Previous research by Boehm and Turner (2003) suggests that the best methodology to use would be *Agile* type software methods, as Table 1 highlights:

Table 1: Suitability of various development methods

Agile method	Plan-driven method	Formal methods
Low criticality	High criticality	Extreme criticality
Senior developers	Junior developers	Senior developers
Requirements change often	Requirements do not change often	Limited requirements, limited features
Small number of developers	Large number of developers	Requirements that can be modeled
Culture that responds to change	Culture that demands order	Extreme quality

Source: Boehm; Turner (2003)

The Agile software development process is based on an iterative approach. It is important to bear in mind that the Agile process is very different from plan-based and formal methods of software development. In each iteration, a set of requirements is created, a prototype is developed from the requirements, and the prototype is then validated. The results of the validation are used to generate the subsequent prototype, and the cycle is repeated. Intuitively, this was the method used to develop the LiVES application up to the current point. **For the purposes of this dissertation, the present cycle has been carried out in more depth than usual, with a somewhat deeper analysis of the current requirements and validation of the prototype.** This provides the basis of our current scientific research.

The method we use is composed of three phases: conceptual, development and validation. We begin with a little background for the conceptual phase, namely the analysis of video applications which was carried out as a part of this research. This was done in an intuitive / exploratory fashion, rather than using formal scientific techniques.

4.1 ANALYSIS OF VIDEO APPLICATIONS

A few months prior to writing this paper, the author performed searches on Google for topics such as “video applications”, “VJ applications”, and “video programming environments”. The top entries were selected as generally being an indication of popularity of packages. The results were then further filtered manually in order to create what the author considers a representative sample of current video applications.

The author then carried out a benchmark examination of the selected video software applications, collating a list of their attributes and features. The various applications were analysed by reading reviews and visiting the application websites. In some cases the author was able to observe users demonstrating some of the features of the software package.

Some anecdotal stories were recalled by the author regarding the history of video applications. The author is also frequently in contact with the developers of various Free Software video applications via mailing lists, email, and chat.

In order to further assist with the research for this paper, the author sent a questionnaire (shown in Appendix B) to users and potential users of video applications via the LiVES user mailing list and website. The participants were asked which features in a video application (in this instance only LiVES was referred to) were important to them. A total of fifteen responses were received during the short time allotted. A second part of the survey asked participants to rate the LiVES application in five non-technical areas. Finally, a third section asked what (if any) additional features the participants felt would be useful for future development of the LiVES application. The raw data are presented in Appendix C.

4.2 CONFERENCES AND SEMINARS

The author had the good fortune to be invited to participate in the Piksel Festival[28] held in Bergen, Norway, in the years 2003, 2004 and 2005. The festival began as an organised meeting of Free Software video application developers. There were numerous technical

[28]PIKSEL FESTIVAL. Bergen, Norway. Available at: <http://www.piksel.no/>. Accessed: 13 Jun. 2013.

discussions and hands-on coding sessions, mixed in with performances and demonstrations by the developers themselves. During these meetings, it became clear that there were three areas which would benefit from collaboration between the various projects: sending and receiving video in real time between applications; a common framework for video effects, and a common language for programmatically controlling applications.

Other attendees at the first meeting included developers of Veejay, Pure Data, FreeJ and effecTV. Obviously this part of the research was done much prior to the other conceptual phases.

4.3 CROSS-REFERENCE OF APPLICATIONS

As a result of this study, the author was able to compile a list of required features and create a cross reference with a selection of more commonly used video applications. Table 2, below, shows this information in tabular form. The requirements themselves are discussed in Chapter 5, however it should be noted that *none of the benchmarked applications by themselves* satisfy all of the potential needs of an Experimental VJ.

Table 2: Comparison of technical features in various video applications

Ap	LS	RE	EF	FC	JA	CR	OS	MJ	IF	OF	VR	CE	P	AC	TV	RP	V4	MM	ME	FW	AP	SS	TH
AR		*					*	M	*		*		$$		*		(1)	*		?	*		*
RE		*			(2)		*	M			?		$$		?	?	(1)	*		?	*		*
VJ	*	*	*	*	*		*	?			*	*	*	*	*	*	(3)	*		?			*
AP			*			?			*	*		*	$$		?			?	*	*	?	*	*
FC			*			?			*	*			$$					?	*	*	*	*	*
KL	*		?	*		?			*	*			*	*					*		*	?	*
OS	*		?	*		?			*	*	*	*	*	*					*		*	*	*
LW	(4)		?			*			*	*	*		$ (4)					*	*	*	*		*
BL	*		*			?			?	?	?		*	*				?	*				*
PD	*	*	?	*	*	?	*	*	?	?	*		*	*	?		*	?		?	*	?	
IS		*	?			?		*	?		*		$$		?	*		*		?			*

Source: Finch (2013)

	Notes		Notes
(1)	Arkaos and Resolume support *Syphon* for OSX, which appears to be similar to v4l.	(2)	Resolume supports SMTP timecodes which are similar to jack transport.
(3)	Veejay supports v4l *input* only	(4)	A limited beta of Lightworks is available for Linux

Key	Meaning	Key	Meaning
*	Feature is known to be supported	<space>	Feature is known to be unsupported
?	No information about feature		
$	Price < $100	$$	Price >= $100 (per year)
AR	Arkaos GrandVJ	LS	Linux Support
RE	Resolume	RE	Real-time effects
VJ	Veejay	EF	Edit individual frames
AP	Adobe Premiere	FC	Frei0r compatibility
FC	Final Cut	JA	Jack audio integration
KL	Kdenlive	CR	Crash recovery
OS (Left)	OpenShot	OS (Top)	OSC support
LW	Lightworks	MJ	MIDI / joystick control
BL	Blender	IF	Wide range of input formats
PD	Pure Data	OF	Range of high quality output formats (encoded)
IS	Isadora	VR	Playback at variable rates / reverse playback
		CE	Clip editor
		P	Price (Y = free)
		AC	Accessibility of code
		TV	TV Card / web-cam input
		RP	Ability to record performances
		V4	Video for Linux (v4l2) support
		MM	Multiple monitor support
		ME	Multitrack editor
		FW	Firewire support
		AP	Audio plugins / filters
		SS	Subtitle / captioning support
		TH	Themes / user friendly GUI

[29]

It is also clear from the list of more common applications given above, that Linux users are poorly served, especially in the area of VJ applications and video programming environments (PD and Veejay being the obvious exceptions). Although Linux is still considered a minority operating system, it nevertheless represents an important segment of the community.

4.4 METHODOLOGY - FLOWCHART

The flowchart at the end of this section (Figure 16) shows the methodology used to produce this dissertation. The phases on the flowchart correspond to the following:

Phase 1: Conception

As mentioned in Chapter 2, the steps carried out for this phase consisted of:

Phase 1.1: Data Collection

The author collected data relating to the principal applications in the area of video processing, making use of online research and other sources. Material was gathered by means of Google searches, specialised forums, chats with application developers and online book research. From this, a list of representative software packages was selected. Applications which are no longer in active development were discarded from the list.

[29] Sources for the table:
<http://vj-dj.arkaos.net/grandvj/detailed-features>
<http://resolume.com/software/>
<http://veejay.sourceforge.net/veejay-HOWTO.html#2.1>
<http://www.adobe.com/products/premiere/features.html>
<http://www.apple.com/finalcutpro/what-is/#revolutionary>
<http://www.kdenlive.org/features>
<http://www.openshot.org/features/>
<http://www.lwks.com/index.php?option=com_content&view=article&id=24&Itemid=179>
<http://www.blender.org/features-gallery/features/>
<http://puredata.info/>
<http://troikatronix.com/isadora/features/>

Phase 1.2: Benchmarking

For each of the selected applications, the features and capabilities were noted, and a cross-reference of comparative features versus applications was created. The features were discovered by means of the application websites, reviews and blogs, enquiries made to users and in some cases observing a demo of the software running.

Phase 1.3: Selection of Features / Requirements

The author derived a list of required features for Experimental VJs, based on reading the literature, discussions with specialists (for example at conferences and seminars), feedback from users, and his own personal experience as an Experimental VJ. The importance of the selected features is validated in Chapter 7.

Phase 2: The Implementation of LiVES

Phase 2.1: Interface Definition

During the development of the application (increasingly so in the more recent development), the author attempted to follow the guidelines set out in the Gnome Human Interface Guidelines[30]. The salient points of these guidelines which apply to LiVES are:

- **Design for people –** know who your user base are and what you want to enable them to do.
- **Don't limit your user base –** make the application accessible – for example, provide keyboard commands for all menu options to cater for disabled users who may have limited movements. Consider internationalisation and localisation issues.
- **Create a Match Between Your Application and the Real World** - Always use words, phrases, and concepts that are familiar to the user rather than terms from the underlying system. Use terms that relate to the user's knowledge of the tasks your application supports.

[30]GNOME Human Interface Guidelines. Available at: <https://developer.gnome.org/hig-book/>. Accessed: 13 Jun. 2013.

- **Make your Application Consistent** - Make your application consistent with itself and with other applications, in both its appearance and its behavior.
- **Keep the User Informed** - Always let the user know what is happening in your application by using appropriate feedback at an appropriate time. When the user performs an action, provide feedback to indicate that the system has received the input and is operating on it.
- **Keep it Simple and Pretty** - Your application should enable the user to concentrate on the task at hand. So, design your application to show only useful and relevant information and interface elements.
- **Put the User in Control** - A user should always feel in control, able to do what they want when they want.
- **Forgive the User** - allow users to quickly undo the results of their actions. If an action is very dangerous, and there is no way to undo the result, warn the user and ask for confirmation. In all cases, the user's work is sacrosanct. Nothing your application does should lose or destroy user's work without explicit user action.
- **Provide Direct Manipulation** - Wherever possible, allow users to act on objects and data directly, rather than through dialogs or explicit commands.
- In addition, there are general guidelines for different types of windows, alert dialogs, graphical widgets and so on.

Phase 2.2: Creation of the Architecture

The architecture of the system is an important stage in the development of software, where the main functional modules are designed, along with the interactions between them. A large part of the architectural design was determined by the choice of the GUI toolkit used primarily, GTK+[31]. The toolkit uses a system of graphical elements (widgets) which may be nested using container type widgets. The widgets are not only set up graphically but they can be made to respond to predefined events and signals by linking these signals to callback functions. For example, when a button is clicked, the application will respond by calling

[31]GTK+ website. Available at: <http://www.gtk.org>. Accessed: 13 Jun. 2013.

whatever function was linked to the click signal for that button. Occasionally the application will call the widget callback functions directly rather than via a signal from the GUI, for example in response to an OSC message.

A further design decision was to split the application into a front end (written in C), and a backend (written in Perl). The backend handles much of the interaction with the operating system and with external applications, and can be used in an asynchronous fashion. This allows the front end to be more responsive and to a large extent independent of the underlying file system and operating system.

Another choice was to divide the software into a core application and plugins. There are several good design reasons for this. Firstly it provides a more or less stable interface for expanding the application. Code within the core can be altered and extended, but provided the plugin API is not changed (or at least remains backwards compatible), then plugin code should not be affected by these changes. This allows for parallel development of the software, one or more developers can be working on the core whilst others can be focused on plugins. In addition, plugin developers can remain concentrated within their area of knowledge – a developer working on an effect plugin does not need to concern themselves with the details of decoding various video formats, which can be done via a different type of plugin. The overriding principal in LiVES is to make plugins as simple to write as possible – the idea being to shift any complexity into the core. In this way the complex code only needs to be written once. Another advantage is that plugins can make use of specialised components like specific libraries or external binaries which may not generally be present on all machines. In this way, the user is not required to install these components in order to use the application, the plugin will simply not be loaded. This reduces the list of pre-requisites required to compile and run the application. However the user can later install the external components should they wish to make use of the corresponding plugins.

Regarding the architecture in general, the code should be well commented, to make it more maintainable. Some guidelines recommend one line in five should be a comment. The author considers this to be somewhat “overkill”, but nevertheless attempts to maintain a comment ratio of at least one line in ten.

Phase 2.3: Creation of the Framework

As discussed in section 5.2, below, an application which intends to combine the major features of a VJ application, video editing application, and video processing environment requires an extremely flexible internal framework. The evolution of this (WEED) is described in Chapter 7.

In addition to the general internal framework, the author devised a means for plugins to request parameters from the core application (host), and for the host to display these parameters graphically in a consistent yet flexible way to the user, and then return the values to the plugin. The author has named this system “RFX” (originally the initials stood for Rendered Effects (FX), since it was first envisaged for that type of plugin – however it is now employed by many different types of plugin, demonstrating its usefulness).

RFX, in summary, operates in the following manner:

- a function in the plugin returns a list of parameters, with some predefined attributes for each one. For example, an integer type parameter would have the attributes: name, label text, default value, minimum value and maximum value.
- The host calls the plugin function, receives this list, and dynamically generates an interface based on the parameter list. The host is free to do this in any way it chooses, the only stipulation is that parameter widgets are assumed by default to be placed vertically from top to bottom in order. Within LiVES this is done in a consistent way to make all plugin interfaces familiar to the user.
- In addition, the plugin may send extra hints to the host about how the parameter window should appear. For example, the plugin may hint that two or more parameters should be placed on the same horizontal row – related parameters might be width and height, x and y coordinates, for example. The plugin can also add explanatory labels. There is also the concept of “special widgets”, which may be linked to more than one parameter. An example of this would be a widget which displays the current frame, and allows the user to click inside it to select a point. The x and y coordinates of the point are linked to two numeric type parameters. Several types and subtypes of special widgets are defined in the RFX specification - this list may be expanded in future should the need arise. Note that these are only *hints* to the host, the plugin should be written so as not to be adversely affected if the host chooses to ignore them.

– After displaying the interface, the host allows the user to set the parameter values, then passes these back to the plugin.

The complete RFX Specification[32] contains some elements which are only applicable to rendered effect plugins. However the layout hinting part, along with the dynamic parameter interface generation within the core has been found useful for other types of plugin too. GUI interfaces for plugins have been discussed in the literature (MAYER, 2002).

It should also be noted that the frameworks were adjusted and updated during the development process in order to enhance their functionality, fix errors in the original spec., and to allow for unforeseen circumstances. The changes were kept as minimal as possible, and were carried out in such a way as to maintain backwards compatibility wherever possible.

Phase 2.4: Creation of the Application

The application was created, following the guidelines, architectural design and using the framework discussed previously. The software was developed using a Free Software license (GPL).

The advantages of a Free (Open) Source implementation and the process behind it have been described by many, including Đurković, Vuković and Raković (2008). Regarding the development process, they note:

- The Open Source model does not function well in the early stages, since many complex design decisions must be made, and spending too much time discussing these would likely impede the progress of the project. The initial prototype planning is generally not done publicly, but rather, is done by one individual or a small group. In this way, the design can also be kept consistent. However, this does imply that some decisions for the design must be postponed until later in the project.
- Open Source projects generally follow a model of prototype development, where the initial project gradually advances in iterations. This allows for greater flexibility, especially during the early development process.
- The code advances in small but constant changes, with fairly frequent version releases.

[32]FINCH, G. RFX specification. Available at: <http://svn.code.sf.net/p/lives/code/trunk/RFX/RFX.spec>. Accessed: 13 Jun. 2013.

As the project becomes more widely used, feedback received from experts and users can be employed to repair bugs and to extend the capabilities and feature set of the project.

- As the project expands, the rate of bug reports increases – many users can find many bugs. As these bugs are corrected, the software becomes more stable. At the same time, more users means more feature requests; satisfying these new requirements increases the capabilities of the software. However, the scope and the focus of the project must be maintained.
- The availability of the code allows participants with varying skills and experience to enter the development process, and to do so in a decentralised manner. In respect of this, it is essential that additions and amendments to the main code base be reviewed carefully by the core development team before they are included. This ensures that a high level of quality is maintained throughout the evolution of the application. The basic principle guiding this is a system of trust which arises naturally. More experienced and more talented code authors are granted a higher level of responsibility and authority in the project.

The above points are very important to keep in mind in order to fully appreciate the history and direction of the LiVES development process, and the philosophy behind it. The development method used is an Agile method: the overall direction is driven by the Open Source model.

For reference, Table 3 below provides a comparison of Proprietary and Open Source software development models.

Table 3: Comparison of Proprietary and Open Source development models

	Proprietary software	**Open Source software**
Resources	Known	**Unknown**
Planning Period	Whole project	**Step by step**
User	Customer who purchased the product	**Participant in software development**
Target	Fulfill the contract	**Solve the problem**
Discipline	Strong	**Weak**
Development	Secret, reliable	**Public**
Cooperation	Face to face	**Via the Internet**
Quality warranty	Management	**Competition**

Source: Raymond (1999)

The concurrent evolutionary model of both requirements and architecture as we see in LiVES has been nicknamed the *Twin Peaks* model (NUSEBEIH, 2001).

The GPL[33] license was selected for the LiVES code for several reasons:

– personal preference of the original developer

– the desire to contribute to the Free Software community by ensuring that the code is available to be reused in other GPL programs

– the wish to have all development carried out in the open, in order to encourage other contributors to advance the project

– the ability to make use of the existing body of GPL code in order to help accelerate the development of the software

Phase 2.5: Creation of the Plugins

The plugin interfaces (APIs) were designed and the plugins were created using the frameworks described above, where applicable. The plugins were developed using a Free Software license (GPL). The interface code between plugin and host has been developed

[33]GNU GPL license. Available at: <http://www.gnu.org/licenses/gpl.html>. Accessed: 13 Jun. 2013.

using the more permissive LGPL[34] license, which provides the additional option for closed source plugins to be used.

Phase 3: Validation of LiVES

There are two types of validation presented here for LiVES. The first is a functional validation. In other words, the system should satisfy all of the functional requirements. In section 7.1, we show a mapping between the obligatory requirements and the feature set of LiVES.

The second type of validation is non-functional. In order to obtain this information, some qualitative experiments were performed. A lesser emphasis was placed on statistical analysis as a means of validation of LiVES as a tool for Experimental VJs, owing to the small number of respondents in comparison to the number of users of the system. To compensate for this, other measures such as the number of users are provided to demonstrate this point. The following process was used for validation:

Phase 3.1: Questionnaires

Three sets of questions were presented to users via the LiVES mailing list and to visitors to the LiVES website. The first set asked respondents to mark which features from the feature list were important to them (in order to help gauge that the correct features had been selected), and to better understand their relative ranking. A second set of questions asked the respondents to rate the application in five non-technical categories. Finally, a third question asked users what, if any, new features they would like to see implemented in LiVES, in order to ascertain that no very important features had been omitted from the list.

Phase 3.2: Interviews

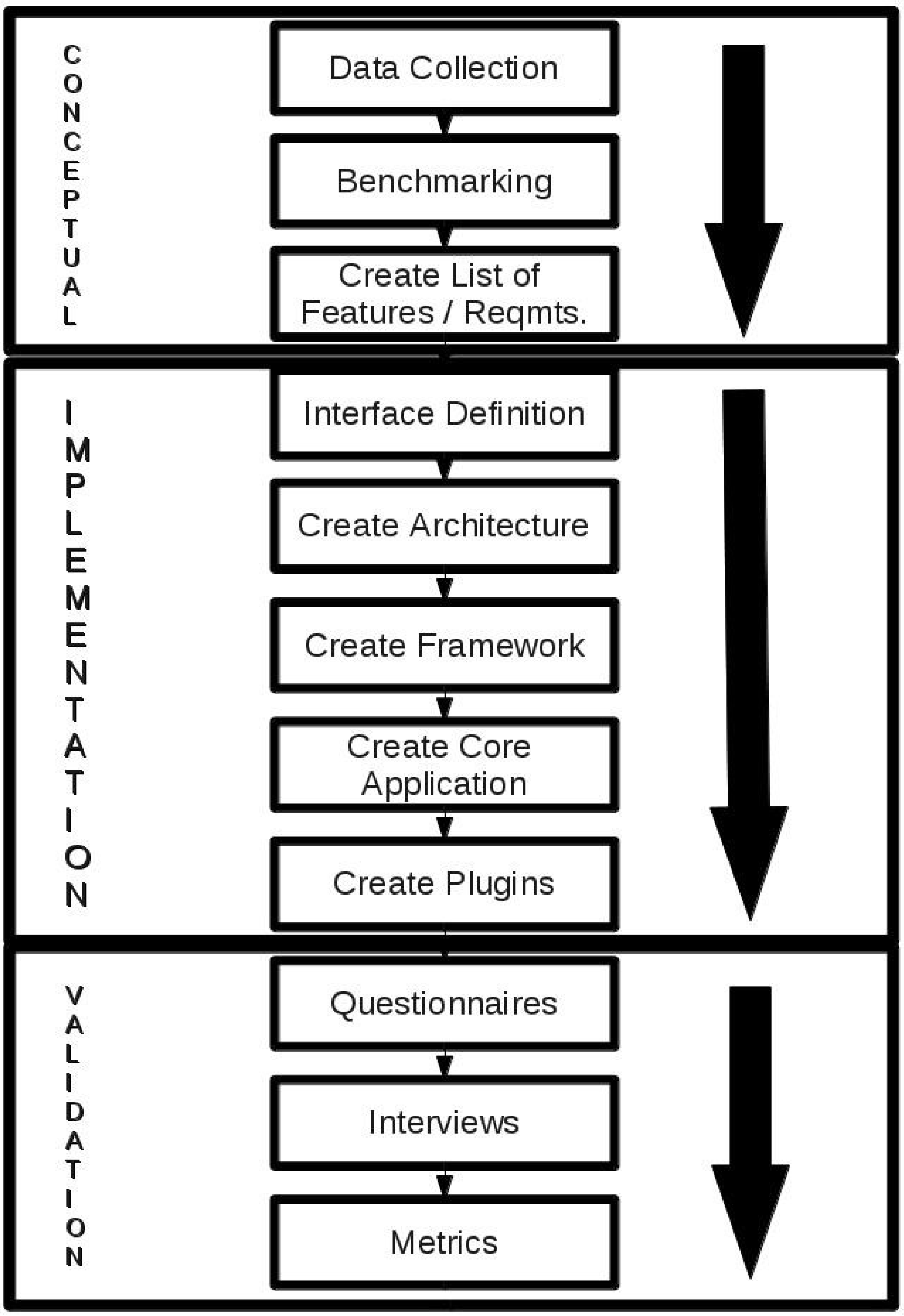
The author engaged in some short interviews via email with some of the more frequent users of the system, which offered some insights into the ways the application is being utilised.

[34]GNU LGPL license. Available at: <http://www.gnu.org/licenses/lgpl.html>. Accessed: 13 Jun. 2013.

Phase 3.3: Metrics

Various metrics and feedback relating to the project were analysed in order to provide a fuller understanding of the impact of the application. Figures such as the number of users, number of members of the user mailing list, as well as the results of reviews and ratings were collated.

Figure 16: Methodology – Flowchart



Source: Finch (2013)

5 THE CONCEPT OF LIVES

5.1 FUNCTIONAL REQUIREMENTS

Following the methodology set out in Chapter 4, various features were selected as being essential functional requirements for the LiVES application. No formal scientific technique per-se was employed to derive this list – instead it was compiled by considering several influences. These influences include - an analysis of the common features of existing video applications; personal experience of the author; discussions with specialists; communications with users (both individuals and groups); and some features which are required to conform with the guidelines used[35]. The features selected are as follows:

- **Playback at variable rates / reverse playback.** The application must be able to play video and audio at any desired rate, both forwards and in reverse, and have fast and accurate random seeking. During playback, it should be able to switch instantly between different video sources. This is a standard requirement of VJ applications.
- **Real-time effects.** The application would need to offer support for real time video effects of various types: filters, mixers, and generators. This is a standard requirement of VJ applications.
- **Edit individual frames.** It must be possible to review and export each individual frame of any video clip which has been imported. This requirement is derived from communications with users.
- **Clip editor.** The software should offer the ability to edit individual clips, providing features like resizing frames, resampling (changing the video frame rate without changing its duration), rotating the frames, and cutting and pasting of frames within the same clip and between different clips. Features for editing the audio of a clip should also be provided. This requirement arose from personal experience and from communication with end users.
- **Multitrack editor.** The application should offer a multitrack interface, through which

[35] Recall also that adhering to the *Twin Peaks* software development process, the requirements are adjusted during the project development, based on a validation of the previous releases.

it should be possible to arrange imported clips using a timeline, apply effects and video transitions, and then render the result as a new clip. This requirement came from user requests[36].

- **Crash recovery.** If the program crashes, then as far as possible the user should not lose any of their work, and should be able to restart the application and continue from a point before the crash occurred. This requirement comes from both the Human Interface Guidelines (the user should never lose any data), and the requirements of VJs that in the unlikely event of a failure during a performance, it must be possible to recover the system as rapidly as possible.
- **Wide range of input formats.** The program should support as large a range as possible of input sources, in terms of video formats. This requirement came from personal experience and from user feedback.
- **Range of high quality output formats.** Support should be offered for as wide a range as possible of encoded video outputs. Encoding should be done to a high level of quality. This should include a range of free/open video formats. This requirement is derived from personal experience.
- **OSC support.** The program should be controllable using the OSC messaging format, which is becoming increasingly used for video applications. This requirement came from conversations with other developers and with specialists.
- **Ability to record performances.** During real-time playback, the user should be offered the option to record their performance – including such items as playback rate changes, clip switches, real-time effects, and audio changes. After playback is finished, the user should be able to render the recording as a new clip. This requirement was developed from personal experience.
- **Multiple monitor support.** The application should support at least two monitors – one for use as a “control” monitor, the second monitor as playback output. For performances one or more of these monitors would actually be a projector. This requirement came from personal experience, user requests[37] and is a standard feature

[36]E.g. <https://sourceforge.net/p/lives/feature-requests/9/> (example of request for improvement to multitrack mode)
[37]E.g. <https://sourceforge.net/p/lives/feature-requests/6/> (request for enhancement to Xinerama, i.e. Dual monitor mode).

of VJ applications.

- **MIDI / joystick control.** It should be possible to control the application by means of a MIDI controller. This includes the ability to map the various MIDI controls to functions in the program as desired. The program ideally would also offer support for control via a joystick or game controller. This requirement came from user requests[38].
- **Frei0r compatibility.** The application should offer support for frei0r video effects (Appendix A - Definitions), which is an extensive set of cross platform, free software video plugins. This requirement arose from discussions with specialists, and the work to implement it in LiVES was partly sponsored by LinuxFund[39].
- **Jack Audio integration.** The application should allow integration with the Jack Audio Toolkit (Appendix A - Definitions), since this is used commonly by multimedia artists. The Jack framework also provides a transport function, which may be used to synchronise the video with software music instruments. This requirement also came about as a result of discussions with other developers and specialists.
- **Audio plugins / filters.** The software should provide the ability to use existing audio plugins, for example, LADSPA (Appendix A – Definitions). This requirement arose from discussions with specialists and via user requests.
- **Firewire support.** Supporting firewire (Appendix A – Definitions) is essential to allow the application to import clips from a large range of video cameras. This need arose from personal experience.
- **TV Card / web-cam input.** The program should provide features to allow direct import of video material from TV cards and from web-cams which may be attached to the machine. This feature may be used to capture material to be further processed, or it may be used as a live performance source. This requirement was derived from user requests and personal experience.
- **Video for Linux (v4l2) or similar support.** This feature allows the application to masquerade as a web-cam itself, and thus enables its video output to be used as input to other applications.(Appendix A – Definitions). This requirement arose from user

[38]E.g. <https://sourceforge.net/p/lives/feature-requests/47/> (request to add MIDI control)
[39]LINUXFUND project page for LiVES. Available at: <http://www.linuxfund.org/projects/lives/>. Accessed: 13 Jun. 2013.

requests and discussions with developers.

- **Subtitles / captioning.** The application should be able to load and display subtitles and captions related to a video clip. This requirement comes from personal experience.
- **Themes / GUI interface.** The application should provide an easy to use and intuitive interface for the user. This requirement comes from user requests and Human Interface Guidelines.

5.2 NON-TECHNICAL REQUIRMENTS

Aside from the technical requirements listed above, such an ambitious piece of software would need to be extremely <u>feature complete</u>, in order to be able to fulfill the combined requirements of each of the areas of VJ application, video editing application, and video processing environment. It should merge the functionality of at least a Resolume, a Premiere and a Pure Data type application in one program. If it could provide *additional* functionality, for example the ability to function as part of an online video editing system, then this would be a bonus.

Such a program would need to be created with a <u>very flexible internal framework</u>. It would need to be able to take video frames from any kind of source and to play them back in any order desired, combine them and apply effects; either in response to real time events (key presses and so on), or else in a predefined order via an event list. It would need to have a very high level of <u>performance</u> to be able to respond to events and process them in real time. It would be vital that it could play back video and audio smoothly. It would need a rich and well defined remote interface to be able to act as a video programming environment. It should support multiple platforms, and be able to scale itself automatically for a range of hardware from low-end to high-end machines, since VJs work with a wide range of hardware, and frequently they must work on equipment which is supplied to them at a venue.

In addition, such software would need to promote <u>ease of use</u> so that VJs can start working with it straight away. The interface would need to be intuitive so that the main features are easy to locate and to facilitate experimentation with different modalities: at the same time it should allow for a full range of functionality and be feature rich to allow for depth of experimentation and customisable use. The interface would need to be configured in such a way that it exposes the functionality of the particular mode of working the user is currently engaged in – real-time VJing, clip editing, multitrack video editing, data-flow programming, scripting and so on. Customisation of the application through an extensive set of preferences is desirable, since VJs work in different ways and for different purposes; thus they should be able to configure the application to facilitate their own style and current needs.

The program would need to be extremely <u>stable</u>. During real-time playback this is a must, since it may be used by a VJ for live performances, or via scripting as a tool for installations, and hence may need to run unattended for long periods.

Ideally, such a program would also be <u>Open Source</u>, as this would allow for the system to be customised rapidly and easily, and would allow for experimentation at the level of code, and provide adaptability for unknown situations. Use of the Open Source model should also result in a higher level of stability and feature completeness, since it opens the code base up to a wide range of contributors. It would also provide the users with peace of mind, knowing that the code will always be available, regardless of the success or failure of the developers. [40]

Finally, the application should be available at <u>zero cost</u> to users, as this makes it accessible to potentially all users, regardless of their financial situation. From personal experience, it is known that many amateur VJs and video editors work unpaid, and so the outlay to purchase a program for such use may be significant.

5.3 CHALLENGES

Apart from the need to have a large amount of technical knowledge, and the great amount of effort involved in producing such a piece of software, there are some particular areas of development which are particularly challenging:

– Internal framework design, in a way which unites effects, frames, events and data. This requires knowledge of the manner in which both real-time and non-real-time

[40]A concrete example of this is the Netscape browser. Although Netscape, the company, no longer exists, the fact that their browser was ultimately open source meant that it was able to be further developed, and eventually this became the immensely popular browser, Firefox. [See also: MOZILLA history. Available at: <http://www.mozilla.org/en-US/about/history/>. Accessed: 13 Jun. 2013.]

video applications function, as well as familiarity with technical details of video and audio (video colourspaces, audio interleaving, etc.) and needs to be done in a flexible and expansible fashion to allow for potential future developments

– Design and implementation of plugins. This again requires knowledge of video and audio technology (for encoders and decoders one needs to know a great deal about different video and audio codecs and container formats). In addition, the interfaces between the host and plugins need careful thought and some element of trial and error is involved.

– Design and implementation of a framework for automatically generating graphical interfaces for plugin parameters. Having to design a complete graphical interface for each plugin would add enormously to the plugin complexity and rapidly become tedious. Nevertheless, plugins frequently have some parameters which must be exposed graphically to the user. This needs to be done in a clear and consistent manner in order to follow good interface design principles.

– Implementing the application in such a way that it does not compromise the security of the system or of the user. For example, the user should not generally be able to access the files of another (projects, clips and so on), unless the other user has specifically allowed that. In addition, the developer must consider the possibility that the application is being run by a system administrator (root, superuser, admin) and prevent that user from inadvertently damaging the system. This is particularly important for an application which is capable of being controlled remotely.

– Design and implementation of the graphical interface. This requires not only consideration of human interface guidelines, but also intimate knowledge of graphical toolkits, for example to be able to create custom widgets when necessary. Localisation issues are also important to understand in this respect; for instance, how will the direction of text (left to right or right to left) affect the layout of the interface ? If a short label is required, what happens if a particular language translator translates the text into a much longer word or phrase ?

– Refactorisation of the code will sometimes be necessary, to make it more maintainable, reduce the complexity of the evolving code and so on. Thus the core developer(s) will need an overall understanding of all areas of the code to carry this out in a manner which does not cause bugs or problems.

- Optimisation of the code. A portion of the code needs to run in real time, so it is vital to understand the intricacies involved here. In addition to the fact that code needs to be highly optimised, multithreading may need to be employed - which in turn necessitates knowledge of thread locking, shared memory and so forth.
- Design / rationalisation of the OSC interface. If a remote interface is to be provided to the application using OSC, then the set of available messages needs to be designed to be internally consistent. This results in making things like scripting easier to understand and unambiguous, and provides a logical framework for extending the message set to expose further functionality.
- Familiarity with hardware. The application will need to work with various peripherals – cameras, soundcards, video cards and others. The developer(s) will need to understand how these devices function at the hardware level, as this has an effect on the way that the software is to be coded.
- Project activities not directly related to coding – documentation, maintenance of the website, mailing list(s), forums, bug and feature request trackers. These are all part of a good software project and offer indispensable support to the end user. These channels of communication are particularly important for an Open Source project due to the direct involvement of the user base within the development process.

6 **LIVES**

6.1 THE HISTORY OF LIVES

LiVES development began in late 2002. The author was inspired to start creating a new video editing application after purchasing a new photo camera. In addition to taking photos, the camera was able to record small clips of video; however for technological reasons, the video clips were limited to a duration of just ten seconds. As well as the limitation in duration, the clips were recorded without audio because of a lack of microphone on the camera. The author decided that a means to increase the usefulness of the camera would be to use a program to join together several of these ten second segments of video, and to add in some audio – perhaps music or a commentary. Finally he hoped to be able to encode the finished result. However there was a problem – the author was committed to using only Linux, and at that time none of the applications available for that operating system could import the camera's format. On the other hand, the author was able to use a different program to actually play the video clips on Linux, although not to edit them. Since the player program was able to output the frames as a sequence of images, the author had the idea of making a simple editor to play the images in sequence and to edit the images, then to add sound, and then to encode. Thus LiVES was born.

Right from the start, a decision was made to develop the program as Free Software. It was hoped that in this way the author could contribute to the Linux ecosystem, and in return could involve the programming community more fully in the development of LiVES.

Having satisfied his original needs, the author began consulting with the community about how the program could be further developed. The principal channels for this were email communications, mailing lists, and personal attendance at conferences and seminars such as the Piksel Festival.

6.2 THE PHILOSOPHY OF LIVES

The application has always been developed as Free Software, using the GNU GPL for the core code, and the LGPL for plugin libraries (to allow for the possible inclusion of proprietary plugins). In addition to the benefits of being open source which were mentioned earlier, the code base is open for any other GPL project to use. As proof of the contribution of the LiVES code to the Free Software community, code for the threading model used for LiVES real-time effects plugins has been copied and adapted by the developer of the Veejay application for use in that program.

Further, the need for Free Software video editing applications is so important that the Free Software Foundation (FSF) lists this as a priority area[41] for Free Software development.

Finally, the fact that LiVES is open source means that the "many eyes" proposition holds true for the code. As a vivid demonstration of this fact, the author received an email from a Redhat developer advising of several systematic security flaws in the LiVES code relating to interactions with the operating system. As a result, these flaws were removed by rewriting a portion of the code, the end result being an application which is much more secure.

The LiVES development process remains flexible, and attention is given to the requests and suggestions of users. These are incorporated into the program whenever possible. An example of this is support for web-cam and TV card input which was a popular feature requested by several users. This feedback has helped LiVES to greater fulfill the needs of a large segment of users.

6.3 THE LIVES COMMUNITY

Various community tools are available for Free Software projects and these are leveraged extensively by the LiVES project. For example, the main website and copies of the packaged source code are hosted on the Sourceforge site, which also provides the mailing lists, bug and feature request trackers, and discussion forums, free of charge.

The code is developed principally for Linux. Each time a source code release is made, volunteers from several Linux distributions are notified, and they generally go on to create a

[41]FREE SOFTWARE FOUNDATION, priority projects. Available at: <http://www.fsf.org/campaigns/priority-projects/priority-projects/>. Accessed: 13 Jun. 2013.

package specific for that distribution, based on the generic source. In this way the package distribution is done in a decentralized fashion.

LiVES is structured in such a way that all messages within the interface can be translated and displayed as localized text. The tool used to do the translation is an element of Ubuntu Launchpad. Translation[42] is, again, carried out by teams of volunteers in a decentralized manner. Owing to the volunteer efforts of the translation teams, the application has been localised at least partially to thirty different languages, including some less common ones such as Catalan, Estonian, Galician, Occitan and Uyghur.

For support, the main channels are the lives-users mailing list and discussion forums on Sourceforge.

The pace of releases is fairly regular, in general there is approximately one release per month. The current development code is always available for viewing and downloading since it is hosted using Subversion, again on the Sourceforge site.

6.4 EQUIVALENT COMMERCIAL DEVELOPMENT

The software is developed rather informally, but to give an idea of the equivalent development effort using proprietary and commercial methods, the site ohloh.com provides an estimate based on the number of lines of code for a project.

According to the Ohloh site, which uses the Basic COCOMO[43] model:

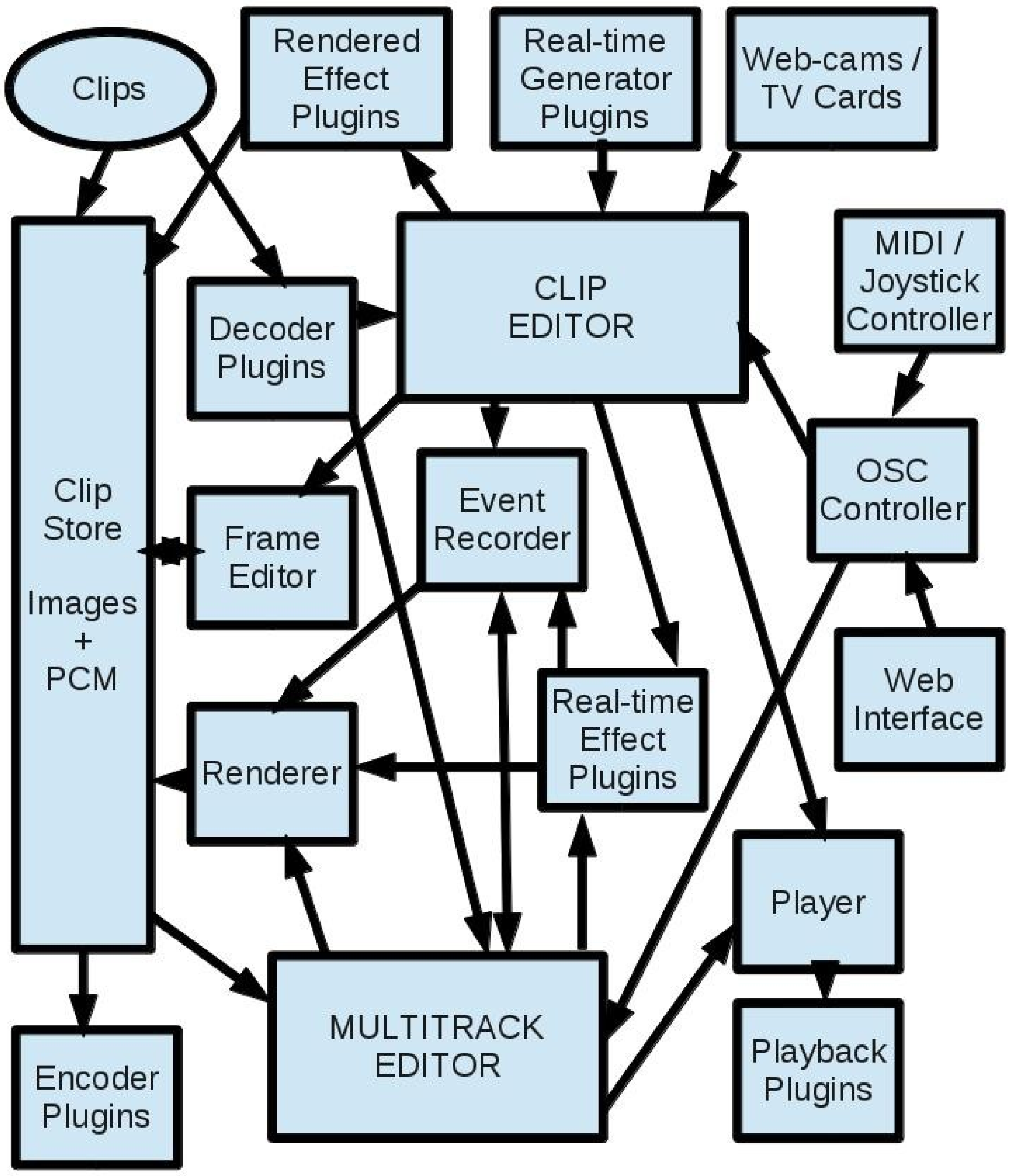
> LiVES has 261,315 lines of code. This represents an estimated 69 person-years of work. At an average programmer salary of $55,000 USD, the total development cost for LiVES, if developed commercially would have been **$3,777,981** USD.[44]

6.5 THE ARCHITECTURE OF LIVES

[42]LAUNCHPAD translation page for LiVES. Available at: <https://translations.launchpad.net/lives/trunk/+pots/lives>. Accessed: 13 Jun. 2013.
[43]WIKIPEDIA, COCOMO. Available at: <http://en.wikipedia.org/wiki/COCOMO>. Accessed: 13 Jun. 2013.
[44]Estimated equivalent commercial cost for LiVES, ohloh.net. Available at: <https://www.ohloh.net/p/lives/estimated_cost>. Accessed: 13 Jun. 2013.

6.5.1 **Functional overview**

Figure 17: Functional overview of the LiVES application



Source: Finch (2013)

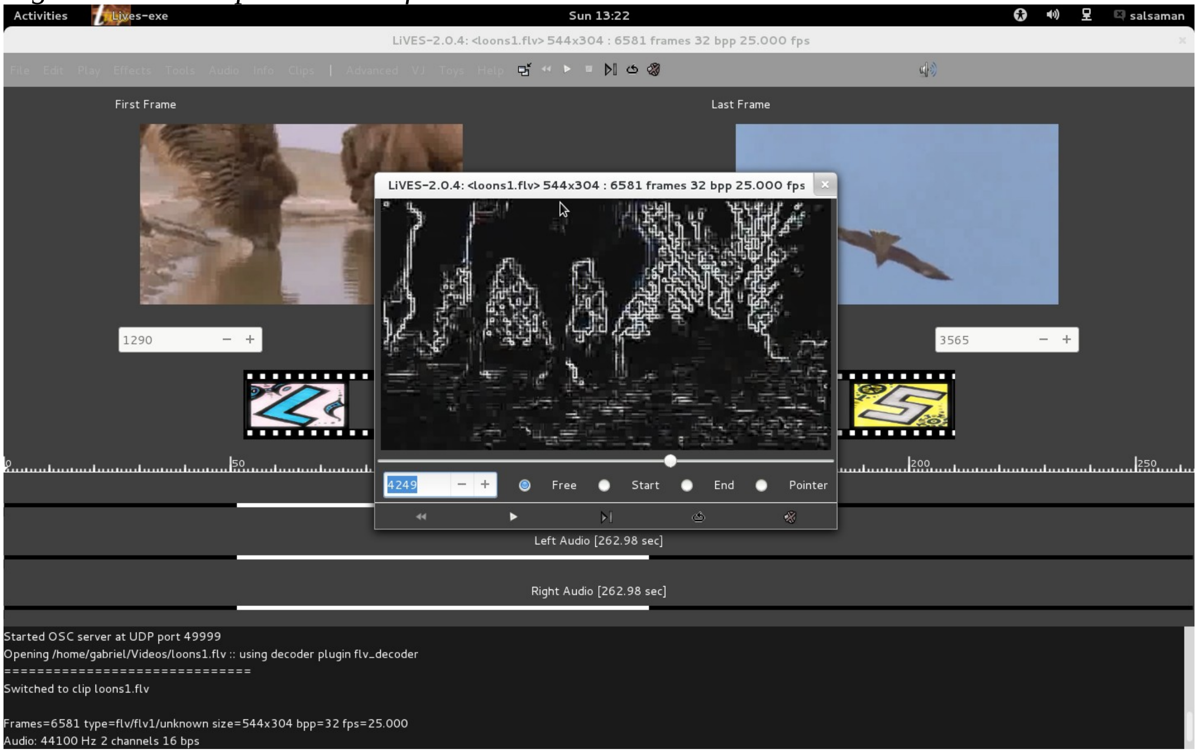
Figure 17 shows the main functional elements of LiVES. Starting at the top left of the diagram, we find **clips** (which are external to LiVES), which may actually refer to video, audio, images or even remote streams. Depending on the source type and format for the clip, the frames in it may be read when required by a **decoder plugin** (see below for a description of the plugins). If no suitable decoder plugin is found, then a fall-back method is used which decodes all of the frames at once and places them in the **clip store**. The clip store is a directory on disk which contains an indexed collection of frames (stored as individual image files), and audio (stored as raw PCM). In addition, each clip in the clip store has a meta-data file which holds details such as the frame dimensions (height and width), frame rate, audio rate and channels, and so on. Material from the clip store may also be passed to an **encoder plugin** to produce an encoded clip. Clips which make use of a decoder plugin also have an entry in the clip store; the contents are initially the meta-data file and decoded audio, alongside an index which denotes which frames have been decoded to images, and which are still within the source video clip. (That is to say, LiVES use a copy-on-write mechanism for clips for which a decoder plugin can be found, in all other cases it uses a copy-on-read mechanism).

Within the LiVES core, the **clip editor** interface can make use of the clip store and the decoder plugins. The clip editor can also take frames and audio from the **real time generator plugins**, as well as from web-cams and TV cards. This material can be played directly using LiVES' internal **player** or any one of a number of **playback plugins**. Alternately, the material may pass through one or more real-time effects before being displayed. Other courses of action are available to the clip editor; clips can be altered using the **rendered effect plugins**, or by the **frame editor** functions. In both these cases, the results are written back to the clip store. In the case where a clip is using a decoder plugin, the plugin is used for reading, and altered frames are written to the clip store. A further possibility in the clip editor is to record real-time clip switches and effects, making use of the **event recorder** in LiVES. After recording is complete, the recorded events may be sent to the **renderer**, which creates a new entry in the clip store, and uses the events from the event recorder to create the clip.

Within LiVES, there is another interface which is the **multitrack** window. The multitrack window also makes use of the clip store and decoder plugins as inputs; it uses the same internal player and playback plugins as the clip editor, the same real-time effects plugins, and it also makes use of the renderer. In this case, the events to be rendered come from the timeline rather than from the event recorder.

Finally, on the right hand side of the diagram we have the **OSC controller** which can control and manipulate both the clip editor and the multitrack interface. For example, in the clip editor, it can be used to start and stop recording to the event recorder, to switch clips, to activate or deactivate real-time effects, or to cut and paste between individual clips. In the multitrack window, it can be used to insert blocks into the timeline. The **MIDI/joystick learner** sits as a layer above this; once programmed, inputs from these devices are converted to OSC messages and sent internally to the OSC receiver as if they had been received externally as OSC. There is also an experimental **web interface**, which works with a web server and communicates with LiVES using OSC messages.

Figure 18: The clip editor interface in LiVES



```
Started OSC server at UDP port 49999
Opening /home/gabriel/Videos/loons1.flv :: using decoder plugin flv_decoder
============================
Switched to clip loons1.flv

Frames=6581 type=flv/flv1/unknown size=544x304 bpp=32 fps=25.000
Audio: 44100 Hz 2 channels 16 bps
```

Source: lives.sourceforge.net

Above (Figure 18), we see the *clip editor* interface for LiVES. The main part of the GUI shows start and end frames for one particular video clip which has been loaded. The optional separate playback window is shown in the foreground. Below the start and end frames we see the single clip timeline. The video duration is represented as the top white bar and the two bars below this are the left and right audio channels. At the bottom of the screen is the message area which keeps track of all the operations of the application.

The position of the start and end frames (which can be adjusted via keyboard, mouse or OSC) defines the selection for the clip. The selection can be used for applying non real-time effects (a.k.a rendered effects) and for cut/paste/insert/delete operations. The latter are collectively referred to as the *frame editor* operations: they make use of the back end process, since they interface directly with the *clip store*.

Multiple clips can be opened, and the user can switch between them in a variety of ways – with the keyboard, mouse scroll wheel, from the Clips menu, via an OSC command, via a configured MIDI controller or via a joystick.

During playback the visual configuration may change a little. If the separate window is hidden, then playback will be shown in a window within the main interface. If the separate window is shown, then playback will be performed in that window. In dual monitor mode, the playback window can be set to switch to the second monitor automatically during playback. In addition, the playback window may be set fullscreen.

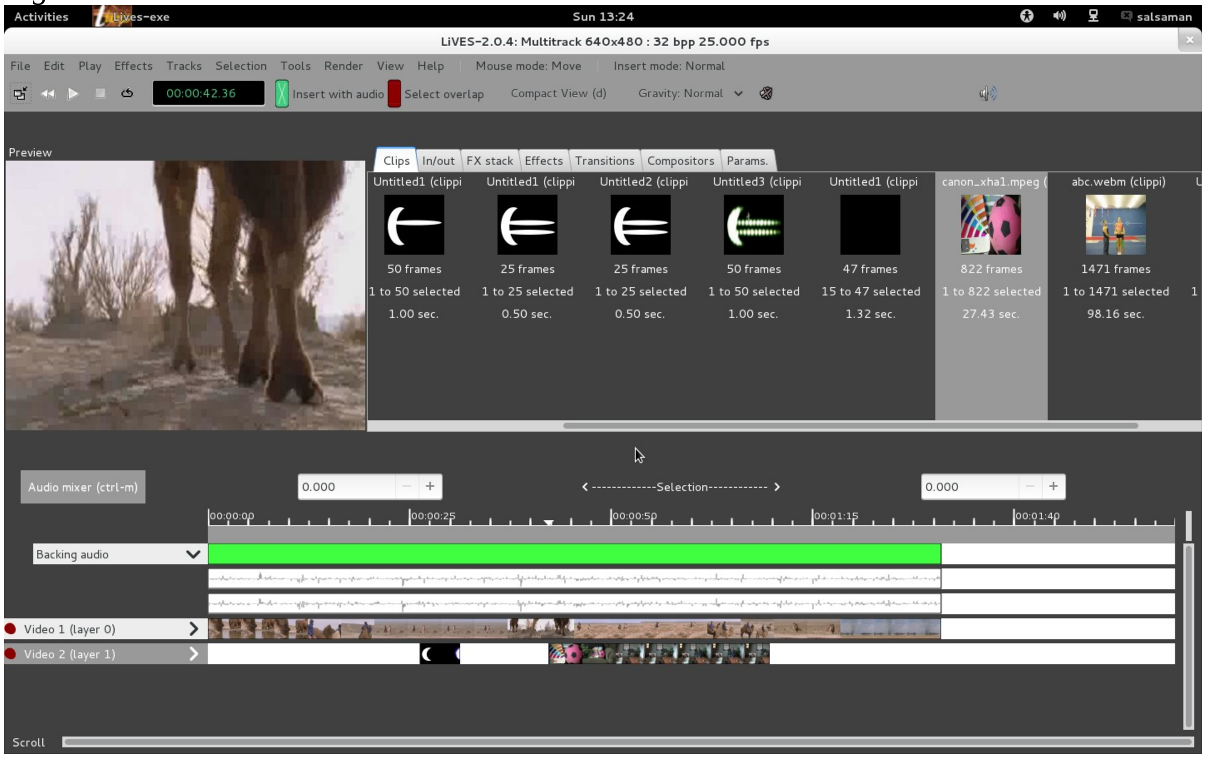
Whilst the video is being played back, the user may switch clips, adjust the playback rate and direction, freeze the playback, enable and disable real-time effects, adjust effect parameters, and switch effect "banks". All of this can be done in real-time, either through the keyboard, a configured MIDI controller or joystick, or via OSC. The user can choose through preferences whether the audio follows clip switches and rate changes or not.

The LiVES *player* is a variable rate playback engine. In normal playback mode, timing information is taken from the soundcard – this is to ensure that video remains in synchronisation with audio, which may play a little faster or slower than it should. If there is no soundcard, or if video is being received from a streaming source, then the system clock is used for timing instead, and audio is resampled on-the-fly to keep it synchronised with the video. The player can also be synchronised by an external source – for example the transport controller which is part of Jack audio, should the user set this through preferences. When a frame is timed to play, the video frames are pulled from their various sources, effects are applied, and the output is sent to whatever player output is active. The audio player runs in a separate thread.

If desired, **record mode** can be enabled before or during playback (again via keyboard, menu option or OSC message). The user can select through preferences which events are recorded (real-time effects, clip switches and audio changes are optional; frame changes and FPS – frames per second – speed changes are always recorded). Recording can be toggled on and off during playback.

After playback, any events which were recorded can be previewed or rendered. If the user chooses to render, then the event list which was recorded is sent to the *renderer*. The purpose of the renderer is to take an event list (a description of frames, effects, effect parameters, audio) and to render this as physical frames and audio. In doing so, this creates a new clip (or optionally, overwrites part of the current clip).

Figure 19: The multitrack editor in LiVES



Source: lives.sourceforge.net

Shown above (Figure 19) is the *multitrack editor* for LiVES. The user can switch between clip edit and multitrack mode in a few ways – via the keyboard, using the mouse in the menu bar, or via an OSC message.

The top left part of the screen is the preview window, which shows the frame at the current cursor time. To the right of this is the “polymorph” window which attempts to show relevant information to the user (for example, when inserting clips in the timeline it shows thumbnails of all the clips, when an effect is inserted it shows the parameters of the effect). The lower part of the screen is the timeline. Generally below this is the same message window which is shown in the clip editor mode, although in this case the timeline has expanded because the user has expanded the backing audio track (shown in light green) to show its left and right components. Here we see two video tracks (Video 1 and Video 2). Any number of tracks may be inserted into the timeline. Clips can be dragged to the timeline, or inserted using the keyboard, menu options, or via OSC.

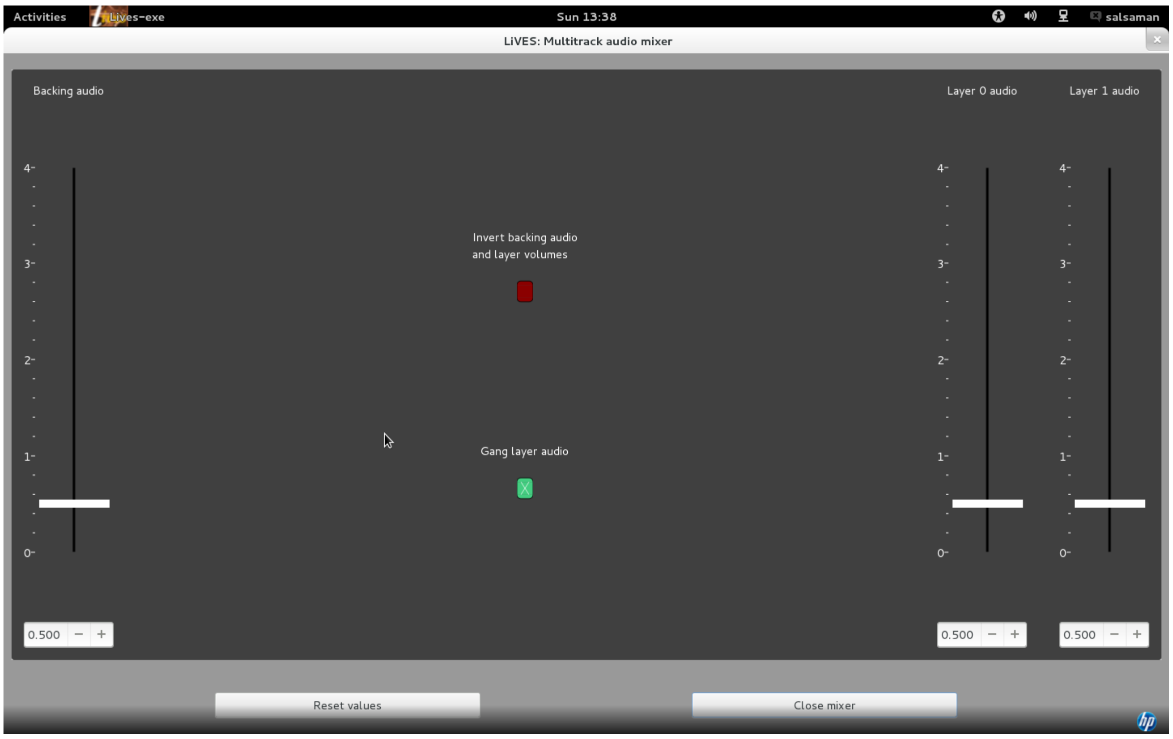


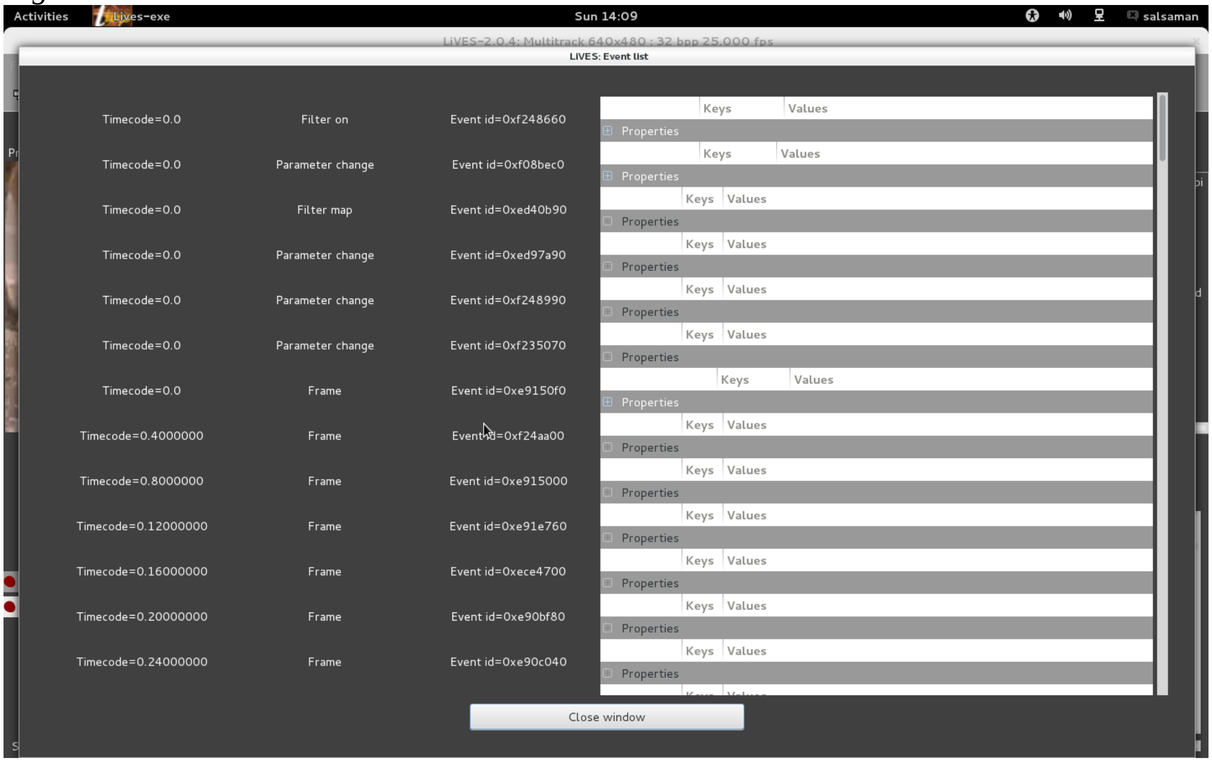
Figure 20: The multitrack audio mixer in LiVES

Source: lives.sourceforge.net

In Figure 20 we see the audio mixer for the LiVES multitrack editor. Here one can set overall levels for backing audio and for the individual track levels. [Within the multitrack window itself, one can apply finer grained, time dependent adjustments to level and pan for each track.]

The timeline in the multitrack editor is used to create an event list (similar to the event list which is created in record mode during real-time playback). The event list can be viewed in the event list viewer window, as the example in Figure 21 shows:

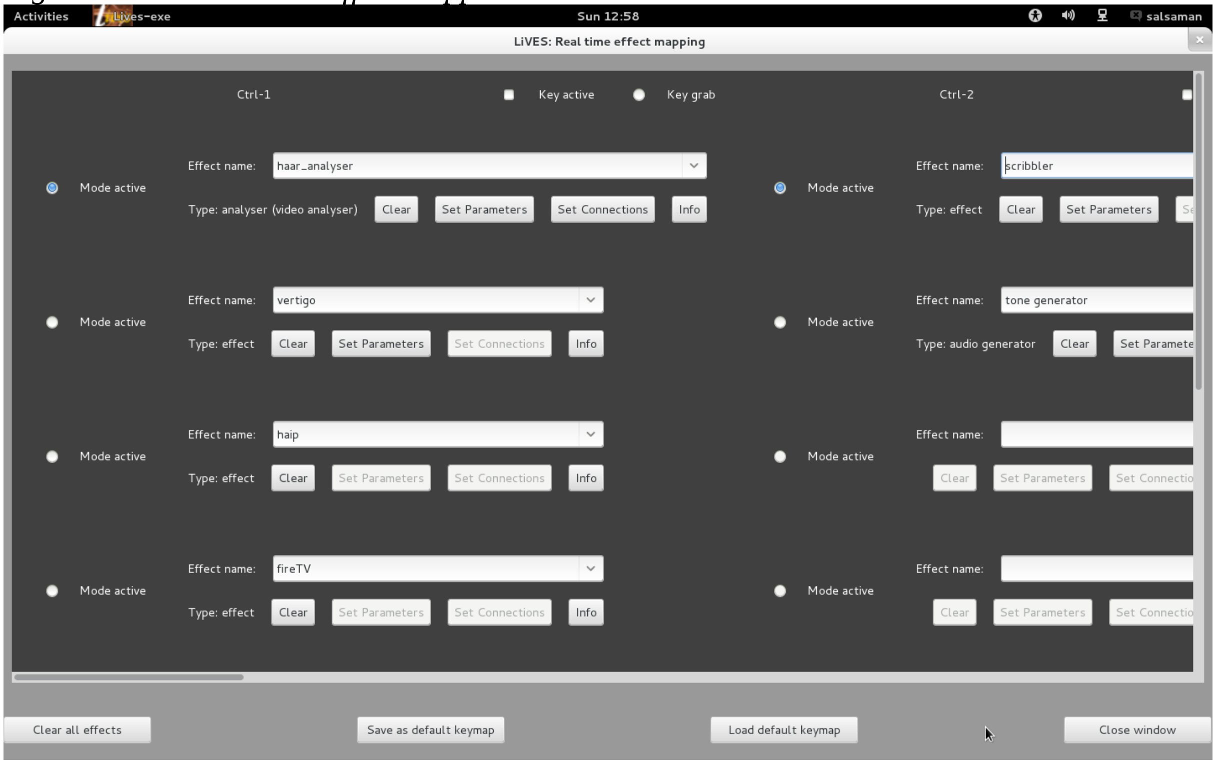
Figure 21: The event list viewer in LiVES



```
Timecode=0.0           Filter on           Event id=0xf248660
Timecode=0.0           Parameter change    Event id=0xf08bec0
Timecode=0.0           Filter map          Event id=0xed40b90
Timecode=0.0           Parameter change    Event id=0xed97a90
Timecode=0.0           Parameter change    Event id=0xf248990
Timecode=0.0           Parameter change    Event id=0xf235070
Timecode=0.0           Frame               Event id=0xe9150f0
Timecode=0.4000000     Frame               Event id=0xf24aa00
Timecode=0.8000000     Frame               Event id=0xe915000
Timecode=0.12000000    Frame               Event id=0xe91e760
Timecode=0.16000000    Frame               Event id=0xece4700
Timecode=0.20000000    Frame               Event id=0xe90bf80
Timecode=0.24000000    Frame               Event id=0xe90c040
```

Source: lives.sourceforge.net

Figure 22: The real-time effect mapper in LiVES



Ctrl-1

Effect name: haar_analyser
Type: analyser (video analyser)

Effect name: vertigo
Type: effect

Effect name: haip
Type: effect

Effect name: fireTV
Type: effect

Ctrl-2

Effect name: scribbler
Type: effect

Effect name: tone generator
Type: audio generator

Effect name:

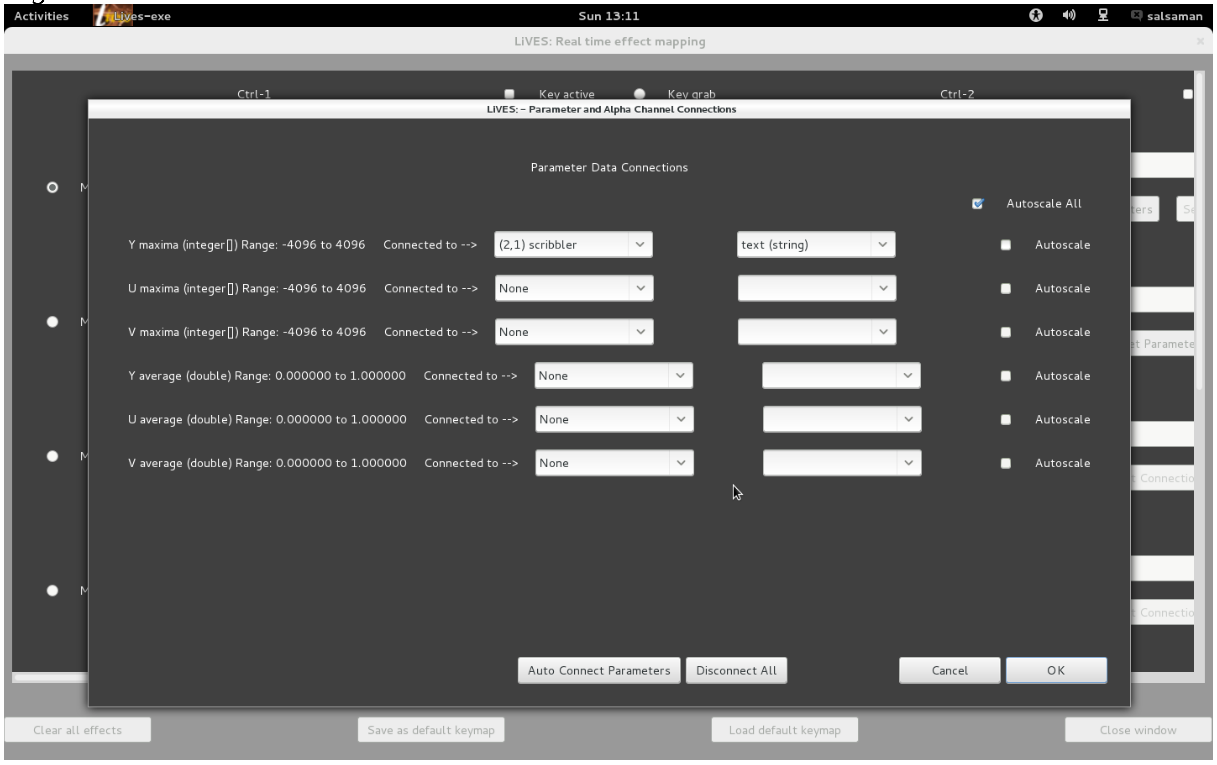
Effect name:

Source: lives.sourceforge.net

When the user chooses to render from the multitrack editor, the event list is passed to the renderer, in the same manner that recorded events are handled. This process creates a new clip, which is added to the clip store.

Above (Figure 22), we see the real-time effect mapper. Here real-time effects can be mapped to various keys on the keyboard. Each key has a bank of effects which can be selected in turn. Effects can also be mapped to virtual keys, which are accessible via OSC, MIDI or joystick control.

Figure 23: The Data Connector in LiVES



LiVES: – Parameter and Alpha Channel Connections

Parameter Data Connections

☑ Autoscale All

Y maxima (integer[]) Range: -4096 to 4096	Connected to -->	(2,1) scribbler	text (string)	☐ Autoscale
U maxima (integer[]) Range: -4096 to 4096	Connected to -->	None		☐ Autoscale
V maxima (integer[]) Range: -4096 to 4096	Connected to -->	None		☐ Autoscale
Y average (double) Range: 0.000000 to 1.000000	Connected to -->	None		☐ Autoscale
U average (double) Range: 0.000000 to 1.000000	Connected to -->	None		☐ Autoscale
V average (double) Range: 0.000000 to 1.000000	Connected to -->	None		☐ Autoscale

Source: lives.sourceforge.net

In Figure 23, we can see the data connector window. This is a recent addition to LiVES. Here, output parameters from one effect can be connected to input parameters of a second. During playback, if both effects are activated, then data will flow from the first to the second.

The *OSC controller* (if enabled by a startup option, or via preferences) runs via a timer which fires every 4 milliseconds. The controller checks to see if there is a message in its queue, and if so will parse and handle the first message in that queue. OSC messages are received generally via a UDP port. The *MIDI/joystick controller* can be configured via a user interface to convert MIDI or joystick events into OSC style messages, which LiVES sends internally to the OSC controller.

6.5.2 **Technical overview**

In terms of code, we can divide LiVES into the **core application** and **plugins**. The core application consists mainly of the clip editor, multitrack interface, renderer, OSC listener, internal player, and the graphical interface. The code here is not really modular since much of it is common to several components. There is, however, a division here into client and server components. The front end is written in C, and there is a back end component written in Perl which is used mainly for interaction with the clip store and file system. In this way, the details of the clip store are largely transparent to the front end. The back end is also employed as an interface to transparently wrap other programs in the system, in the cases where these are used (for example for fallback video decoding). Doxygen documentation for the front end portion is available on the LiVES website[45].

A recent initiative in the LiVES code is to abstract the low level details of the user interface. LiVES currently uses the GTK+ toolkit, and this initiative originated shortly after GTK+ version 3 was released. Prior to this, LiVES was built around GTK+ version 2, and the new version introduced some backwards incompatible changes. Rather than stick with the now deprecated GTK+ 2 code, or to migrate the code to GTK+ 3, a decision was made to abstract the graphical interface. Rather than call the graphical code directly, an abstraction layer was introduced, and within the abstraction layer all of the handling for the differing versions of GTK+ takes place. The end result is that LiVES can now be compiled against either GTK+2 or GTK+ 3, without requiring any further alterations to the code. This work is still ongoing with the end goal being to allow LiVES to be compiled against any of several graphical toolkits. A further exciting possibility would be to include integration with the LiVES webserver, to generate the user interfaces automatically using HTML and JavaScript.

LiVES uses a custom library called “libweed” which implements the WEED (short for <u>W</u>eed is an <u>E</u>ffects and <u>E</u>vents <u>D</u>river) framework. Weed is an extremely flexible framework, and is used extensively within LiVES and within many of its plugins (real-time effects, and playback plugins).[46]

[45]DOXYGEN documentation for LiVES. Available at: <http://lives.sourceforge.net/doxygen/LiVES/index.html>. Accessed: 13 Jun. 2013.

[46] Several documents describe the Weed framework:
<http://lives.svn.sourceforge.net/viewvc/lives/trunk/weed-docs/weedspec.txt>
<http://lives.svn.sourceforge.net/viewvc/lives/trunk/weed-docs/weedaudio.txt>
<http://lives.svn.sourceforge.net/viewvc/lives/trunk/weed-docs/weedevents.txt>
<http://lives.svn.sourceforge.net/viewvc/lives/trunk/weed-docs/weed-utils.txt>
<http://lives.svn.sourceforge.net/viewvc/lives/trunk/weed-docs/weed-plugin-utils.txt>
<http://lives.sourceforge.net/doxygen/libweed/index.html>

6.5.3 **Plugins**

LiVES also makes use of several types of plugin. The advantages of using plugins are discussed by (CHATLEY; EISENBACH; MAGEE) and others, e.g. (MAYER; MELZER; SCHWEIGGERT, 2002). The types of plugins used in LiVES are described briefly here.

- **Decoder plugins**

Decoder plugins may be written in C or C++. The goal of these is to retrieve metadata from a video clip if possible, and if this succeeds, to return a decoded version of a specified frame from that clip. The interface is very simple, consisting of just a handful of mandatory function calls: a function to get the plugin version, another to get clip metadata, and another to pull a frame from a clip. This latter function has two requirements – it must return exactly the frame requested, and it must be done as rapidly as possible. To ensure this, various methods are employed, such as indexing the positions of keyframes[47] in an internal list. Other optional functions include a function to decode audio from the clip.

- **Encoder plugins**

Somewhat the counterpart of decoder plugins are encoder plugins. The objective here is to take a sequence of images from the clip store (possibly with accompanying PCM audio), and produce a new clip. Encoder plugins are binaries and may be written in any language – the plugin simply needs to parse its commandline options and follow a few simple rules. In fact there are encoder plugins written in both Perl and Python.

- **Rendered effect plugins**

Rendered effect plugins are used in the clip editor component of LiVES. There are four types of rendered effect plugins: batch generators, non-batch generators, filters, and transitions. Batch generators are used to generate a batch of frames in one go, these frames are then imported into LiVES via the clip store. Non-batch generators generate frames one by one. Filters take one frame input, apply some effect, and produce one frame output.

[47] For many compressed video formats, full information for each frame is not stored in the video file. The decoder must begin by decoding from a *keyframe*, and then continue decoding frames sequentially until the target frame is reached.

Transitions take frames from two different clips in the clip store and combine them into one frame. Rendered effect plugins need not run in real-time; thus the effects they produce can be as complex and processor intensive as desired. These types of plugins are generated from special script files. Within the LiVES application there is a tool to help create and modify these scripts. The scripts are compiled into Perl applications (although in future they could be compiled into other languages), which are the actual plugins themselves.[48]

- **Real time effect plugins**

The goal of real time effect plugins is to apply an effect in a very short period of time. This type of plugin may be written in C or C++ (examples exist of both). They are used within the clip editor and within the multitrack interface in LiVES. Real-time effect plugins make use of the Weed architecture mentioned earlier. There is a huge variety of real time effect plugins depending on the number of frames input (zero, one, two, or multiple), the number of frames output (zero or one), and whether or not they produce data output. Some of these plugins also handle audio rather than video channels. A number of them are actually wrappers for other effect frameworks. **Real time effect generators** are a specific type of this plugin, which have no input video channels, but do have output video channels.

- **Compound effect plugins**

These are really a type of pseudo plugin; they consist of references to a set of real-time plugins to be run in sequence as a single unit, along with details of data connections between the internal parts. They may also set specific defaults for some of these internal plugins, hiding those parameters from the user. To the user then, the plugin appears as one effect rather than multiple. An example of this is the *image stabilizer* plugin in LiVES which consists of component plugins which analyse the motion between subsequent images in the video stream, a plugin to average the motion over the entire frame, a plugin to handle these values parametrically, and another plugin to shift the frame vertically and horizontally. The user could set this up themselves as separate plugins and then save this as a patch, but it is more convenient to have it ready made as a single unit.

- **Playback plugins**

This type of plugin is used to replace the LiVES internal display plugin, but they are

[48] The script format is described in the document: <http://svn.code.sf.net/p/lives/code/trunk/RFX/RFX.spec>

only active when LiVES is playing back in a certain mode (fullscreen and in a separate window). These plugins are written in C or C++. They are created for specific output purposes, for example optimised playback or streaming.

Some of these plugin formats are described in section 17.3.3 of the LiVES manual[49].

The use of plugins in LiVES has allowed other developers to participate in advancing the project. For example, the openGL playback plugins were created by another developer, and some of the encoder plugins were developed by a third.

6.5.4 **Optimisations, concurrency**

Parallel processing is used in some areas of the code where this would result in performance improvements, for instance in some of the real-time effects, and some colourspace conversions. Some considerations regarding the use of parallelisation of video effects have been discussed by Mayer-Patel (1999). The core of the LiVES code is not optimised for any particular hardware – instead this is achieved through using external libraries and plugins. The idea here is to make the core code as portable as possible.

It should be noted that the LiVES code is completely concurrent - meaning that multiple copies of the application can be run in parallel on the same machine by the same or different users without problems.

6.5.5 **Language breakdown**

Table 4 shows the lines of code per language which have been developed for the LiVES application.

[49]FINCH, G. LiVES manual, plugin formats. Available at: <http://lives.sourceforge.net/manual/LiVES_manual.html#section17.3.3>. Accessed: 13 Jun. 2013.

Table 4: Language breakdown for the LiVES application.

	Language	Code Lines	Comment Lines	Comment Ratio	Blank Lines	Total Lines	Total Percentage
🟧	**C**	223,613	18,884	7.8%	64,519	307,016	84.4%
🟨	**Perl**	13,789	1,399	9.2%	4,041	19,229	5.3%
⬛	**shell script**	8,060	1,319	14.1%	637	10,016	2.8%
🟪	**Python**	6,970	4,762	40.6%	2,639	14,371	4.0%
🟥	**C++**	3,132	856	21.5%	1,421	5,409	1.5%
🟩	**HTML**	2,410	0	0.0%	0	2,410	0.7%
⬜	**Automake**	1,317	89	6.3%	478	1,884	0.5%
⬜	**Autoconf**	1,259	176	12.3%	444	1,879	0.5%
🟪	**Java**	708	323	31.3%	216	1,247	0.3%
⬜	**Make**	57	18	24.0%	21	96	0.0%
	Totals	261,315	27,826		74,416	363,557	

Source: https://www.ohloh.net/p/lives/analyses/latest/languages_summary

6.6 UNIQUE FEATURES

LiVES, in addition, has some unique features for an application of its type. These unique features make LiVES into a more powerful and useful tool, allowing greater flexibility in its use. Furthermore, these unique features provide enhanced possibilities for experimentation and novel ways of interacting with the application. Some examples of these unique features are:

- **The possibility of editing recorded performances**

Whilst a number of VJ applications offer some kind of ability to record performances in real-time, LiVES takes this to another level, by enabling the user to edit their performance after it has been recorded. This is due to the fact that LiVES makes use of the same structure internally both for performance recording, and for the multitrack editor. After the user has recorded something, and playback has finished, one of the options offered (in addition to previewing and rendering) is to open the recording in the multitrack window. The idea here is to give the user the opportunity to make fine adjustments and to perform other editing tasks prior to rendering and encoding whatever was recorded. This feature enables interesting and novel workflows – the user can record in real time in a very expressive way, and yet, still retain the ability to make precise adjustments to the recorded results. Whilst this feature was initially introduced as an experimental option in LiVES, it is now quite stable and maturing to the point where it can be offered as a mainstream option.

- **Signal/data processing**

This is a relatively recent feature of LiVES, and is one which is traditionally found only in video programming environments. Certain real-time plugins distributed with LiVES can now operate on various types of data – either by analysing video or audio which is fed in to them (for example, motion detector, Fourier analyser), processing data (for example, the data processor plugin), or acting on data (for example, the Vector Visualiser plugin). There is also an interface for creating data connections between plugins, so that the data output of one plugin may be fed automatically into the input of another. There is also support for compound plugins, where various constituent plugins act as a single unit, with data passed internally between them. Several types of variable are supported: boolean, integer, float (double), and string. Variable types are converted automatically (for instance a boolean can be converted to a string with value "0" or "1"). List type variables are also handled, and there is support for 2 dimensional arrays of floats (these are treated as alpha channels). In addition, variables from audio and video filters can be connected together, which allows for greater expressiveness by the user.

- **Support for many types of third-party plugins via its extremely flexible plugin system (LADSPA, frei0r and libvisual)**

LiVES uses an extremely flexible framework, WEED, for real-time effects plugins. Via use of wrapper plugins, it is possible for LiVES to support many other types of plugin without changing them. Using this framework, LiVES will support frei0r video effects, libvisual[50] video generators, and LADSPA audio plugins out of the box. Support for Freeframe, another free plugin framework was considered, and adding this would be possible – however Freeframe supports only 32 bit pointers, and a requirement of LiVES code is that it

[50]LIBVISUAL website. Available at: <http://libvisual.org/>. Accessed: 13 Jun. 2013.

work equally well on 64 and 32 bit systems.

- **OSC control for video editing**

Whilst it is fairly common for VJ applications to include support for OSC control methods, LiVES in addition can make use of it for purely video editing tasks. For example, via OSC, one can send messages like:

/mt/ctrack/set 1	[set the current timeline track to 1]
/block/insert 5	[insert currently selected frames from clip 5]

The set of OSC commands for video editing is currently quite small, but can be easily expanded should further need arise.

- **Bi-directional OSC**

An increasing number of tools in the video space make use of the OSC protocol for control purposes. LiVES takes this one step further and provides bi-directional OSC. For example, an OSC message can be sent to LiVES asking for the number of clips loaded, and LiVES will respond on another, user defined port with the information requested. This enables not just control of the application via OSC, but in fact it is possible to discover the entire state of the application. This allows for far more complex scripts to control LiVES, and a remote application controlling LiVES could present this information graphically using any interface desired. A third port can be used as a listening port; if this is enabled then LiVES will send notification when certain pre-defined events occur, for example when a clip is opened or closed. Responses and notifications are sent as plain text, so a receiving program does need to understand the OSC protocol for parsing responses.

To illustrate this, here is a small set of OSC messages to obtain some data from LiVES.

/lives/open_status_socket 49998	[open a status socket on UDP port 499998]
/lives/ping	[LiVES will return “pong” on port 49998]
/clip/count	[LiVES will return the number of clips open]

/effect_key/parameter/value/get 7 5 2	[get the value of the third element of the sixth parameter of the currently active effect bound to keyslot 7]

- **Flexible and extensible support for external device support (MIDI/joystick mapper)**

The LiVES code makes it a fairly simple task to add new controller type devices whenever such a need may arise. In functional terms, the process as currently implemented works as follows:

- the user activates the MIDI interface on the machine, or connects a joystick
- the user enters the MIDI/joystick learner interface window, via a menu option
- the user touches a MIDI control, or moves the joystick or presses a button
- the interface shows what has been altered (e.g. MIDI controller 12, joystick axis 2)
- the user can select an event type to link this to, for example “Set playback ratio”
- after connecting the controls that the user wishes to uses, the mapping can be saved as a “Device map”. This map may be reloaded at startup time, or later during program execution.
- when the user repeats the action which was “learned”, the appropriate message is constructed and sent internally by LiVES to the OSC controller. The variable part of the control is converted to a variable part of the OSC message. For example, if the user connects a joystick axis to a command, then the axis value is used as a parameter to construct the OSC message.

Internally, what occurs is the following. Each 4 milliseconds, the program checks all device sources to see if there is a message. If there is, then a string of numbers separated by spaces is constructed. The first number is a “magic number” for the device type; e.g. 1 for MIDI, 2 for joystick. The second number is the sub-entry within that device, so for MIDI it might be 1 for controller, 2 for key press, for joystick it might be 1 for axis, 2 for button, and so forth. The third number is the index for that device and sub-entry – so for joystick, button, we would use a value of 0 to represent button 0. The next numbers represent the variable part in the device, so for a joystick button, 1 might be press, 0 might be release. For a MIDI control, it would contain the controller values.

Once we have constructed this string, LiVES does a lockup in an internal table to see if the initial portion of the string matches a message type set by the user. If so, then an appropriate OSC style message is constructed, with the variable parts of the string replacing the variables in the OSC message, which is then sent. LiVES is programmed with the minimum and maximum values for the device controls, and also knows the minimum and maximum values of some of the OSC messages, such as when setting an effect parameter value, so in some cases automatic scaling of the values can be done. In other cases, the user can enter scaling values and offsets for the device input when setting up the mapping.

One nice feature of this method is that the user can select whether a value is constant or variable. Imagine we are generating a string for joystick button 0 press. The string might look something like “*1 1 0 1*”, where the first “1” represents joystick, the second “1” represents button, the “0” is the index of the button, and the final “1” represents “press”. The user can define all of this as constant – so that pressing joystick button 0 would trigger a command with no variable parameters; releasing the button would generate “*1 1 0 0*” which could be mapped to a different command. Alternately, the user could make the last value a variable, so that pressing joystick button 0 would produce a variable with value 1 or 0 depending on whether the button was pressed or released, and hence could be mapped to a command with one parameter. Internally, the difference is that in the first case we can have two entries in our look-up table: “*1 1 0 1*” and “*1 1 0 0*”. In the second case we would have “*1 1 0*” instead. This is the motivation for using strings – when a device change is converted to a string, we only try to match an entry in the table with the initial portion of the string – everything else in the string becomes a variable.

This makes is it fairly easy to add new devices in future to LiVES. All we require is:

– add a new magic number for the device

– enumerate subtypes for that device (“controller”, “key”, “button”, “axis”, etc) and the default number of constants for each sub-type (e.g. 1 for controllers – the index number, 2 for buttons – index number and press/release state)

- set minimum and maximum values for variables for the sub-type (if known), to allow for auto scaling
- some sub-type/index combinations are exceptional – for example a MIDI controller with a certain index number is actually "program change", so this has a special display name (and possibly different minimum/maximum values and number of parameters). These exceptions need to be defined.
- devise a means to convert the device changes to a string "x y z....", where x is the magic number, y is the sub-type, and z is the index value. Everything else follows as parameter values.

LiVES will handle all of the rest, using existing code.

- **Ability to run in "headless" mode**

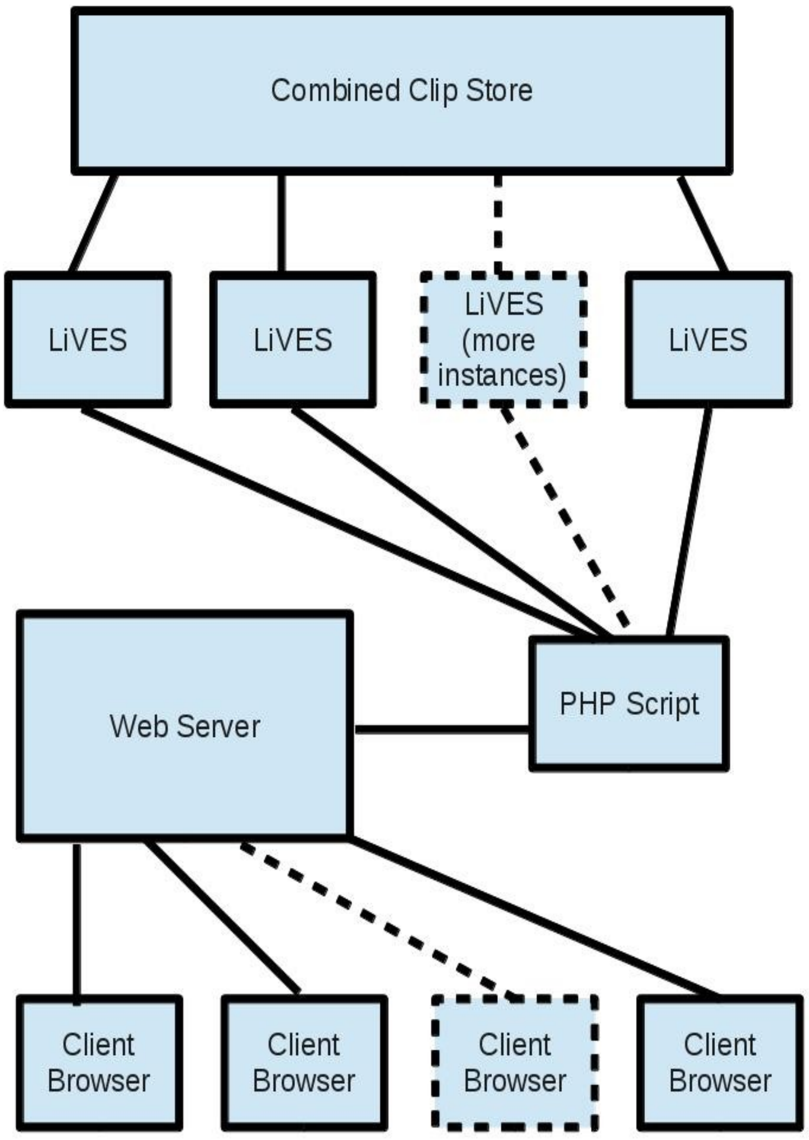
Another fairly unique feature in LiVES is the ability to run in "headless" mode; that is, without using any attached video monitor. For example, one can run LiVES without a user interface and send the output video as a stream to a remote machine. Combining this with OSC control turns LiVES into a true video server. Whilst this feature does exist in some other VJ applications (Veejay for example), and certainly so for video programming environments, LiVES is unique in extending this feature to the range of processing operations which can be carried out whilst operating in this mode. One can perform sequential clip editing operations and lay out an entire multitrack timeline for rendering.

- **HTML/JavaScript interface**

Taking the video server idea a step further, LiVES has an experimental webserver interface. The interface operates via a webserver on the local (or another) machine, and HTML/JavaScript served to client machines. A small PHP script running in the webserver can be used to send and receive OSC messages to and from LiVES, which are then handled by JavaScript functions in the client browser. In this way, it is possible to use LiVES as an online video editing application. Previews of the video being edited are streamed from LiVES back to the client via the webserver. Each client may have its own copy of LiVES running, which uses a unique set of ports for OSC communication. The LiVES instances may connect to the same clip store (for collaborative editing), or else they may have a sandboxed clip store. This idea is still in the prototype phase (due to lack of development time), but a single client version running over the internet has been demonstrated.

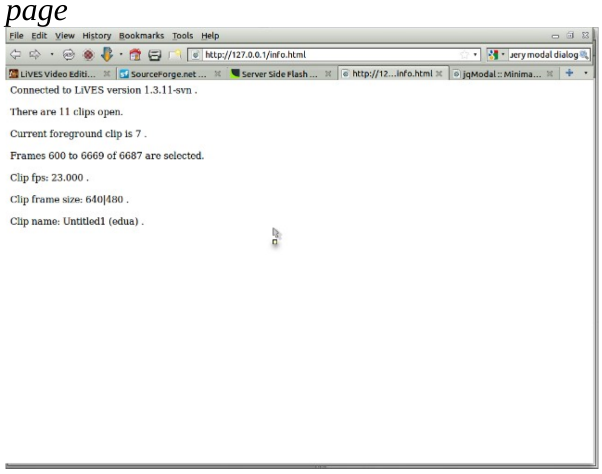
The diagram below (Figure 24) represents a theoretical setup for this.

Figure 24: LiVES setup for an online video editing system



Source: Finch (2013)

Figure 25: LiVES online demo: information page



Connected to LiVES version 1.3.11-svn .

There are 11 clips open.

Current foreground clip is 7 .

Frames 600 to 6669 of 6687 are selected.

Clip fps: 23.000 .

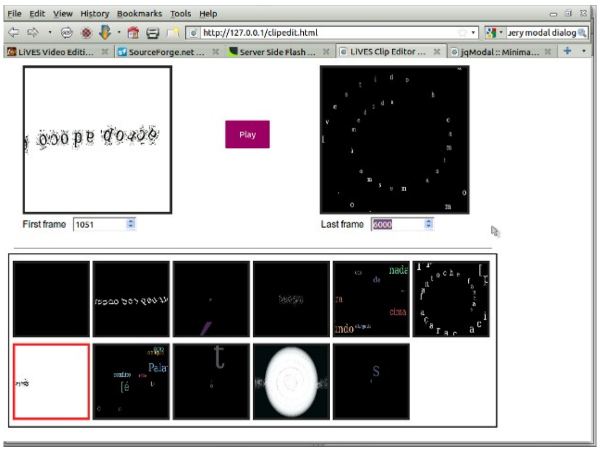
Clip frame size: 640|480 .

Clip name: Untitled1 (edua) .

Source: lives.sourceforge.net

Shown above (Figure 25) is the information page from the LiVES online demo. OSC messages are sent to query LiVES about its current state. The results are formatted and displayed to the end user.

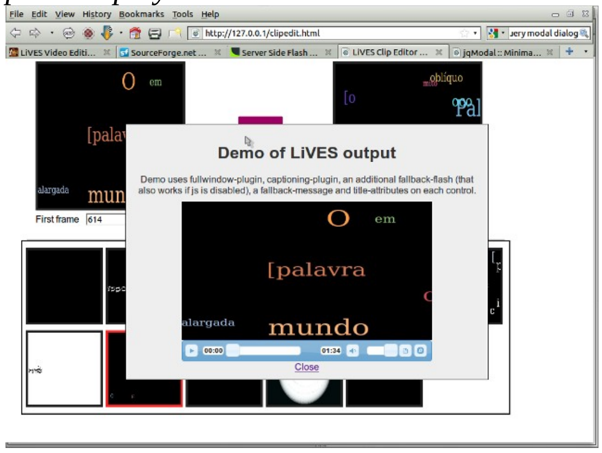
Figure 26: LiVES online demo: clip edit mode



Source: lives.sourceforge.net

In clip edit mode within the online demo (Figure 26), OSC is used to get the current clip index number, and the start and end frame numbers for the current clip. OSC commands are sent which cause LiVES to create thumbnails for for the first and last frames, and representative frames for all other clips. The thumbnails are transmitted to the user via the webserver. Clicking on the spinbuttons for start and end frames, or clicking to select a different clip sends an appropriate OSC message to LiVES. Hovering the mouse over a clip thumbnail shows information about that clip. Since this is a demo, it is not possible to edit the clips here, but in theory this could be implemented using existing OSC messages.

Figure 27: LiVES online demo: streaming preview player



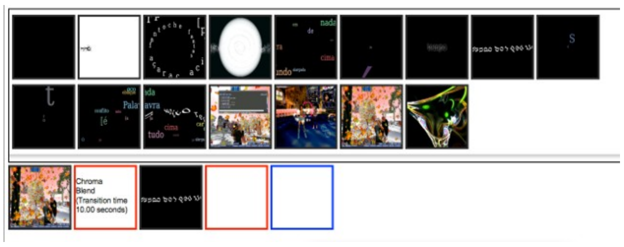
Demo of LiVES output

Demo uses fullwindow-plugin, captioning-plugin, an additional fallback-flash (that also works if js is disabled), a fallback-message and title-attributes on each control.

Source: lives.sourceforge.net

Clicking on Play sends an OSC message which causes LiVES to start streaming the currently selected clip using a playback plugin. The webserver forwards the stream to the user who views it using a JavaScript player (Figure 27).

Figure 28: LiVES online demo: multitrack mode



Source: lives.sourceforge.net

Part of the demo shows the multitrack editor implemented online (Figure 28). A square in the lower area is highlighted in blue. Clicking on a clip thumbnail in the upper area causes that clip to be inserted into the layout via an OSC message. A transition is automatically inserted between adjacent clips in the red square. The transition causes one clip to fade smoothly into the next clip. Clicking on a Play button sends an OSC message for LiVES to start playing a preview, via a streaming playback plugin (exactly as is done in the clip editor). This is a very simplified variant of the multitrack editor in LiVES available through the GUI, designed as a simple proof of concept. The demo could be extended in various ways – clip editing and effects could be added in the browser interface, a more advanced multitrack editor could be constructed, and the user could be provided with a means to render and encode a clip ready for download. This is limited only by the range of OSC messages available in LiVES, which is already very extensive.

In addition to this, the system could allow collaborative online editing – whereby a group of users, perhaps separated geographically, could work together to edit the same material. Each would have access to a combined clip store, and a combined set of layouts for the multitrack editor.

7 **RESULTS AND VALIDATION**

Here we discuss the results of the validation which was carried out in order to determine whether or not LiVES is achieving our aim of meeting the demands of Experimental VJs.

7.1 VALIDATION OF TECHNICAL FEATURES

In response to the questionnaire asking about which features of a video application were important, fifteen replies were received during the time allotted for this, and the results have been collated below in Table 5. Due to the relatively small number of responses received in comparison to the estimated number of LiVES users, these results are not intended to provide any type of formal statistical analysis, but simply to confirm to some extent that the list of features derived earlier matches with the requirements of this set of users and potential users, and to get some idea of the relative ranking of those features amongst the small sample.

It may be noted that none of the respondents selected “Data Connections” as being an important feature. Whilst this is traditionally a feature found in video programming environments, the lack of response may well be due to the fact that this feature is a very recent addition to LiVES and is as yet undocumented. It may also be observed that none of the respondents selected “Local Language Support”. In this case, this result may be due to the fact that the questionnaire was published only in English (which is the default language of the application, mailing list and website). Thus foreign language speakers who may consider this important may have been inadvertently excluded from the survey. These areas may be of interest for further investigation.

Table 5: List of features considered important in a video application, in order of popularity.

Linux Support	13
Real-time Effects	11
Stability	10
Ease of Use	9
Level of Support	6
Ability to Edit Individual Frames	6
Frei0r Compatibility	6
Jack Audio Integration	6
Crash Recovery	5
OSC Support	4
MIDI/Joystick Interface	4
Wide Range of Input Formats	4
Range/Quality of Output Formats	4
Control of Playback at Various Rates	4
Clip Editing Features	4
Price	3
Accessibility of Code	3
Web-cam/TV Card Input	3
Ability to Record Real-time Performances	3
v4l2 Compatibility	3
Multi Monitor Support	3
Multitrack Editing	3
Frequency of Releases/Bug-fixes	2
Fire-wire Inputs	2
Audio Plugins	2
Subtitle Support	2
Themes/Appearance	2

Source: Finch (2013)

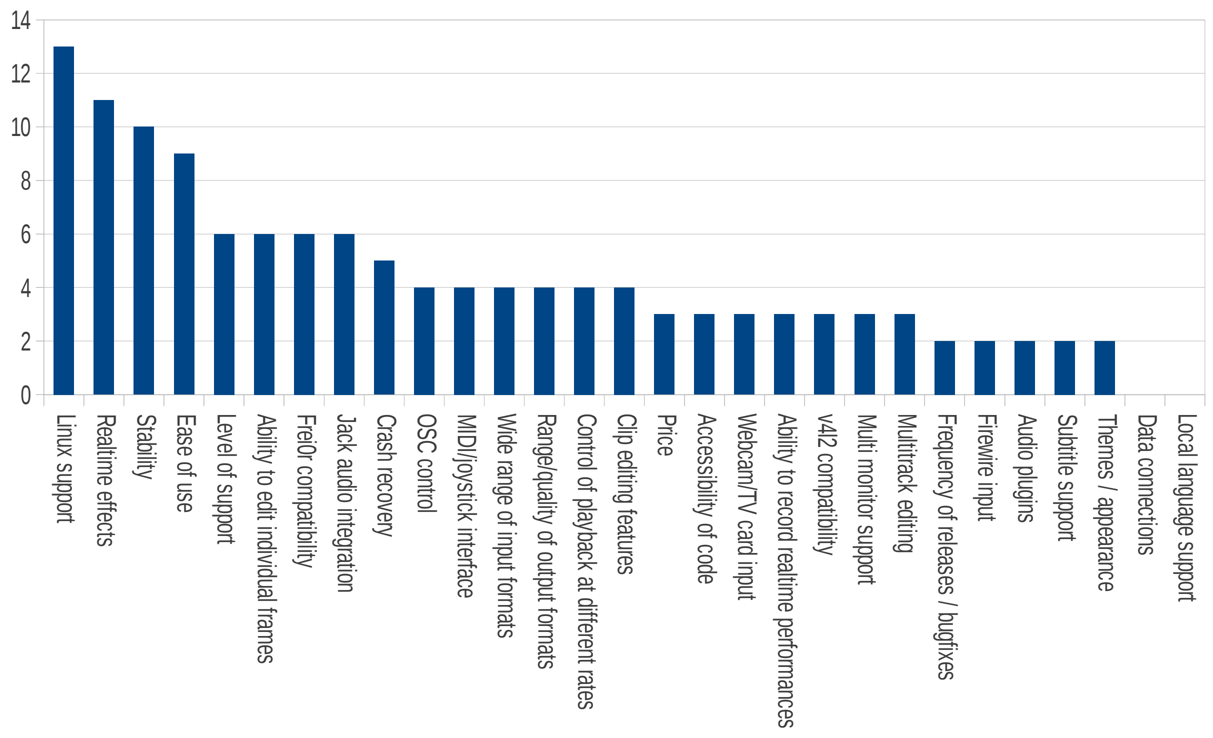


Figure 29: Popularity of various features in LiVES

Source: Finch (2013)

With regard to non-specific technical features, it should be noted that ten of the respondents (66% of the sample) mentioned “stability”, whilst nine (60%) mentioned “ease of use / user friendliness”. Six of the respondents (40% of the sample) stated that “Level of Support” was important to them. Two of the users (13% of the sample) said that “Frequency of releases/bug fixes” was important.

Some of the applications come *close* to meeting all of the users' most important needs – in particular, Veejay and Pure Data. Both of these are feature rich applications – however, Veejay lacks extensive input and encoding options, and does not provide the multitrack composition features which 20% of the respondents required. Pure Data also satisfies many of the requirements; however due to the complexity of the interface and the time needed to learn it, it does not satisfy particularly well the requirement of “ease of use”.

Comparing the feature list of LiVES[51] with the list of technical requirements (which has been reproduced below from the current LiVES website), we can see that LiVES satisfies all of them. (Items marked in bold represent matches with the requirements list.)

Platform

- Stable and well tested core.
- Fully cross platform for GNU/Linux and many flavours of Unix (e.g. BSD, openMosix, IRIX, OSX/Darwin, Solaris).
- Runs on at least x86, amd64, ppc and xbox/x86.
- The LiVES audio/video platform is custom extendable through RFX plugins.
- Allows quick and easy prototyping of new tools, utilities, effects, transitions, generators and more, using the included RFX builder window.
- Plugins can be written in Perl, C, C++, python, or any other language, allowing O/S level access to individual frames within clips.
- Will scale for high/low end hardware. Can be split into client/server components.
- **Control every function in LiVES remotely using OSC protocol.**
- Packages for most major GNU/Linux distributions: Ubuntu, Mandriva, Gentoo, Debian, Suse, Fedora Core, Rock Linux, Source Mage, Alt Linux, Frugalware and Dynebolic.
- 100% original, non-proprietary code.

Video

- **Loading and editing of almost any video format**
- Some formats can be opened instantly using decoder plugins (e.g. webm, wmv, flv, dv

[51]FINCH, G. LiVES feature list. Available at: <http://lives.sourceforge.net/index.php?do=features>. Accessed: 13 Jun. 2013.

and ogg/theora).

- **Smooth playback at variable frame rates, forward and in reverse.** Display framerate can be controlled independently of playback framerate.
- **Frame accurate cutting and pasting within and between clips.**
- **Saving/re-encoding of clips, selections, and individual frames.**
- Lossless backup/restore.
- Streaming input and output.
- **Support for live firewire cameras and TV cards**
- **Real time blending of clips** (various chroma and luma blends).
- Can handle in/out streams in LiVES to LiVES or yuv4mpeg format. Streams can be piped from stdout into other applications.
- **Supports fixed and variable framerates.** Playback rate can be smoothly adjusted independent of display rate.
- **Ability to 'scratch' with video - that is to move smoothly backwards and forwards through it, and to record yourself doing so.**
- Playback can use LiVES' own internal player, there is also a high performance fullscreen SDL playback plugin, and a new openGL playback plugin
- Internal support for RGB24, RGBA32, YUVA, YUV, YUV422, YUV420 (jpeg and mpeg), YUYV, YUV411, and UYVY palettes; one step conversion with chroma super and subsampling is implemented.
- Clamped and unclamped YUV is supported.
- Ability to edit many filetypes and sources including remotely located files (with mplayer/ffmpeg libraries), and directories of images.
- Real time capture/recording of interactive (via mouseclicks) external windows.
- **Encode to any of the 50+ output formats which are now supported** (e.g. mjpeg, mpeg4, mpeg1/2, h264, VCD, SVCD, DVD, ogg/mp4 ogm, Matroska mkv, dv, swf, Ogg Theora, Dirac, MNG, Snow, xvid, and even animated GIF and PDF!)
- Encoder formats can easily be extended through the encoder plugin API.
- LiVES will suggest the best settings for saving to each format.
- Resampling of video (time stretching) to any frame rate (1 to 200 fps - accurate to 8 decimal places); option to auto-resample or speed up/slow down between clips.
- Ability to instantly alter the playback speed of video and audio independently.
- **Rotation, resizing and trimming of video clips.**
- Deinterlacing, subtitle removal. Auto deinterlacing for dv can be enabled.
- Instant saving/loading of clips for performances/presentations.

Audio

- Can load mp3, vorbis, mod, it, xm and wav files.
- LiVES can also load tracks directly off CD to use with your video (using cdda2wav).
- Ability to save audio selections, and append audio.
- Sound can be trimmed to fit video selections.
- Sample accurate cutting and pasting of audio within and between clips.
- Resampling of audio (rate, channels, sample size, signedness and endianness); audio is auto-resampled between clips.

- Supports (auto)inserting of silence and deletion of audio sections.
- Able to record from any external audio source.
- Fade in/fade out feature for clips.
- Audio speed and direction can be smoothly adjusted; both in real time and when rendering.
- **Support for LADSPA audio plugins.**

Effects/Transitions

- Many effects, including random/targeted zooming, panning of video, colour cycling and colorisation/colour filtering.
- Merging/compositing of frames is possible: e.g. frame-in-frame, fade in/out and transparency.
- Real time previews as the effect is processing.
- **Support for the Frei0r effect plugin architecture** (via a wrapper) which will allow sharing of real-time effects with other applications.
- Use real time effects to blend clips together, regardless of frame size or fps. Luma and chroma blending are currently supported.
- **Multiple real time effects are possible during playback** (VJ mode), these can also be rendered to frames.
- Effects and transitions are now fully customisable using the RFX builder window.
- Effects/blends can also be applied to incoming streams in real time.
- Dynamic loading of effects.

Multitrack

- **Multitrack window with drag and drop**
- Intelligent screen organisation - shows you only the information which is relevant, no more and no less
- Support for an almost limitless number of tracks and effects
- Rapid rendering - resize/resample and effects apply done in a single pass
- Tracks can be laid out entirely with keyboard, or with mouse, or a combination of both
- Multitrack settings can be targeted for a specific encoder, or generic
- Layouts can be saved and reloaded
- Audio blocks can be timestretched and even reversed
- Non-destructive editing, with multiple levels of undo/redo.
- Full automation of effect parameters.
- Any number of layers can be composited together into a single layer.
- Support for stereo backing audio track + stereo audio track per video track
- Automatic gain control
- real-time mixing/previewing of audio
- Channel mixer volume control + fine grained, time variable per-channel volume and pan control.
- Auto-transitioning of audio with video.

Extras

- **Full crash recovery.**
- **Configurable multi-monitor screen placement.**
- Simple and intuitive menu layout.
- Remote monitoring of the application can be enabled
- VJ functions can be controlled via keyboard, **joystick or MIDI controller**
- I18N text support. Translations into French, Czech, German, Japanese, Dutch, Portuguese, Spanish, Italian, Russian, Turkish, Hungarian, Slovak, Simplified Chinese, Finnish, Brazilian Portuguese, Ukrainian, Uzbek, Polish, Uyghur, Catalan, Galician, Romanian, Estonian, Telugu and Hebrew are included.
- Pulse audio support.
- **Support for audio output through jack.**
- **Jack transport support (master or client)**
- **Support for .srt and .sub subtitle formats.**
- Import tool for importing clips directly from YouTube.
- Full integration with upcoming videojack standard (work in progress)
- RFX builder allows rapid prototyping of new effects, transitions, generators, utilities and tools. Custom RFX scripts can be exported to share with others or downloaded and imported. Test scripts are run in a sandbox to allow safe testing of new plugins.
- Midi sequence synchronisation (start/stop).
- Can load single images or directories of images in numerical order and assemble them into videos or slideshows.
- Ability to play music through xmms (including random selection of tracks).
- **Shuttle controls for firewire cameras/recorders. Can grab from DV and HDV formats.**
- Can stream out (on stdout) using yuv4mpeg format.
- **Can play back through a vloopback device** (Linux only).
- Project files (clips and layouts) can be exported and imported
- Toys!

GUI

- Based on gtk+ 2.16+, runs under KDE, Gnome, Metacity, Fluxbox, Compiz and any known window manager.
- Several built in themes/skins available (see screenshots). Custom themes will be supported soon.

7.1.1 **Qualitative responses**

In addition to the survey, a few users were interviewed and asked to provide example use-cases based on what they do with video. These types of use cases are important in order to better comprehend the combinations of features which a typical user might employ in their workflow. Such understanding is valuable in order to simplify these workflows and to help ensure that all of the steps contained in them are available within the application.

Furthermore, a paper published by IBM (Use Cases, Best Practices, 2003) states:

> *Functional requirements are typically written from the point of view of the software, but use cases are written from the 'voice of the customer.' This expression, which comes from the quality movement, refers to discovering the stated and unstated requirements of product customers and users. Building a software product without understanding their needs is a sure path to failure. Use cases are arguably the best requirements technique we have for describing 'the voice of the customer' in software products.*

- **Use Case 1**

One user outlines his need "t*o be able to save a 30 second video (created with a camera) and be able to review it frame by frame with another piece of software.*"

The features required for this task include:

– Import video from a camera (either via firewire or by decoding the video saved on disk)

– Shuttle control of the camera (possibly) to locate the desired segment

– The ability to review the selected clip frame by frame

– The ability to save a selected frame as an image format, in order to allow it to be imported into an external application for further processing

- **Use Case 2**

Another states: "*I need a full featured, but easy to operate, open source video editor that I could adapt to accept my new camera's slightly funky video format.*"

This user mentions other features:

- feature completeness
- ease of use / user-friendliness
- open source code
- adaptability of the code
- variety of input formats to handle many types of camera input

• **Use Case 3**

A third user provided the use-case: “*...improvise with live video, like randomly switching clips with music, add some effects and record the improvisation to a new clip.*” The features involved here are:

- real time playback
- real time effects
- real time performance recording of video, audio and effects and clip switches
- rendering of the performance to a new clip
- (possibly) followed by encoding of the new clip

Note that whilst each of these tasks could be handled separately by one or more of the programs mentioned in Chapter 2, there is *no one program* there which could handle *all* of these use cases together. If these use cases were the *only* requirement for that user then depending on the task in hand, they could possibly use one of the programs above. However, if they had other requirements in addition to the use cases mentioned, then they would quite possibly need to use a second or a third application to complete all of their tasks. LiVES is the only tool which can cover all of these use cases and more.

Fourteen of the participants responded to the question asking which features of LiVES they found important. If we examine only the specific technical features, we can divide these up roughly into features commonly found in video programming environments, video editing applications, and VJ applications. The responses are shown in Table 6 below:

Table 6: Cross reference of LiVES features

VPE		VIDEO EDITOR								VJ APPLICATION									
OSC		AR	ME	SS	EF	FW	CE	IF	OF		VR	RE	TV	AP	FC	V4	MJ	MM	JA
					*		*	*	*		*								
												*					*		
		*	*	*		*		*	*		*	*						*	*
		*		*	*		*					*			*			*	*
												*						*	
*								*			*	*			*				*
*								*				*			*				
															*				*
			*		*		*		*			*							
		*	*				*				*	*	*	*	*	*	*		
								*				*					*		
*					*			*	*			*	*			*			
					*									*					*
*						*						*	*		*	*	*		*

Source: Finch (2013)

As can be seen, there is a lot of overlap between the three application areas – in most cases the respondents marked features in two or more application areas, and in four cases they marked features from all three application areas. If this is a representative sample of all LiVES users, then this would suggest that the LiVES application is fulfilling a unique role for the majority of users.

Table 7: Key for Table 6

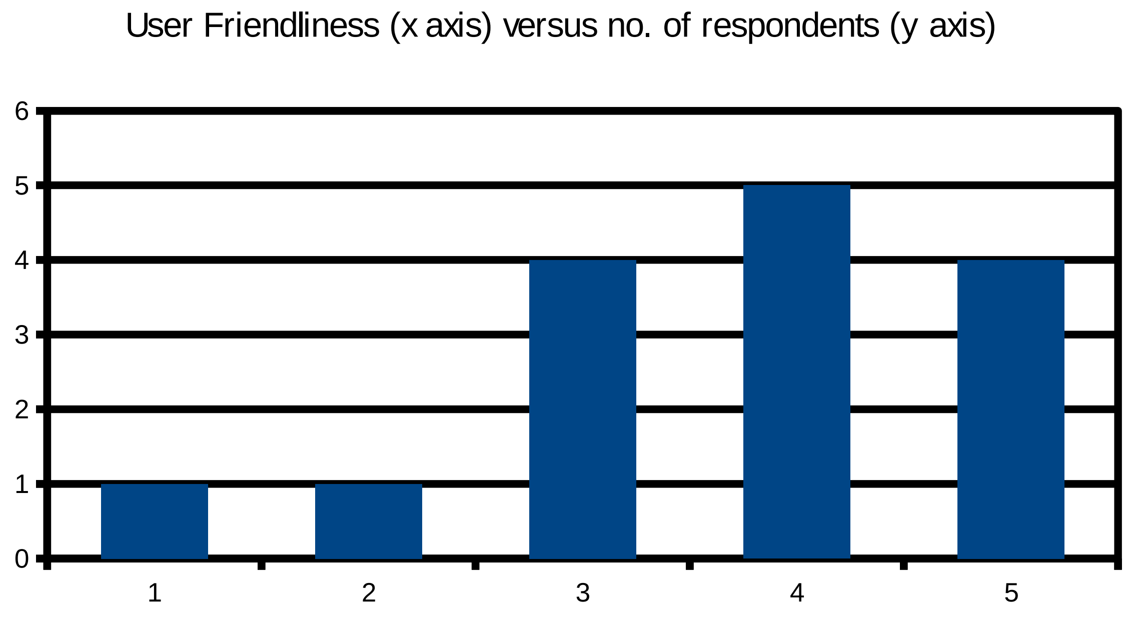
VPE	Video Programming Environment		
OSC	OSC support		
RP	Ability to record performances	ME	Multitrack editor
SS	Subtitle support	EF	Edit individual frames
FW	Firewire support	CE	Clip editor
IF	Wide range of input formats	OF	Range/quality of output formats
VR	Playback at variable rates	RE	Real-time effects
TV	TV card / webcam support	AP	Audio plugins
FC	Frei0r compatibility	V4	Video for Linux support
MJ	MIDI/ Joystick support	MM	Multi-monitor
JA	Jack audio integration		

Source: Finch (2013)

7.2 VALIDATION OF NON-SPECIFIC REQUIREMENTS

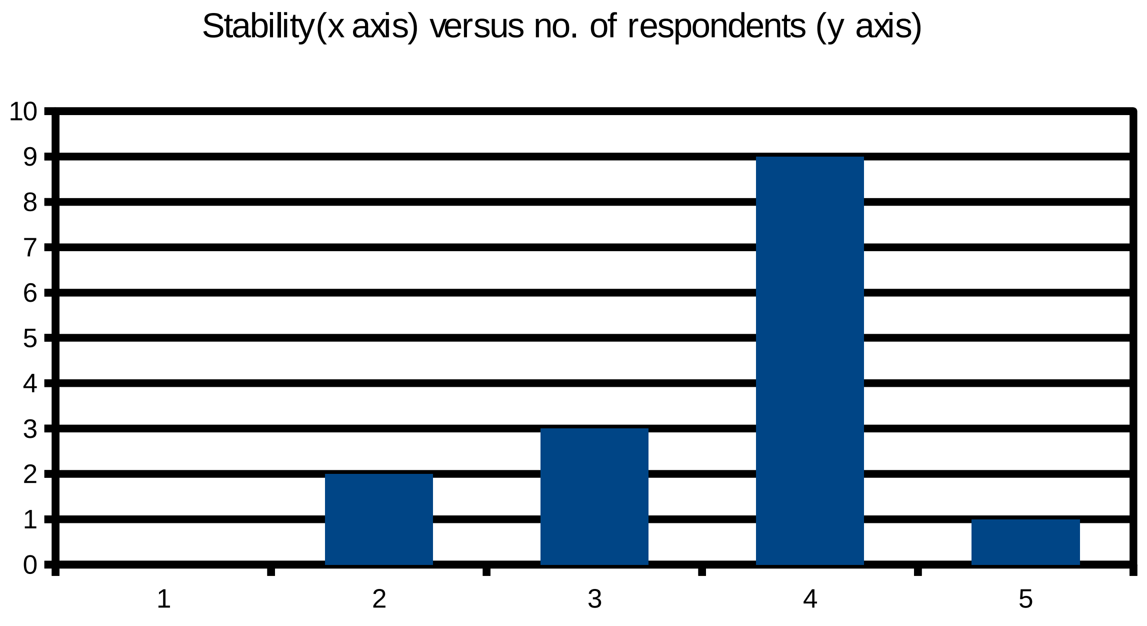
The user ratings for non-specific features were collated and are show in the charts below (Figures 30 – 34).

Figure 30: User Friendliness rating for LiVES



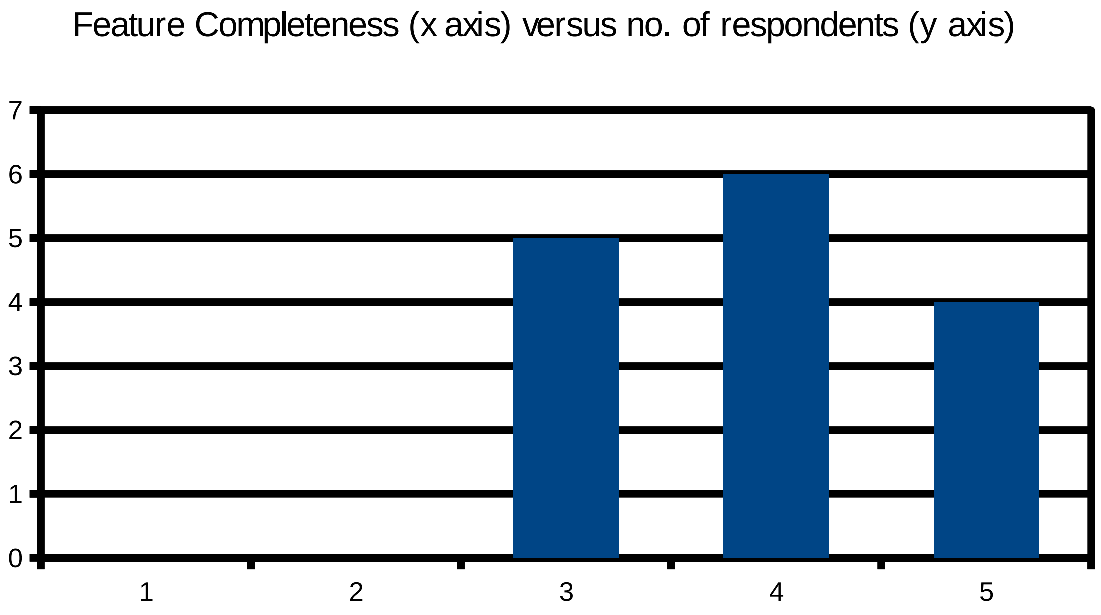
Source: Finch (2013)

Figure 31: Stability rating for LiVES



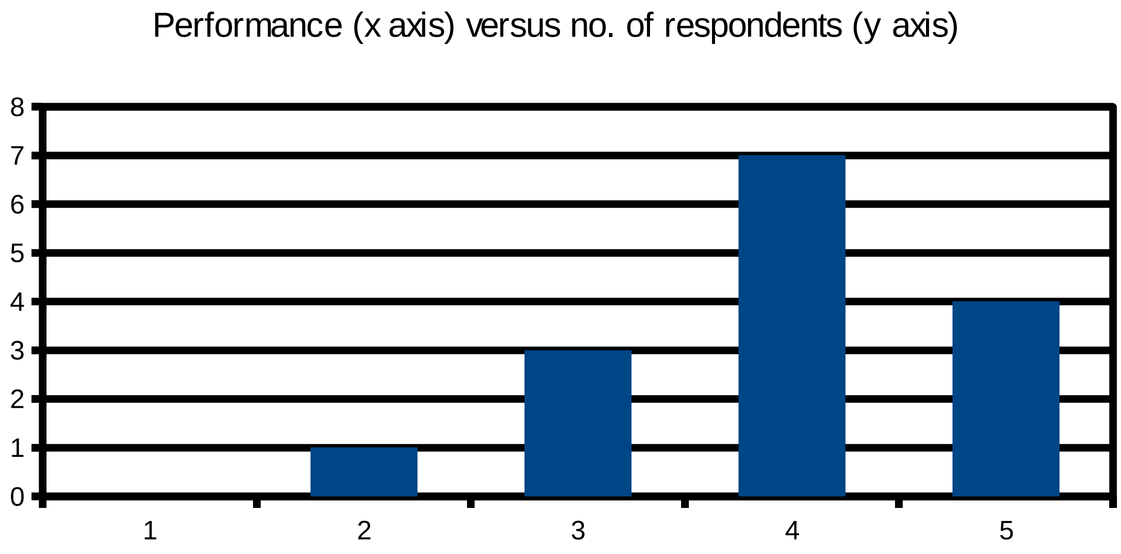
Source: Finch (2013)

Figure 32: Feature Completeness rating for LiVES



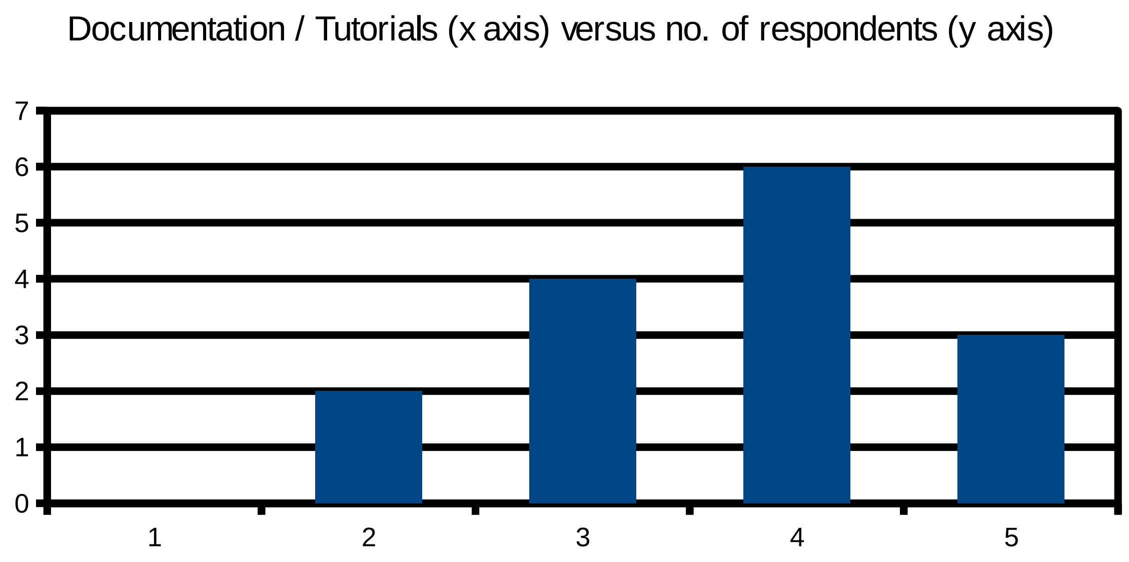
Source: Finch (2013)

Figure 33: Performance rating for LiVES



Source: Finch (2013)

Figure 34: Documentation / Tutorials rating for LiVES



Source: Finch (2013)

As can be seen, according to the results received, LiVES was rated well above average in all categories.

Some users also provided more detailed feedback; for example, "*I prefer to work in a Linux environment, and LiVES is simply the best video editor around in this environment.*", "*I like the logic in LiVES which makes it feel fairly intuitive.*", "*I am not a VJ, but I love the VJ features. It makes LiVES even more fun to play around with - and boosts my creativity.*", "*LiVES is fast. In my normal environment it is much faster to work with than any other video editor I have available or have access to.*", "*The GUI is simple, yet complete. Lives does all I need it to.*"

7.3 RESULTS OF THE THIRD SURVEY

A third question asked the respondents: "*What new features would you most like to see developed in LiVES*". One user stated that they would like to see a graphic equaliser for audio. In fact, this feature is already available through the use of LADSPA effects in LiVES, so perhaps better documentation would be helpful here (this is a newer feature which is not mentioned in the manual).

Another replied: "*Multiple streams / engines running through one interface, each engine with its own clip set, video out device and OSC port with a view window that show each stream.*". Again this feature is already available to a large extent simply by running multiple copies of LiVES simultaneously.

A third user responded: "*audio reactive real-time effects*". This is *somewhat* possible now in LiVES using data connections – LiVES has a "beat detector" audio plugin and the output of this is available for use. A missing component would be a plugin to accept this output and enable/disable various other effects and parameters in real-time. It can certainly be connected to individual effects to produce some interesting results. Again, this is a newer features which has not yet been well documented.

A fourth user mentioned a desire to see video mapping implemented in LiVES. This is a difficult area not so much from a technical point of view but rather in terms of creating an interface. However, this is an area which could be explored in the future.

7.4 RESULTS FROM DISCUSSIONS WITH SPECIALISTS

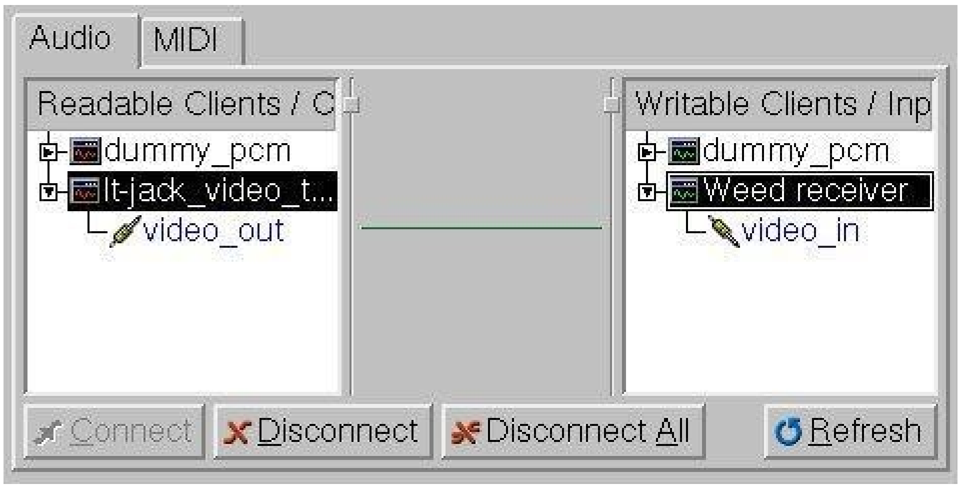
From discussions at the Piksel conferences, it was clear that three areas would benefit users greatly from collaborative, open development amongst various Free Software packages. These areas were: a framework for sending and receiving video frames in real-time between various applications; a common set of video effects which could be used by any application which wished to do so; and a common syntax for controlling programmatically various video applications. The first of these areas remains to be fully implemented, however, the latter two have been incorporated within the LiVES application.

- **Sending/receiving frames between applications**

The idea here was to have the ability to send video frames from one application to another in real time, as was already possible with audio using Jack. Users would benefit from having far greater flexibility in the way they worked with multiple tools. The tool would need to be cross platform since the idea was to benefit as many users as possible. There exist some tools to do this already, such as gstreamer[52] and v4l, but each has drawbacks. Gstreamer is heavily linked with the gnome window manager for Linux, and the developers felt that it would be preferable to have an independent framework. V4l works well in some circumstances, but it will only work on Linux, and since it is a kernel module it requires some technical knowledge to set up.

Several developers expressed an interest in developing a version of Jack for video, and work was begun on this. After the original developers seemed to lose interest, the author himself picked up this project and rewrote much of the code, and attempted to publicise this project, updating the videojack server code. The author also created demonstration videojack sources and sinks for Pure Data, but due to unfamiliarity with the internals of that application, was unable to implement coding and decoding of video frames. Attempts to get assistance from the Pure Data community proved fruitless. In addition, keeping pace with the changes in the audio version of Jack was too much effort, and the Jack developers were understandably unwilling to commit an untested patch to the main branch of the audio sever. Despite this, LiVES still supports videojack input and output and the code for videojack is still available[53]. Figure 35 shows routing of connections via videojack being between two copies of LiVES, using the standard qjackctrl application which has not been modified in any way.

Figure 35: Videojack routing video frames from one copy of LiVES to another



Source: Finch (2013)

[52]GSTREAMER website. Available at: <http://gstreamer.freedesktop.org/>. Accessed: 13 Jun. 2013.
[53]PROJECT page for videojack. Available at: <http://sourceforge.net/projects/videojack/>. Accessed: 13 Jun. 2013.

- **A common framework/repository of video effects**

A second area of collaboration which was discussed was video effects. The aim here was to create a common set of cross platform video effect plugins, in much the same way as audio developers had done with LADSPA. The original name of the project was LiVidO – Linux Video Objects. Users would benefit since rather than each project creating their own set of plugins, all developers would be contributing to a common base. From an early version of the LiVidO idea emerged frei0r. Eventually LiVidO itself was all but abandoned due to disagreements amongst the developers regarding the final specification - only Veejay currently implements what became the final LiVidO spec. The author himself reverted to the penultimate version of the spec, and made some enhancements to produce the WEED framework for LiVES. The frei0r project is still doing very well, with now well over a hundred effects. The WEED library has had only a dozen or so minor changes in the seven years since its conception, indicative of the excellence of the original collaborative design.

Figure 36: Humourous flyer for a development conference / presentation, one of the events where the author and others worked on the LiVidO spec.

FROM TOSTI WITH LOVE
WITH LIVIDO DEVELOPERS:
NIELS "CONTRØL" ELBURG
VEEJAY.SF.NET
Veejay is a visual instrument and realtime video sampler
GABRIEL "SALSAMAN" FINCH
LIVES.SOURCEFORGE.NET
LiVES is a Free, Open Source video editor "and" a VJ tool.
BECAUSE THE MEDIA SHOULD BE OPEN... BECAUSE THE MEDIA SHOULD BE FREE...
DENIS "JAROMIL" ROJO
FREEJ.DYNE.ORG
FreeJ is an instrument for realtime manipulation of various video layers combined with live effects.
Livido Opensource Video Effects
10/07/2005 from 14.00 Hrs >
PRESENTATIONS: DYNEBOLIC, FREEJ, LIVES, VEEJAY, MUSE, HASCIICAM, LIVIDO
TOSTI
THE OPEN SOURCE TEST INITIATIVE
TOSTITILBURG.NL
OPENINGSTIJDEN di t/m zo 14.00 - 17.00 wo 19.00 - 21.00 vrij 21.00 - >
SPOORLAAN 340

Source: TOSTI TILBURG (2005)

- **A common syntax for controlling video applications**

The third area which was discussed as was thought to be useful to users was the development of a common syntax for controlling video applications. Since a few video applications were starting to implement OSC as a control protocol, the developers considered that the best way to do this would be to work out jointly a common syntax to fit on top of OSC. This initiative partially achieved its aim – a larger array of video applications do now implement OSC. The author continued to discuss this initiative on the OSC mailing list; however, interest for the common syntax dropped amongst other developers. Nevertheless, it inspired the author to arrange and improve the OSC syntax within LiVES[54].

[55]

7.5 RESULTS OF LIVES DEVELOPMENT

Here we examine some metrics which may help to illustrate the impact of LiVES on the user community.

7.5.1 **Estimates of the number of users**

Obtaining an accurate figure for the number of LiVES users is difficult, since users are not required to register in any way in order to download or use the application. In addition, there are several sites which mirror the LiVES source code, and binary versions are packaged for various Linux distributions.

However, there are some methods which can be used to make an educated guess.

[54]LiVES OSC commands. Available at: <http://openmediacontrol.wetpaint.com/page/LiVES+commands>. Accessed: 13 Jun. 2013.

[55]As a personal note, the author believes it regrettable that later iterations of the Piksel Festival have been focused almost exclusively on performance and very little on development. The early Piksel gatherings were a great resource and a meeting of minds for Free Software video developers.

- **Debian popcon**

The Debian distribution packages new releases of LiVES, and the distro also contains a tool called popcon (short for “popularity contest”). By means of this tool, users may inform the Debian developers of which packages they have installed, and which of those they use on a regular basis.

The popcon page for LiVES may be found easily by searching the popcon site[56]. Here we can see that (at the time of writing), LiVES is installed on 0.61% of all machines as registered by the popcon tool. Recent versions are installed on 0.56% of machines, with 0.07% of respondents using it regularly (the “vote” score).

The total number of Debian users is not known, but a mailing list thread[57] suggests a figure of between 360,000 and 1,000,000. If we assume a figure of 500,000, and that popcon represents a typical sample of Debian users, this would signify about 1,000 LiVES users for Debian, with perhaps 100 regular users.

Although Debian is a very popular Linux distro, it is only one of many. Assuming it represents around 10% of the Linux (the table at the right on Distrowatch[58] suggests this to be about right), and making the assumption that Debian is typical of all Linux distros, this would produce a figure of around 10,000 users, with 1,000 of those using it on a regular basis.

We have made some assumptions here: namely that popcon represents a typical sample of Debian users, that Debian is a typical Linux distro, and that everybody who has LiVES installed uses it, even if only occasionally. Nonetheless, this should be sufficient to provide a very rough estimate.

However, this figure only accounts for binary packages of LiVES. Many LiVES users prefer to download the source code and compile it themselves. To obtain an estimate for these additional users, we need to use other means.

- **The LiVES homepage**

The main LiVES site registers around 3,000 downloads of source code per month,

[56]POPCON web page for LiVES. Available at: <http://qa.debian.org/popcon.php?package=lives>. Accessed: 13 Jun. 2013.
[57]FORUM post discussing estimates of user count for Debian. 2010. Available at: <http://forums.debian.net/viewtopic.php?f=10&t=56773>. Accessed: 13 Jun. 2013.
[58]DISTROWATCH website. Available at: <http://distrowatch.com/>. Accessed: 13 Jun. 2013.

during those months where there is a new release. In addition there are various other sites which also host the code – for example, a copy of the code is also available for download directly from Sourceforge. A typical month might register about 400 - 800 downloads.[59]

An interesting statistic would be to look at the cumulative number of downloads from one release to the next. Unfortunately such historic data is not collated. But, for example, for the time period 1st May 2013 to 27th May 2013, there were 2,122 downloads of all versions of LiVES for the sourcecode linked from the main LiVES website[60]– updated monthly. There was no new release that month. Adding this to the approximately 3,000 downloads in April (the month in which the release was updated) gives a running total of over 5,000 source code downloads. Presumably there would be further downloads in June, July, etc. Adding in the figures for Sourceforge gives an additional 730 downloads for a grand total of around 6,000 source code downloads. Naturally, one cannot assume that each person who downloads the code actually compiles it and goes on to become a user, so therefore this figure should be considered an upper estimate.

Nevertheless, taking all of the above figures into account, a rough estimate of the number of LiVES users would be 10,000 – 20,000 with around 1,000 – 2,000 of those using it on a regular basis, and around 100 – 200 using it very regularly.

7.5.2 **Website visitors**

According to Google Analytics, for the month of April 2013, the number of visits to the LiVES homepage[61] was measured at 10,774; of these, 9,455 were unique (non-repeat) visitors. 88% of these visitors were new visitors, whilst 12% had visited the site previously (Figure 37). The real figure is likely higher than this, since many browsers block Google Analytics trackers. From the download statistics we know that about one third of those visitors downloaded the source code.

The majority of visits were from Italy, with the USA in second place. This is followed

[59]LIVES download statistics, Sourceforge. Available at: <https://sourceforge.net/projects/lives/files/stats>. Accessed: 13 Jun. 2013.
[60]LIVES download statistics for the current month. Available at: <http://salsaman.home.xs4all.nl/statistics.html>. Accessed: 13 Jun. 2013.
[61]LIVES website. Available at: <http://lives.sourceforge.net>. Accessed: 13 Jun. 2013.

by visits from Germany, and the UK (Figure 38). In just the twelve days from 26 May 2013 until 7 Jun 2013 the website received visits from no fewer than 128 countries.[62]

[62]ANIMATED globe showing LiVES visitor locations. Available at: <http://www.revolvermaps.com/?target=enlarge&i=220i2g5eeou&color=ff0000&m=0&ref=null>. Accessed: 13 Jun. 2013.

Figure 37: Worldwide visitors to the LiVES website for April 2013

Google Analytics

Go to this report

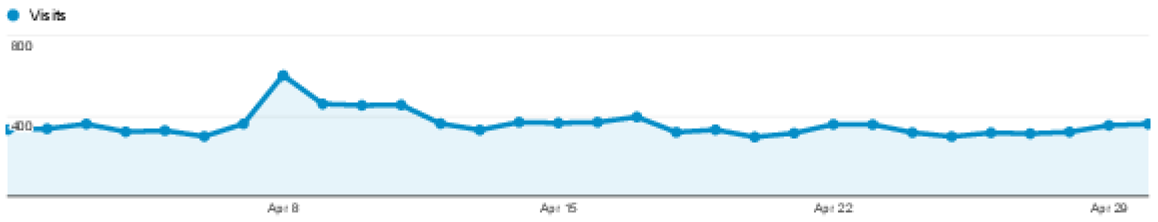
http://lives.sourceforge.net - http://lives.sourceforge.net
lives.sourceforge.net [DEFAULT]

Traffic Sources Overview

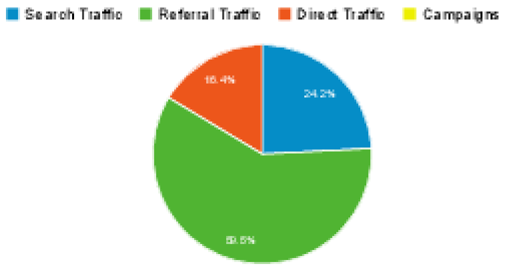
Apr 1, 2013 - Apr 30, 2013

● % of visits: 100.00%

Overview



10,774 people visited this site

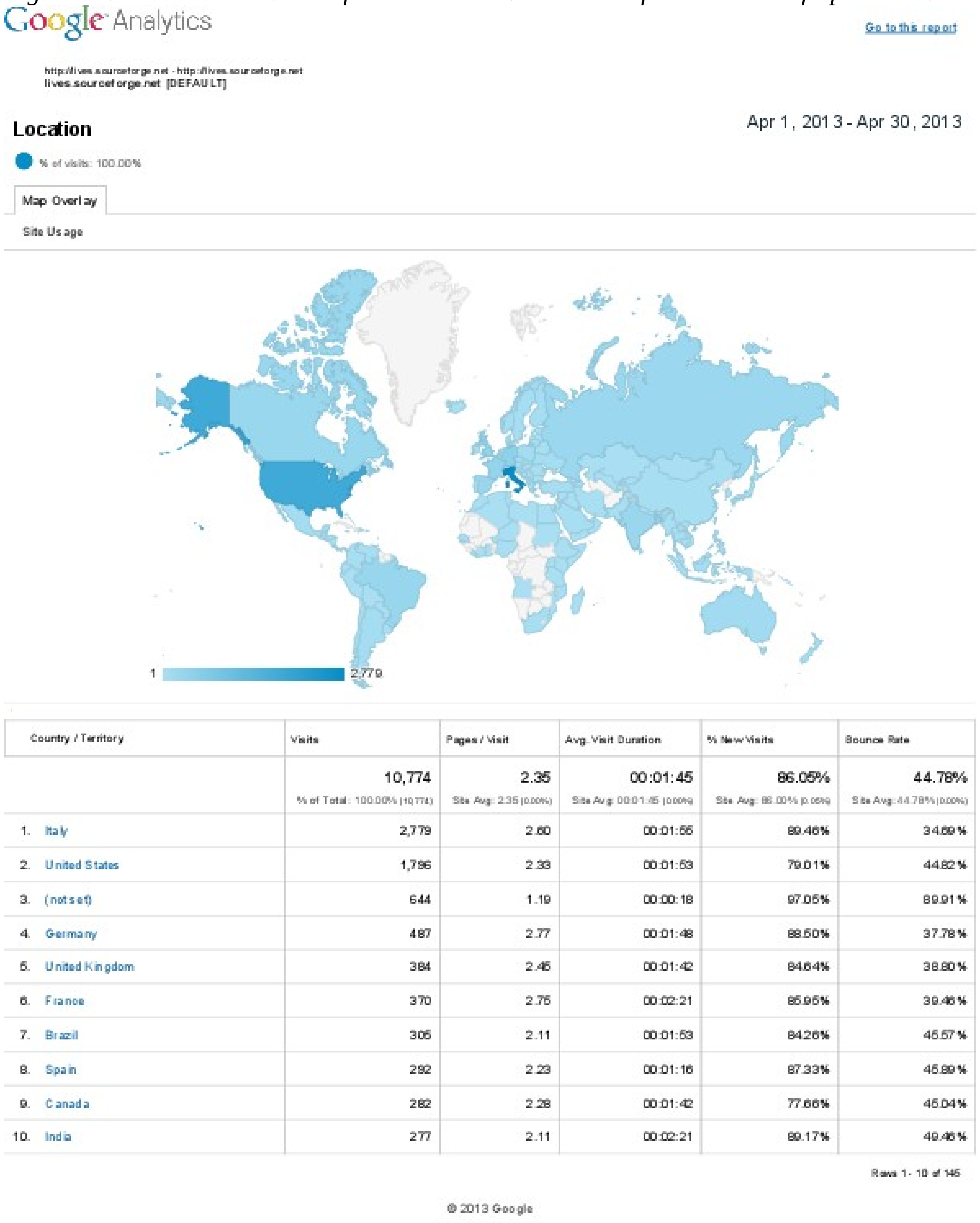


	Keyword	Visits	% Visits
1.	(not provided)	1,404	53.92%
2.	lives	498	19.12%
3.	lives video editor	64	2.46%
4.	open source video editing	22	0.84%
5.	open source video editor	19	0.73%
6.	lives download	18	0.69%
7.	live's	15	0.58%
8.	i lived download	12	0.46%
9.	lives linux	11	0.42%
10.	linux lives	9	0.35%

view full report

Source: Google Analytics

Figure 38: Global distribution of visits to the LiVES website for the month of April 2013.



Google Analytics

Go to this report

http://lives.sourceforge.net - http://lives.sourceforge.net
lives.sourceforge.net [DEFAULT]

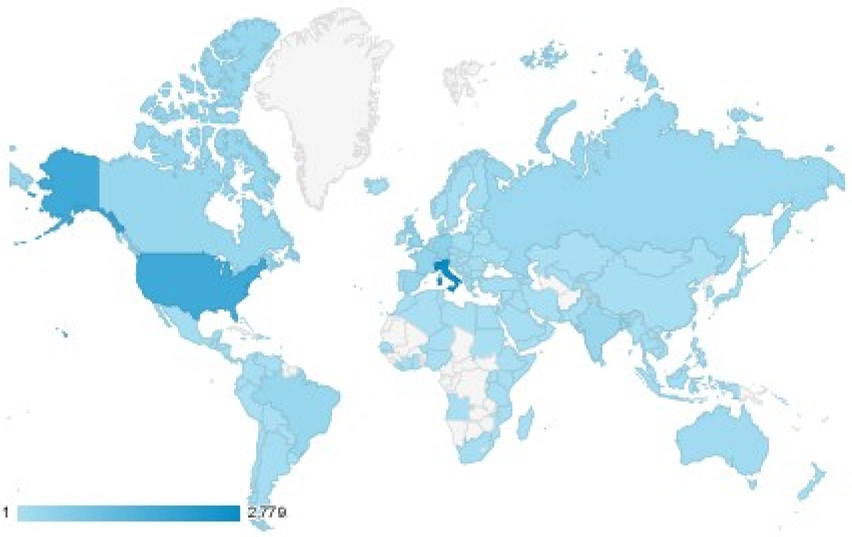
Location

Apr 1, 2013 - Apr 30, 2013

● % of visits: 100.00%

Map Overlay

Site Usage



	Country / Territory	Visits	Pages / Visit	Avg. Visit Duration	% New Visits	Bounce Rate
		10,774 % of Total: 100.00% (10,774)	2.35 Site Avg: 2.35 (0.00%)	00:01:45 Site Avg: 00:01:45 (0.00%)	86.05% Site Avg: 86.00% (0.05%)	44.78% Site Avg: 44.78% (0.00%)
1.	Italy	2,779	2.60	00:01:55	89.46%	34.69%
2.	United States	1,796	2.33	00:01:53	79.01%	44.82%
3.	(not set)	644	1.19	00:00:18	97.05%	89.91%
4.	Germany	487	2.77	00:01:48	88.50%	37.78%
5.	United Kingdom	384	2.45	00:01:42	84.64%	38.80%
6.	France	370	2.75	00:02:21	85.95%	39.46%
7.	Brazil	305	2.11	00:01:53	84.26%	45.57%
8.	Spain	292	2.23	00:01:16	87.33%	45.89%
9.	Canada	282	2.28	00:01:42	77.66%	45.04%
10.	India	277	2.11	00:02:21	89.17%	49.46%

Rows 1 - 10 of 145

© 2013 Google

Source: Google Analytics

7.5.3 **The lives-users mailing list**

This is the main form of communication between LiVES users and the developer(s). The mailing list has currently 143 members, which is consistent with around 1% of all users or 10% of regular users given the estimates above – this does not seem an unreasonable value.

7.5.4 **Reviews in the media**

LiVES has received a few reviews in the media, and these have been overwhelmingly positive.

The online journal Linux Insider (a subsidiary of ECT News Network, one of the largest e-business and technology news publishers in the United States), released an article in 2012 entitled “LiVES: A Rich Video Editor With Layer Upon Layer of Features” in which the author writes:

> *LiVES is an advanced video editor that can double as a video jockey (VJ) tool. It is surprisingly powerful. But its interface makes it rather simple to learn. In fact, it has so many feature levels that this app would be right at home as the video editor of choice in any professional film editing studio.*[63]

In another example, a Brazilian journalist reviewed LiVES and stated “LiVES is a free (gratis) video editor, which possesses many special features and functions, placing it on a professional level.)”[64] [65].

In 2009, Linux Journal (founded in 1994, and describing itself as “the original magazine of the global Linux community”) released an article entitled “It LiVES! Video

[63]LINUX INSIDER online journal. Available at: <http://www.linuxinsider.com/story/74201.html>. Accessed: 13 Jun. 2013.
[64]*LiVES é um editor de vídeos gratuito, que possui muitas funcionalidades e recursos especiais, colocando-o em um patamar profissional.*
[65]BAIXAKI website (Portuguese language). Available at: <http://www.baixaki.com.br/linux/download/lives.htm>. Accessed: 13 Jun. 2013.

Editing For FOSS Movie Makers”[66]. The author of the article describes the features in the then-current version of LiVES and concludes: *“Simply stated, LiVES is terrific.”*

In 2008, the pod-cast website hackerpublicradio gave LiVES a glowing review[67].

A lesser known website has a review which finishes with the conclusion

> *LiVES allows you to create very high quality video in a way that's easy to manage and effective. It offers many advantages to the modern video jockey and editor alike.*[68]

LiVES is frequently listed amongst the top Free/Open Source video applications, for example: (site 1[69]), (site 2[70]).

In 2007, a widely read independent magazine for the Ubuntu distribution, Full Circle, listed LiVES amongst the top five audiovisual applications for Ubuntu[71].

7.5.5 **Awards**

LiVES has won a few awards, including:

- in 2012, LiVES was selected as Project of the Month by the Distrowatch site.
- in 2009, LiVES was voted one of the 10 finalists for the category Best Multimedia Application in the Sourceforge Community Choice Awards.
- in 2008-2009, the non-profit LinuxFund helped to raise more than $7,000 to help fund

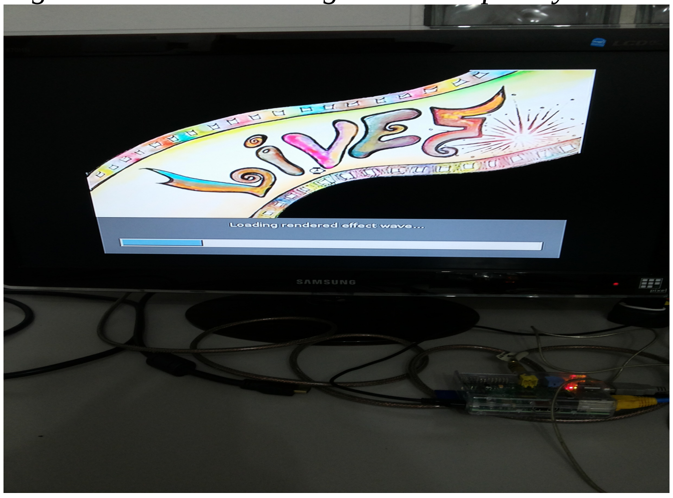
[66]LINUX JOURNAL article. Available at: <http://www.linuxjournal.com/content/it-lives-video-editing-foss-movie-makers>. Accessed: 13 Jun. 2013.
[67]HACKERPUBLICRADIO episode 114, audio recording. Available at: <http://www.hackerpublicradio.org/eps.php?id=0114>. Accessed: 13 Jun. 2013.
[68]<http://www.steves-digicams.com/knowledge-center/how-tos/video-software/video-editing-tools-an-introduction-to-lives.html>. Accessed: 13 Jun. 2013.
[69]EXAMPLE website. Available at: <http://wolfcrow.com/blog/a-quick-guide-to-the-5-best-open-source-video-editing-software/>. Accessed: 13 Jun. 2013.
[70]EXAMPLE website. Available at: <http://www.smallbusinesscomputing.com/ProductReviews/Software/5-best-open-source-video-editors-for-small-business-2.html>. Accessed: 13 Jun. 2013.
[71]FULL CIRCLE MAGAZINE issue 7. 2007. Available at: <http://fullcirclemagazine.org/issue-7/>. Accessed: 13 Jun. 2013.

LiVES development.

7.5.6 **Uses of LiVES**

LiVES can run under Linux, Mac OSX, BSD and Irix (a Windows version is currently in development). It has been tested and runs well on a range of hardware from large, multi-core servers down the humble Raspberry Pi (Figure 39).

Figure 36: LiVES running on the Raspberry Pi



Source: courtesy VJ Pixel

LiVES is included in at least two live distributions: Dynebolic[72] and AVLinux[73]. This gives users the opportunity to use the application even without having to install a copy of Linux – it can be run from a bootable DVD or USB stick.

Saul Goode has created a set of rendered effects plugins[74].

The application has been used in numerous shows and performances, including:

- The Waag, Amsterdam, Netherlands (2004)
- Riereta, Barcelona, Spain (2004)

[72]DYNEBOLIC website. Available at: <http://www.dynebolic.org/>. Accessed: 13 Jun. 2013.
[73]AVLINUX website. Available at: <http://www.bandshed.net/AVLinux.html>. Accessed: 13 Jun. 2013.
[74]GOODE, S. Third party plugins for LiVES. Available at: <http://chiselapp.com/user/saulgoode/repository/RFX-GIMP/home>. Accessed: 13 Jun. 2013.

- RunMe City Camp, in Aarhus, Denmark. LiVES was used for the closing VJ set, performed in conjunction with Alexei Shulgin[75] and Victor Soroka.
- HAIP 06, Ljubljana, Slovenia. Performance with Kentaro Fukuchi. (2006)[76]
- Festival de Invierno, Garanhuns, Brazil. Performance with Futuriveis/mediasana. (2007)
- Tipoia Festival, Brazil. Performance with Orquestra Contemporaneia de Olinda. (2007)[77]
- Campus Party Brazil. Collaborative performance with VJ Pixel, 2009[78] [79] [80]
- An installation of randomised clips and effects using LiVES ran interrupted for 30 days. Tilburg, Netherlands, 2005.

Clips created with LiVES have also had some success:

- Follow Love (salsaman) was shown at the .screen exhibition in the Teatergarasjen, Bergen, Norway (2004)
- A clip edited with LiVES was shown on Rotterdam TV (10/07/2004)
- Two clips produced by Marco de la Cruz using LiVES: "Time After Time" and "Anthem Part 2" were shown at Animaritime 2004 (New Brunswick, Canada)
- A poetic clip entitled “Before the Uncreation”[81] (salsaman and Karima Hoisan) has been viewed over 10,000 times.

7.5.7 **Reactions from users**

[75]WIKIPEDIA, Alexei Shulgin biography. Available at: <http://en.wikipedia.org/wiki/Alexei_Shulgin>. Accessed: 13 Jun. 2013.
[76]VIDEO clip of “Effect LiVES”. Available at: <http://video.kiberpipa.org/media/HAIP06_Effect_Lives/>. Accessed: 13 Jun. 2013.
[77]<http://www.youtube.com/watch?v=7_sJ8ZUeK7U>. Accessed: 13 Jun. 2013..
[78]<http://www.youtube.com/watch?v=bvF8q6gNdqM>. Accessed: 13 Jun. 2013.
[79]<http://www.youtube.com/watch?v=RseN7XbvRuw>. Accessed: 13 Jun. 2013.
[80]<http://www.youtube.com/watch?v=FlbJ6iq_QBY>. Accessed: 13 Jun. 2013.
[81]<http://www.youtube.com/watch?v=kpG6A-2fP4s>. Accessed: 13 Jun. 2013.

One user writes:

> ***The integrated video editing & VJ playback was the main thing[s] that brought me to LiVES**...**[I] Didn't really have the time for** the usual dance with **a separate editing app, making clips as new files** (usually involves clumsyfying [sic] **re-encoding** too), and **then making the setup in a performance app**.*
>
> *(<http://studio.kyperjokki.fi/engine/LivesExperience>. Accessed: 13 Jun. 2013.)*

Finally, on the Sourceforge project page for LiVES[82], some users have left short comments:

"What necessary words... super, a magnificent idea", *"It works. It supports pretty much everything you can think of. It doesn't crash in the hands of a casual use[r]"*, *"Perfect open source project! 10X!"*

[82]USER comments for LiVES, sourceforge. Available at: <http://sourceforge.net/projects/lives/reviews/?sort=usefulness&filter=all#reviews-n-ratings>. Accessed: 13 Jun. 2013.

8 **CONCLUSIONS**

In summary, we believe that this dissertation provides a valuable contribution, firstly by examining the needs of a wide range of users of video processing tools, and then by examining a possible solution which would satisfy those requirements.

Feedback received demonstrates LiVES' unique and wide ranging contribution to the field of video processing. It satisfies the technical requirements of a large range of users, in particular those of what we term “Experimental VJs”. It provides a high level of user friendliness with an extensive feature set which caters to various classes of user. Some of these features are unique to the application. In addition, LiVES generally received a high rating for performance and stability. At the same time, it is developed as a free/open source application, which enables rapid customisation and experimentation with the code. LiVES has reached a fairly wide audience in demographic terms, and has received a great deal of praise in the media.

8.1 FUTURE WORK

There is surely a lot of potential for LiVES to increase further in terms of user base and in terms of additional features. Given the popcon figure of less than 1% of Debian users having the application installed, there is presumably a lot of room for growth. In particular, it would benefit LiVES development greatly to have a larger number of developers involved. The core LiVES code can be utilised in many ways – as a VJ tool, as a multitrack editing tool, as a video server, as a video programming environment, as an online video editing service, and as a digital signal processor. With a larger development team, each of these facets could be pursued in far greater depth, whilst still leveraging the common core code. In addition, the heavy use of plugins in LiVES can enable some developers to work on plugins without the necessity to involve themselves in the core code.

Regarding the requirements side of LiVES, it would be useful to have a more complete understanding of the needs of the target user base and to compile a more extensive list of use cases (currently in use and desired use cases). A good way to achieve this would be face-to-face meetings with existing users. It has long been the desire of the author to organise a LiVES user group conference, however a lack of resources for the project has prevented this from occurring.

Nevertheless, LiVES continues the tradition begun by such equipment as the Video Toaster, FM Towns, and the software practices of the demoscene coders.

Kathleen Forde (2005), interviewed for “The VJ Book”, perhaps sums this up the best:

> *If you look at the development of the software that live visual artists use from a time-line perspective, you can see a real influence of what came before and what after, a clear evolution. Research and design and programming and software are their own art form… We're getting to a point where the research and the process of creating tools that enable more artwork to be made is, to me, almost an extension of Performance as Object-- sometimes that's as much of an artistic moment as many of the objects, performances, and scores, that come out of it.*
> *(FORDE, 2005, p. 39–40).*

REFERENCES

BOEHM, B.; TURNER, R. **Balancing agility and discipline**: a guide for the perplexed. Boston MA: Addison-Wesley, 2003. p. 55–57.

CHATLEY, R.; EISENBACH, E.; MAGEE, J. **Painless plugins**. London: Imperial College, 2003. Available at: <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.137.3509&rep=rep1&type=pdf>. Accessed: 13 Jun. 2013.

ĐURKOVIĆ, J.; VUKOVIĆ, V.; RAKOVIĆ, L. Open source approach in software development: advantages and disadvantages. In: **Management Information Systems**, Armonk, NY, v. 3, n. 2, p. 29-33, 2008.

FORDE, K. Interview. In: SPINRAD, P. **The VJ book**: inspirations and practical advice for live visuals performance. Los Angeles, CA: Feral House, 2005. p. 39-40.

FUKUCHI K.; MERTENS S.; TANNENBAUM E. EffecTV: a real-time software video effect processor for entertainment. In: **Lecture Notes in Computer Science**. Berlin: Springer-Verlag, v. 3166, p. 602-605, 2004.

IBM. **Use cases, best practices**, 2003. Available at: <http://www.ibm.com/developerworks/rational/library/content/03July/2500/2784/2784_use_cases.pdf>. Accessed: 13 Jun 2013.

JÁCOME, J. **Sistemas interativos de tempo real para processamento audiovisual integrado**. 2007. 127 p. Dissertation (Masters in Computer Science) – Centro de Informática, Universidade Federal de Pernambuco, Recife. Available at: <http://jarbasjacome.files.wordpress.com/2009/04/dissertacao_vimus_jarbasjacome2007.pdf>. Accessed: 13 Jun 2013.

MANOVICH, L. Cinematic and graphic: cinegratography. In: **______ The Language of new media**. Cambridge, MA: MIT Press, 2001. p. 309-330.

MAYER J. Graphical user interfaces composed of plug-ins. In: EUROPEAN GCSE YOUNG RESEARCHERS WORKSHOP, 4TH, 2002, Erfurt, Germany. **Proceedings**. Kaiserslautern, Germany: Fraunhofer IESE, 2002. p. 28-32.

MAYER J.; MELZER I.; SCHWEIGGERT F. Lightweight plug-in-based application development. In: **Lecture Notes in Computer Science**. Berlin: Springer-Verlag, v. 2591, p 87-102, 2002.

MAYER-PATEL K. D. **A Parallel software-only video effect processing system**. Berkeley: University of California, 1999. Department of Computer Science. Available at: <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.3.3166>. Accessed: 1 Jul. 2013.

MILES, A. Programmatic statements for a facetted videography. In: LOVINK, G.; NIEDERER, S. **Video vortex reader**: responses to YouTube. Amsterdam: Institute of Network Cultures, 2008. p. 223-230. Available at: <http://networkcultures.org/wpmu/portal/publications/inc-readers/videovortex/>. Accessed:

25 Sep. 2013.

NUSEBEIH B. Weaving together requirements and architecture. In: **IEEE Computer**, Los Alamitos, CA, v. 34, n. 3, p. 115–117, 2001. Available at: <http://mcs.open.ac.uk/ban25/papers/computer2001.pdf>. Accessed: 13 Jun. 2013.

RAYMOND, E. **The Cathedral and the bazaar**. Sebastopol, CA: O'Reilly Media, 1999.

SHERMAN, T. Vernacular video. In: LOVINK, G.; NIEDERER, S. **Video vortex reader**: responses to YouTube. Amsterdam: Institute of Network Cultures, 2008. p. 161-168.

SPINRAD, P. Introduction. In: ______ **The VJ book**: inspirations and practical advice for live visuals performance. Los Angeles, CA: Feral House, 2005. p 13-15.

APPENDIX A - Definitions

- **MIDI**[83]

MIDI is Musical Instrument Digital Interface, a technical standard that describes a protocol, digital interface and connectors and allows a wide variety of electronic instruments, computers and other related devices to connect and communicate with one another. MIDI technology was standardized in 1983.

- **OSC**[84]

OSC is Open Sound Control – a network control protocol originally used to control audio applications. More recently it is becoming ever more widely used to control video applications.

- **Firewire**[85]

Firewire, also known as the **IEEE 1394 interface** is a serial bus interface standard for high-speed communications and isonchronous real-time data transfer. It is used by many cameras to transfer video and audio material to a computer or other device.

- **Jack**[86]

The Jack Audio Toolkit - JACK is system for handling real-time, low latency audio. It has an audio part for connecting audio between devices and applications, and a transport part, for synchronised playback.

- **Frei0r**[87]

Frei0r is a simple, open source, cross platform framework for video effects. It provides filters, mixers and generators by means of minimalistic plugin API. The behaviour of the effects can be controlled from the host through simple parameters. The intent is to solve the recurring reimplementation or adaptation issue of standard video effects.

- **LADSPA**[88]

[83]WIKIPEDIA, MIDI. Available at: <http://en.wikipedia.org/wiki/MIDI>. Accessed: 13 Jun. 2013.
[84]ABOUT OSC. Available at: <http://opensoundcontrol.org/introduction-osc>. Accessed: 13 Jun. 2013.
[85]WIKIPEDIA, FireWire. Available at: <http://en.wikipedia.org/wiki/Firewire>. Accessed: 13 Jun. 2013.
[86]JACK Audio Toolkit website. Available at: <http://jackaudio.org/>. Accessed: 13 Jun. 2013.
[87]WIKIPEDIA Frei0r. Available at: <http://en.wikipedia.org/wiki/Frei0r>. Accessed: 13 Jun. 2013.
[88]LADSPA website. Available at: <http://www.ladspa.org/>. Accessed: 13 Jun. 2013.

LADSPA is a standard and framework that allows software audio processors and effects to be plugged into a wide range of audio synthesis and recording packages. There is a large number of open source plugins which have been written using this framework.

- **v4l**[89]

V4l (video for Linux) is a module for the Linux kernel. By connecting with this module, it possible for any application which produces video output to appear to the system as if it were a hardware web camera. This in turn implies that the application can be used an input source for other programs which work with web-cams, without requiring any changes in the second program.

The terms "*Free Software*" and "*Open Source Software*" are sometimes a source of confusion. The terms are sometimes used interchangeably, but in fact they describe different processes.

- **Open Source Software**

Open Source Software is software where *the source code is freely available* (i.e., without any requirement for payment or any other obstacles) for anybody to inspect and study.

- **Free Software**

Free Software is a type of Open Source Software, but it also guarantees more freedom for the user. Free Software is software for which everyone has the right not only to inspect and study the source code but also *to use it for any desired purpose, including the right to redistribute it, without monetary or other restrictions*.

Note that "Free" here is used in the sense of "Freedom of Speech" (libre), not necessarily in the sense of "Free from payment" (gratis).[90]

In addition, **copyleft licenses** require that these same freedoms are passed on to subsequent users. Therefore, each time a binary is passed on (distributed) it must be accompanied by the source code, and must maintain the original license.

[89]ABOUT Video4Linux. Available at: <http://en.wikipedia.org/wiki/Video4Linux>. Accessed: 13 Jun. 2013.

[90] This may at first appear confusing given that we mentioned the phrase "*without monetary or other restrictions*". However, this freedom only applies to users who have legally obtained a copy of the source code. Therefore, the original developer (copyright holder) can charge others a fee to obtain an original copy of the code.

APPENDIX B - The Questionnaire

LiVES Survey

Please take a moment to fill in the details below.

*Required

1. **First name (optional)**

2. **Last name (optional)**

3. **Email address (optional)**

How would you rate the following in LiVES:

4. **Feature completeness** *
Mark only one oval.

	1	2	3	4	5	
Poor	⬭	⬭	⬭	⬭	⬭	Excellent

5. **User friendliness** *
Mark only one oval.

	1	2	3	4	5	
Poor	⬭	⬭	⬭	⬭	⬭	Excellent

6. **Performance** *
Mark only one oval.

	1	2	3	4	5	
Poor	⬭	⬭	⬭	⬭	⬭	Excellent

7. **Stability** *

Mark only one oval.

	1	2	3	4	5	
Poor	◯	◯	◯	◯	◯	Excellent

8. **Documentation / tutorials** *

Mark only one oval.

	1	2	3	4	5	
Poor	◯	◯	◯	◯	◯	Excellent

9. **How did you discover LiVES ?**

10. **What other video editing applications do you use regularly (if any) ?**

11. **Are there any features you would like to see added to LiVES ?**

12. **Which existing features in LiVES are most important to you ?**
More than one item may be selected.
Tick all that apply.

- ☐ Price
- ☐ Level of support
- ☐ Frequency of releases / bugfixes
- ☐ Linux support
- ☐ Accessibility of code
- ☐ Ability to edit individual frames
- ☐ Webcam/TV card input
- ☐ Firewire input
- ☐ Audio plugins
- ☐ Realtime effects
- ☐ Data connections
- ☐ OSC control
- ☐ MIDI/joystick interface
- ☐ Ease of use
- ☐ Stability
- ☐ Ability to record realtime performances
- ☐ Frei0r compatibility
- ☐ v4l2 compatibility
- ☐ Wide range of input formats
- ☐ Range/quality of output formats
- ☐ Control of playback at different rates
- ☐ Clip editing features
- ☐ Crash recovery
- ☐ Multi monitor support
- ☐ Jack audio integration
- ☐ Subtitle support
- ☐ Local language support
- ☐ Themes / appearance
- ☐ Multitrack editing
- ☐ Other: ....

APPENDIX C - Questionnaire Results

Feature completeness	User friendliness	Performance	Stability	Documentation / tutorials	Which existing features in LiVES are most important to you ?	How did you discover LiVES ?	What other video editing applications do you use regularly (if any) ?	Are there any features you would like to see added to LiVES ?
5	5	5	5	5	Price, Level of support, Frequency of releases / bugfixes, Linux support, Ability to edit individual frames, Ease of use, Stability, Range/quality of output formats, Control of playback at different rates, Clip editing features, Crash recovery	Suggested to me by someone on the alt.os.linux.mandriva newsgroup when I was looking for an application that I can view individual frames with.		
5	5	5	4	5	Price, Realtime effects, MIDI/joystick interface, Ease of use	Campus Party, maior evento de tecnologia do mundo	FFmpeg, kdenlive, @pen Shot, cinelerra, cs6	Mapeamento de imagens

Feature completeness	User friendliness	Performance	Stability	Documentation / tutorials	Which existing features in LiVES are most important to you ?	How did you discover LiVES ?	What other video editing applications do you use regularly (if any) ?	Are there any features you would like to see added to LiVES ?
5	5	5	4	5	Linux support, Firewire input, Realtime effects, Ease of use, Stability, Ability to record realtime performances, Wide range of input formats, Range/quality of output formats, Control of playback at different rates, Crash recovery, Multi monitor support, Jack audio integration, Subtitle support, Multitrack Editing	I searched sourceforge for videoeditors	Not regularly, but occasionally I use OpenShot, iMovie, Premiere and Pinnacle	We all love new toys, don't we? But do we really NEED them?
4	4	3	2	4	Linux support, Ability to edit individual frames, Realtime effects, Stability, Ability to record realtime performances, Frei0r compatibility, Clip editing features, Crash recovery, Multi monitor support, Jack audio integration, Subtitle support	I discovered LiVES when I was search for a video editor to create AMV's. A simple Google search revealed its existence to me and I've been using it ever since.	Sony Vegas and OpenShot	The main one I'd like to see is an unlimited undo that works in the clip editor. I know this feature is there in multitrack but having it available in the clip editor would be a HUGE improvement in my opinion and help the software's adoption.
5	4	4	4	4	Price, Linux support, Accessibility of code, Realtime effects, Ease of use, Stability, Multi monitor support			

Feature completeness	User friendliness	Performance	Stability	Documentation / tutorials	Which existing features in LiVES are most important to you ?	How did you discover LiVES ?	What other video editing applications do you use regularly (if any) ?	Are there any features you would like to see added to LiVES ?
3	1	3	3	2	Linux support, Ability to edit individual frames, Realtime effects, OSC control, Ease of use, Stability, Frei0r compatibility, Control of playback at different rates, Jack audio integration, Themes / appearance		kdenlive	
3	4	4	4	4	Level of support, Linux support, Realtime effects, OSC control, Ease of use, Frei0r compatibility, Wide range of input formats	I need a live video system, and i search what i can use with linux. And so after several trys, i liked lives	Kdenlive	
4	3	4	4	2	Linux support, Stability, Frei0r compatibility, Jack audio integration	By searching "video edit linux"		
3	3	2	2	3	Linux support, Ability to edit individual frames, Realtime effects, Ease of use, Stability, Range/quality of output formats, Clip editing features, Crash recovery, Multitrack editing	I have testing many software in Debian distrib	none	I have problems with the latest versions. The melted deteriorated after some updates and I'm looking for overlay effects that I can not do.

Feature completeness	User friendliness	Performance	Stability	Documentation / tutorials	Which existing features in LiVES are most important to you ?	How did you discover LiVES ?	What other video editing applications do you use regularly (if any) ?	Are there any features you would like to see added to LiVES ?
4	4	4	4	4	Level of support, Linux support, Webcam/TV card input, Audio plugins, Realtime effects, MIDI/joystick interface, Ease of use, Ability to record realtime performances, Frei0r compatibility, v4l2 compatibility, Control of playback at different rates, Clip editing features, Multitrack editing	discovered lives in the dynebolic distro.		audio reactive realtime effects.
3	2	5	4	4	Level of support, Linux support, Realtime effects, MIDI/joystick interface, Ease of use, Stability, Wide range of input formats, Crash recovery	In 2004 I did VJ in the CC launch in Brazil, as part of FISL. At that time, I decided to invite some local VJs, but the software I was using doesn't had a easy way to run on Windows. When I looked for a solution, I've found that dyne:bolic come with LiVES and used it for the presentation (both in my computer and the computers of the other VJs).	Kdelive	Possibility to have preview of videos before sending them to the main screen.

Feature completeness	User friendliness	Performance	Stability	Documentation / tutorials	Which existing features in LiVES are most important to you ?	How did you discover LiVES ?	What other video editing applications do you use regularly (if any) ?	Are there any features you would like to see added to LiVES ?
4	3	4	3	3	Level of support, Linux support, Accessibility of code, Ability to edit individual frames, Webcam/TV card input, Realtime effects, OSC control, v4l2 compatibility, Wide range of input formats, Range/quality of output formats	Internet search	OpenShot	- Multiple streams / engines running trough one interface, each engine with it's own clip set, video out device and OSCP port with a view window that show each stream.
4	5	4	4	4	Linux support, Ability to edit individual frames, Audio plugins, Stability, Jack audio integration	I was searching for a Linux based video editor to "play with."	KDELive	None come to mind, but I am a relative new comer to video editing.
4	4	4	4	3	Level of support, Frequency of releases / bugfixes, Linux support, Accessibility of code, Webcam/TV card input, Firewire input, Realtime effects, OSC control, MIDI/joystick interface, Stability, Frei0r compatibility, v4l2 compatibility, Jack audio integration, Themes / appearance	whit dynebolic	kdenlive	yes mlt support
3	3	3	3	3				