
ofxPiMapper

Projection Mapping Tool for the Raspberry Pi

By

KRISJANIS RIJNIEKS



Department of Media, Master's Degree programme in New Media
AALTO UNIVERSITY, SCHOOL OF ARTS, DESIGN AND ARCHITECTURE

Thesis supervisors: Markku Reunanen, Rasmus Vuori

OCTOBER 2015

ABSTRACT

OfxPiMapper is an open source tool for projection mapping i.e., video mapping or spatial augmented reality, created as an add-on for openFrameworks; This allows the possibility of integration with almost any kind of graphic—static, video, generative, interactive or any custom project of a similar kind. It is designed to run on the ARM powered Raspberry Pi pocket-size computer. There currently exist both commercial, closed source and free, open source tools available for projection mapping; This thesis documents the attempt to create a new open source tool which inherits some features from those existing tools and which considers the development of new specific features. Furthermore, existing projection mapping software and details regarding the design and development of ofxPiMapper, as well as the process of releasing it open source, are discussed in this master's thesis.

Keywords: projection mapping; video mapping; spatial augmented reality; software development; ARM; tool; openframeworks; open source; raspberry pi

ACKNOWLEDGEMENTS

First of all I would like to thank my thesis supervisor Markku Reunanan for putting so much effort into reading and re-reading my thesis even while he was supposed to focus on his. Thank you Markku for all your suggestions—they (sorry for the em dash) led me the right way and gave much inspiration that was needed for writing my thesis and will be valid afterwards when working on other projects.

Thank you Helsinki Media Lab for being so flexible and allowing me to study the way I feel right, thesis supervisor Rasmus Vuori for accepting all my personal study projects and for tips regarding my thesis structure. Anna Arsniva for being an example of professionalism, always answering my emails before I have managed to switch to the next task in my to-do list. Lily Diaz-Kommonen for widening my view of design in general. I would also like to thank the Media Lab library staff Ilpo Kari and Pekka Sallonen as well as Janne Lehtimäki for always being able to help with hardware, tips and being open for an interesting discussion.

Fellow students Matti Niinimäki, Antti Hietaniemi, Sébastien Piquemal, Suse Miessner, Sasha Kazantsev, Henrik Hackenberg, Jonna Maarit, Nelly Sääksjärvi, Ranjit Menon, Akihiko Piko Sugiura and others—it was a pleasure to get to know you and I am honored to have you as friends.

Thanks Aalto Media Factory for having the best coffee in Arabia campus (Jon Fabritius), Aalto Fab Lab (Anu Määttä, Juhani Tenhunen) for keeping the lab running and hosting my workshops. Wolf Jeschonnek, Murat Vurucu and Nicolai Hertle for inviting me to be a part of the Fab Lab Berlin and thus allowing me to continue with the idea of building a fab lab as it was the topic of my initial master's thesis proposition.

Erich Stüssi and Philip Silva, colleagues at the Fab Lab Berlin, for additional pairs of eyes for reading chapters of my thesis, identifying inconsistencies and showing ways to fix them. The rest of Fab Lab Berlin for showing interest and enthusiasm towards my thesis, joining Geek Parkour trainings and generally being nice.

And finally, my partner Irina Spicaka for understanding, love and putting so much effort into creating a future vision of what the practical outcome of this thesis, the ofxPiMapper projection mapping tool for the Raspberry Pi, could be. I love you and we are going to make this big :]

TABLE OF CONTENTS

	Page
List of Tables	ix
List of Figures	xi
1 Introduction	1
1.1 Motivation	3
1.1.1 Personal Experience	4
1.1.2 Projection Mapping Installations	7
1.2 Goals	7
1.2.1 Create New Software for Projection Mapping	8
1.2.2 Optimize the Software for Low-End Hardware	8
1.2.3 Release the Software as Open Source	9
1.3 Structure of the Thesis	9
2 Background Information	11
2.1 Spatial Augmented Reality	11
2.2 Raspberry Pi	12
2.3 OpenFrameworks	13
2.4 C++	15
2.5 Linux	15
2.6 Git and GitHub	16
3 Existing Software	17
3.1 User Research	17
3.2 Internet Research	19
4 Existing Software Analysis	23
4.1 Specialized Projection Mapping Software	24
4.1.1 MadMapper	24
4.1.2 Visution Mapio	25
4.2 Software with Additional Projection Mapping Functionality	27

TABLE OF CONTENTS

4.2.1	Resolume Arena	28
4.2.2	Millumin	29
4.2.3	VDMX	30
4.3	Creative Coding Frameworks	31
4.3.1	vvv	31
4.3.2	Processing	32
4.4	OpenFrameworks Add-ons	33
4.5	Summary	34
5	Software Development	37
5.1	Software Requirements	38
5.2	User Interface	40
5.2.1	Graphical User Interface	41
5.2.2	Keyboard Interface	44
5.2.3	OpenFrameworks Add-on API	45
5.2.4	Open Sound Control API	45
5.3	Software Architecture	46
5.3.1	Overview	46
5.3.2	Subsystems	47
5.3.3	Putting It All Together	52
6	Hardware-Specific Testing	55
6.1	OpenFrameworks on the Raspberry Pi Test	55
6.2	Texture Mapping Test	56
6.3	Remote Control Test	56
6.4	Generative Texture Test	57
6.5	Video Texture Test	57
6.6	Summary	59
7	Releasing Open Source	61
7.1	Releasing on GitHub	61
7.2	Documentation	62
7.2.1	Usage Instructions	63
7.2.2	Development Guidelines	63
7.2.3	Programming Conventions	64
7.2.4	Collaboration	64
8	Conclusions	67
8.1	Evaluation of the Initially Set Goals	67
8.2	Future Research and Development	69

Bibliography

71

LIST OF TABLES

TABLE	Page
2.1 Raspberry Pi pricing (reichelt.de 6 May 2015)	13
3.1 Projection mapping software user questionnaire	18
4.1 OpenFrameworks projection mapping add-ons	34
5.1 Current Key Mapping of ofxPiMapper	45

LIST OF FIGURES

FIGURE	Page
1.1 The Ambassadors (1533) by Hans Holbein the Younger. Source: Google Cultural Institute	2
1.2 Displacements by Michael Naimark (1980). Screenshot. Source: naimark.net	2
1.3 Typical projector—screen setup for PNG-mask-based projection mapping	4
1.4 Hardware setup for the installation You and Now	5
1.5 Rijnieks K., Spicaka I., Buravicky P., Installation Metasphere	6
2.1 Projection mapping components	12
3.1 Projection mapping software used by the community	19
4.1 MadMapper user interface	25
4.2 Visution Mapio user interface	26
4.3 Visution Mapio rotation tool behaviour	27
4.4 Projection mapping settings in Resolume Arena	28
4.5 Millumin projection mapping user interface	29
4.6 VDMX projection mapping user interface	31
4.7 User interface of badMapper	32
4.8 SurfaceMapperGUI interface	33
5.1 Initial sketch of ofxPiMapper interface states	42
5.2 OfxPiMapper openFrameworks add-on initialization	47
5.3 OfxPiMapper system overview	48
5.4 Undo/redo stack	49
5.5 Flux architecture event flow	52
5.6 ofxPiMapper event/data flow	53
5.7 ofxPiMapper stores and view analogy with the Flux architecture	53
5.8 MVC Architecture	54
6.1 Generative texture test algorithm	58

LIST OF FIGURES

- 8.1 Projection Mapping with the Raspberry Pi workshop at the CLICK Festival 2013 . . . 68

INTRODUCTION

Projection mapping, also known as video mapping and spatial augmented reality [31], sometimes referred to as simply mapping [36, p. 15], is an approach to turn almost any surface into a screen. Lately it has been used in live audiovisual shows, art installations, dance, theatre as well as different research projects.

As stated in the Illustrated History of Projection Mapping by Jones B. [31], this approach is not new, but the term “projection mapping” is. Projection mapping is about geometric transformations of an image: homothety, homography and anamorphism [36, p. 16]. Anamorphism has been used by many artists in the past, for example Hans Holbein the Younger used it in his famous painting “The Ambassadors” (Figure 1.1) where a skull is painted in the foreground of the image in a way that it is only recognisable by viewing the painting from an angle.

Creating a three-dimensional illusion by projecting light on a flat surface can be also considered projection mapping; Light installation Afrum [78] is one example of this form. In Afrum, projected light as a medium from a halogen projector is passed through pierced metal plates towards the corner of a white room creating the illusion of a three-dimensional object.

Art installation Displacements [41] is another example that proves the use of projection mapping techniques from the past. A continuous scene is captured by a rotating camera in a room. The recorded objects are then painted white (covering the texture). The texture is then re-applied to the objects by using a rotating video projector that projects the previously captured footage (Figure 1.2).

The way we think of projection mapping today is not that old, but it is older than the term “projection mapping” [31]. Digital projector and computer technology has simplified the way we approach projection mapping. One does not have to pierce sheets of metal in order to shape a beam of light. Software tools can be used to produce previously unseen projection mapping



FIGURE 1.1. The Ambassadors (1533) by Hans Holbein the Younger. Source: Google Cultural Institute



FIGURE 1.2. Displacements by Michael Naimark (1980). Screenshot. Source: naimark.net

effects.

As projection mapping has become more popular, software tools for more demanding users have emerged. Demanding users can be defined as those who have used the first available

projection mapping software tools, such as the MadMapper [18] with basic functionality. Those users now expect extra functionality and extendability on top of the features they already are familiar with. These tools (Section 4.1) have solved many user interface and usability-related problems through the development of projection mapping, but better tools still need to be created.

My project, `ofxPiMapper` is an attempt to solve some of the problems (discussed in Chapters 3 and 4) related to projection mapping software and also hardware used for this purpose. The `ofxPiMapper` tool can be used in many contexts—for example, research projects such as the “Toolkit support for interactive projected displays” by Hardy et al. [29] and “Combining multiple depth cameras and projectors for interactions on, above and between surfaces” by Wilson et. al. [87] could benefit from such a tool. The same goes for audiovisual performances, art installations and also fields like archeology and exploration of cultural heritage artifacts [55]. `OfxPiMapper` attempts to reduce time investment in complex projects where projection mapping is a part of an integrated solution. This includes interactive spaces such as “The Office of the Future” by Raskar et. al. [50] and similar projects, where a lot of effort must be invested in analyzing sensor input and generating visual feedback and complex interactive or generative art projects with hardware or commercial software budget constraints.

More people are continually becoming interested in projection mapping for use in their creative projects, however, the needed skill-sets and hardware/software costs can often be a limitation for them to get started. In the context of this master’s thesis a research question can be set: how can projection mapping be made more available through the production of an open-source projection mapping tool for low-end (cheaper, available) hardware?

1.1 Motivation

The motivation to develop a new projection mapping tool mainly comes from my personal experience with various projection mapping projects. Since 2001 I have been working with animation and Vjing [66]; Around 2008 projection mapping became a topical term. It was starting to become boring to work with with visuals projected on a rectangular 4:3 or even 16:9 aspect ratio surface. Projectors have become cheaper and more available, thus the perception of them as a tool for visual arts (as a light brush to paint surroundings rather than a tool for viewing presentation slides) became more apparent among VJs and visual artists in general.

At the time, there were no easy to use tools, only “hacks” and similar solutions available to people familiar with programming. One of the methods at the time was using masks—creating a Portable Network Graphics or PNG image [30] with opaque and transparent areas, then using that as an overlay for the source material. VJ software at the time allowed multiple different layers of video or image file sources; the top layer in the masking approach would be the PNG mask that allowed some parts of the video pass through the transparent areas of the image.

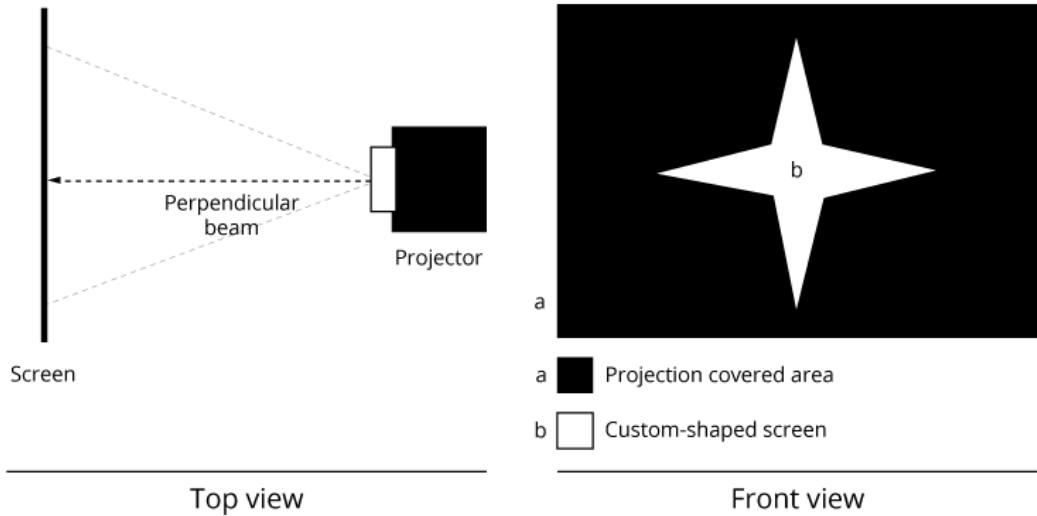


FIGURE 1.3. Typical projector—screen setup for PNG-mask-based projection mapping

This approach worked well for two-dimensional surfaces where the projector beam was aimed perpendicularly towards the surface (Figure 1.3).

Another popular approach was to use visual programming software tools like the Quartz Composer [15] and create custom setups that would allow individuals to manipulate an image in virtual 3D space or do triangle mesh, also known as simplicial complex [28, p. 7], and homography [17] based image transformations. VJs who had their own setup of this kind seemed to have reached an almost impossible level of professionalism before the first projection mapping software tools appeared on the market.

MadMapper was one of those first video mapping software tools. It revealed projection mapping as a way of visual expression for many artists and designers with no advanced computer experience. Many other software tools emerged at that time and the most popular ones are discussed in this thesis. It was possible to use both of the previously described approaches (PNG and image transformations) with MadMapper through user-friendly interface. I have used MadMapper in many projects since then, it provides flexibility as it integrates well with other niche software on the Mac OS X operating system along with being fast and easy to use.

1.1.1 Personal Experience

One of my most recent projects using MadMapper for projection mapping was a project named "You and Now". It was an interactive installation that was located in the Dome Square in Riga, Latvia during the annual light festival, Light Riga in November 2013. I was commissioned to

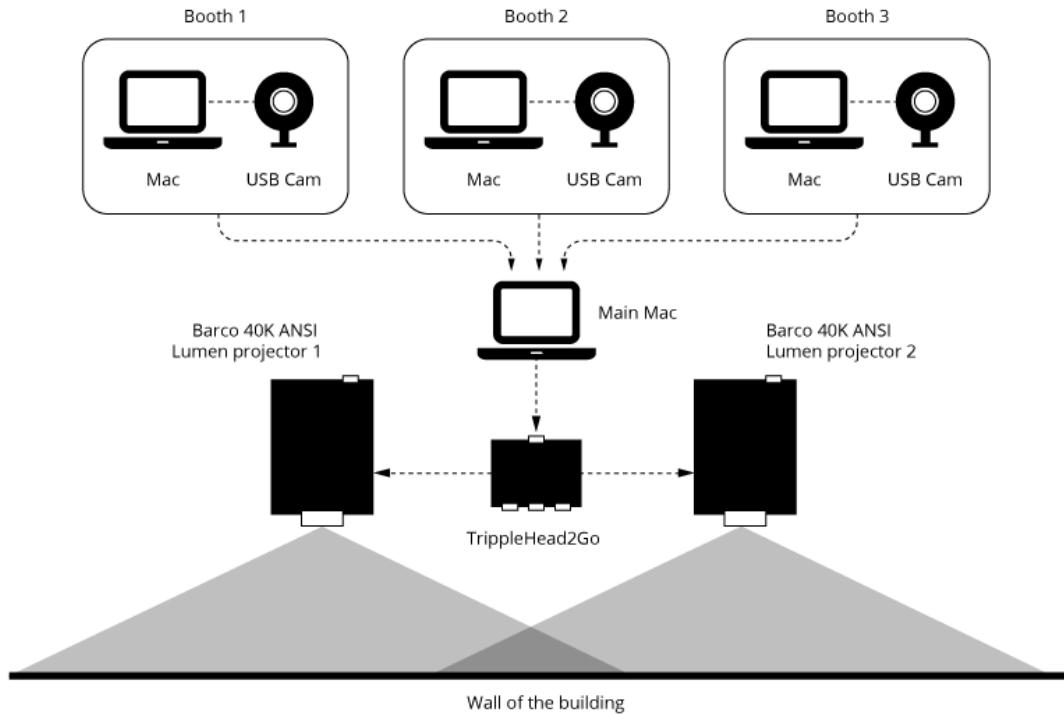


FIGURE 1.4. Hardware setup for the installation You and Now

design the hardware and software setup as well as to develop a custom software solution.

The solution consisted of three video booths, each equipped with a computer running the Mac OS X operating system and a Full HD USB camera. 10 second long videos were recorded using a custom software installed on the computers; they captured and post-processed the videos on-the-fly, sending the video to a main computer afterwards. The main computer was also running the Mac OS X operating system and was set up to receive the video files from the booth computers over the local wired network. Custom software running on the main computer mixed the videos last received in a semi-generative animation that was 2 minutes long and was played back in a loop. A hardware setup diagram of the installation can be seen in Figure 1.4.

Two high-brightness projectors were used to fully cover the facade of a building with the final image, while a projector blending software called Blendy VJ [65] was used to transform the output generated by the custom software. Finally MadMapper was used to map the image on the facade of the building. Syphon [6] made it possible to pipe the video from the custom software, through Blendy VJ, to MadMapper and finally cover the facade with an animated texture.

The production time of the project was 2 weeks. Without having off the shelf software like Blendy VJ and MadMapper it would not be possible to create the installation in such short



FIGURE 1.5. Installation Metasphere

time. Based on my experience, at least 2 more weeks would be needed to integrate projection blending and projection mapping features in the custom software part itself if the Mac OS X platform did not provide an option to share video output between running applications via Syphon.

Another of my projects using MadMapper for projection mapping is the installation and audio-visual performance Metasphere (Figure 1.5), done together with Irina Spicaka and Platon Buravicky [58].

The project itself is based on a hemispherical screen that is a polygonal structure built of triangles. Custom software was made to generate visuals that would be projection-mapped onto the hemispherical screen. The custom software was build around a 3D mesh dataset that modified its structure according to different types of inputs. Then, via Syphon, the generated image was forwarded to MadMapper where it was transformed to perfectly fit the screen.

Installations can last for long periods of time and MadMapper being able to run only on Mac OS X computers makes it an expensive process. With each new project of this kind, I get new ideas of how to simplify the process by using new tools and hardware, thus creating increasingly robust solutions. These kinds of projects are expensive mainly because of the hardware used. There is no way of saving resources on the projectors, but the computer hardware and software can still be optimized. Personal experience is not my only motivation, there are also other scenarios where a new projection mapping tool like ofxPiMapper will be useful.

1.1.2 Projection Mapping Installations

The workflow of projection mapping installations usually involve work on one's personal computer and likely also involves using commercial software such as MadMapper. For installations that run for long periods of time there is often no possibility of leaving one's personal computer on location. This scenario involves transferring the entire setup to a new computer or in the worst case a third party will be responsible for transferring the configuration. This can make the setup of the installation an unbearable process for all involved.

Furthermore, if the installation is to be stored, moved and set up again, the set-up process must be repeated. If the original author is not involved in the setup, problems and difficulties are extremely common—starting from the right choice of hardware and software to maintenance.

`OfxPiMapper` intends to solve some of the problems mentioned above. First, it attempts to eliminate the need for commercial projection mapping software. Second, as it runs on lower-end computers, such as the Raspberry Pi, separating the content of the art-piece from the hardware becomes unnecessary. The Raspberry Pi is inexpensive and small enough to not cause problems regarding the right choice of computer hardware in a remote location along with saving human work hours setting up software and debugging. The Raspberry Pi does not have parts like cooling fan and a hard drive which can fail and cause trouble later on.

Preparing a live audiovisual show can be a complicated process, especially if custom programming and a custom hardware setup is required. It gets much more complicated when frequent location changes are added, as in the case of artist tours when one has to perform in multiple places scattered around the world in a short period of time.

`OfxPiMapper` would make the setup of an audiovisual show more robust and prepared for frequent mobility. It would allow the creation of clone projects that would cost less and be physically smaller. This would make the transportation and maintenance costs of such projects smaller as well the projects themselves—less risky.

Research projects like the Office of the Future by Raskar, Ramesh et al. [50] are another motivation of mine for undertaking this project. `OfxPiMapper` attempts to provide a reusable projection mapping solution for the ideas discussed in the Office of the Future concept and similar projects. Open-source projection mapping tools like `ofxPiMapper` will serve as useful building blocks for applications in this context.

1.2 Goals

This thesis attempts to solve some of the problems existing in the projection mapping field by introducing a new software solution called the `ofxPiMapper`. By summarizing the real-world problems mentioned in the Motivation section, we can specify the following goals:

- Create new software for projection mapping

- Optimize the software for low-end hardware
- Release the software as open source

It should be possible to integrate `ofxPiMapper` with other software to be able to create robust generative solutions with centralized interface as using multiple software tools within a solution on a single computer can make the customization and control of the project a very frustrating process. In the following subsections each of the goals will be discussed in greater detail.

1.2.1 Create New Software for Projection Mapping

The main and most important goal of this thesis is to create a new tool for projection mapping that would include already accepted features from existing projection mapping software and introduce new ones that would make the projection mapping process with the new software easier in specific contexts.

The previously mentioned MadMapper projection mapping software is exceptionally good, the main downside being its ability to run solely on computers equipped with Mac OS X operating system. It would be useful to have all the good features of MadMapper, but not being limited to Mac OS X—being able to choose the hardware platform and operating system without compromising the projection mapping tool of choice. Therefore new features based on personal experience and the one of the projection mapping community could be added.

1.2.2 Optimize the Software for Low-End Hardware

The software will be designed in a way it is able to run on low-end hardware. Raspberry Pi and other ARM (limited instruction set processors mostly used in smartphones) platform [8] based computers are the main targets here. Projection mapping software compatible with the Raspberry Pi would make a big difference regarding the pricing of projection mapping related projects.

Raspberry Pi has been chosen as the main development platform, but the tool itself will support other popular hardware and operating system combinations. As `ofxPiMapper` is written in C++ programming language and will run on the Raspberry Pi pocket-size computer, other ARM based platforms will be compatible, Mac OS X, Windows and many flavours of Linux also being the case. Base platform of `ofxPiMapper` (`openFrameworks`) is a multiplatform one and thus allows to compile and run same source code on all the operating systems `openFrameworks` is configured for.

1.2.3 Release the Software as Open Source

To make the projection mapping tool even more available—it will be released as open source software thus making it possible for other people contribute features that they need instead of asking for them to a centralized entity. An important part of the process is the development of easy to grasp documentation—not only about the usage of the software but also about the recommended ways of developing it. In *The Cathedral and the Bazaar* by Eric S. Raymond [52] open-sourcing a project is also seen as a way to improve quality of the software as the source code is being opened for examination by numerous pairs of eyes of other programmers around the globe.

I decided to develop `ofxPiMapper` as open source software because I understand the limits of the time that I can allocate for programming. In case the software proves itself useful for someone, he or she might want to improve it in some way. By allowing the users of `ofxPiMapper` to access the source code I free them from the constraints of me not having time for developing new features they want as soon as they want. I also decided `ofxPiMapper` to be open source software rather than free software [27]. I wanted to keep the possibility for users to use the source code in their closed source and commercial projects with the only rule being that the copyright notice is being included. I decided to do so because I have experienced practical value of this through using `openFrameworks`, which is licensed under the MIT license [60, p. 85]—it legally allowed me to create my own closed source commercial projects by basing them on the `openFrameworks` source code.

Developing `ofxPiMapper` as an open source software tool will also make it easier to create custom software solutions where projection mapping is only a part of the whole. Instead of using many precompiling binaries and relying on specific operating system features, it will be possible to reuse the source code as a base or an add-on for something unique.

1.3 Structure of the Thesis

In this section I discuss the structure of my thesis. This section is a part of the Introduction or Chapter 1. It introduces one to the field of projection mapping and discusses my motivation for creating `ofxPiMapper`. Chapter 2 introduces to the terms and concepts used in the thesis if they are not clear already.

Next, existing software in the field of projection mapping is discussed. In Chapter 3 different sources are used to obtain a list of currently used software that is related to projection mapping. User Research (Section 3.1) summarizes a user questionnaire about software tools used by the community—which are the most popular ones, what are the features that users value most and what features are still missing. In Chapter 4 categories of projection mapping software are formed and specific examples are being discussed.

Chapter 5 discusses different aspects of the development of `ofxPiMapper`—its software

CHAPTER 1. INTRODUCTION

requirements, user interface and software architecture. In Chapter 6 I describe different hardware specific tests that have been done prior or during the development of `ofxPiMapper`. Chapter 7 is about releasing `ofxPiMapper` open source, tools and web platforms used for it, problems faced and how they were solved.

Conclusions about the development process of the `ofxPiMapper` in regard to the initially set goals are discussed in the Conclusions (Chapter 8). The future research and development guidelines are outlined there based on how well the initially set goals were met.

CHAPTER



BACKGROUND INFORMATION

This chapter discusses some of the most used concepts and technologies in this thesis document. Production use and commercial software has introduced specific terminology approaching problems that are being solved in the projection mapping software and its interface. Nevertheless projection mapping research has been going on before media artists became curious about the topic.

Additionally, topics that are connected with the development of the `ofxPiMapper` are discussed in this chapter, such as the main target hardware platform, programming language and framework used, main target operating system, as well as version control system and tools for source code distribution.

2.1 Spatial Augmented Reality

Projector based spatial displays, a subset of spatial augmented reality displays [4, p. 83], is the academic term for what artists and other practitioners today understand with projection or video mapping. Most of the academic information about projector based spatial display systems can be found under the term “spatial augmented reality”. This section introduces key concepts of projector based spatial displays.

Spatial augmented reality is connected to augmented reality—more specifically to spatial augmented reality displays that “detach the display technology from the user and integrate it into the environment” [4, p. 7]. The term “projector based spatial display” suggests that a projector is its main component. Projector is just a light source, and a screen is required to make the projected image visible to the user. Therefore a space with walls and objects is required in order to be able to use one or more projectors for creating an illusion.

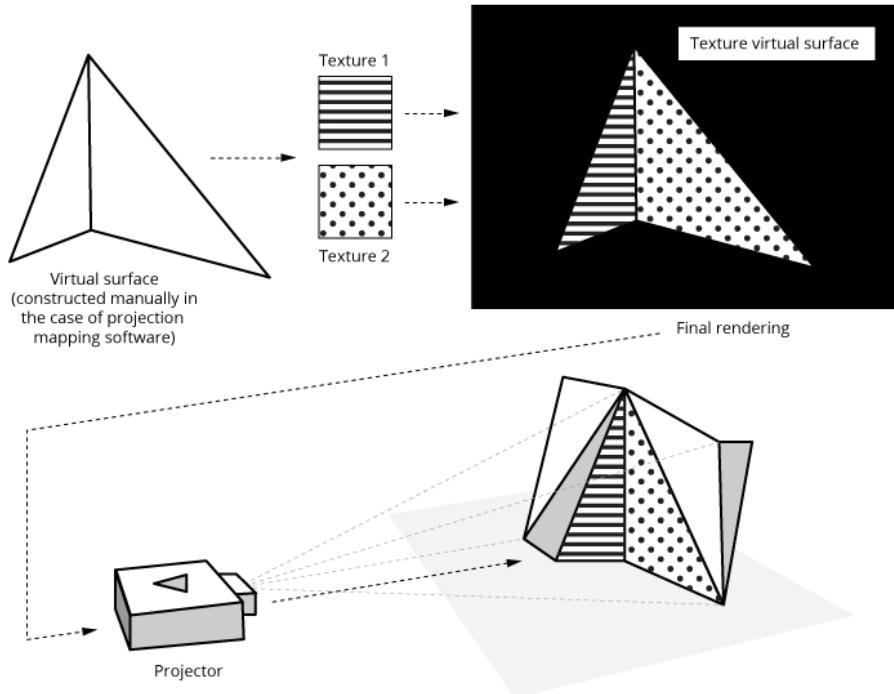


FIGURE 2.1. Projection mapping components

The tool in production (`ofxPiMapper`) is compatible with projector based spatial augmented reality displays where projector is used to “seamlessly project images directly on a physical objects’ surfaces instead of displaying them on an image plane (or surface) somewhere within the viewers’ visual field” [4, p. 87].

The way how most projection mapping software works can be described as forward and backward mapping of a texture source (image, video, generative animation), where the virtual surface that resembles the irregular display is constructed manually by the user of the software (Figure 2.1). The two-pass rendering method involving forward and backward mapping is described in detail in the book *Spatial Augmented Reality: Merging Real and Virtual Worlds* [4, p. 101]. The final render of the virtual surface (where the texture sources are already mapped onto it) is being projected on the real non-planar display using a projector.

2.2 Raspberry Pi

Raspberry Pi is a mini-computer that was developed in the UK by the Raspberry Pi Foundation with the intention of promoting the teaching of basic computer science in schools [86]. The idea to develop something like the Raspberry Pi appeared in 2006 “when Eben Upton, Rob Mullins, Jack Lang and Alan Mycroft, based at the University of Cambridge’s Computer Laboratory,

Raspberry Pi 1 Model A+	Price €	Raspberry Pi 2 Model B	Price €
Raspberry Pi	22.50	Raspberry Pi	38.50
Charger (5V, 1.2A)	6.40	Charger (5V, 1.2A)	6.40
SD Card (8GB, Class 10)	8.00	SD Card (8GB, Class 10)	8.00
Total	51.40	Total	67.40

TABLE 2.1. Raspberry Pi pricing (reichelt.de, 6 May 2015)

became concerned about the year-on-year decline in the numbers and skills levels of the A Level students applying to read Computer Science.” [48]

The first version of the Raspberry Pi was released on February 2012. Since then it has experienced many upgrades the latest being the release of the Raspberry Pi 2 model B on February 2015. The Raspberry Pi 1 has a ARM1176JZF-S 700 MHz processor and a VideoCore IV GPU (Graphic Processing Unit). Model A has 256 MB of RAM, model B—512 MB. In contrast, Raspberry Pi 2 has a 900 MHz quad-core ARM Cortex-A7 processor, which makes it approximately six times faster than the Raspberry Pi version 1.

The development of `ofxPiMapper` started with taking the Raspberry Pi 1 as the main target platform. Raspberry Pi was chosen because of its popularity and availability of information. The Raspberry Pi forum currently (6 May 2015) has 127,089 members in comparison to 10,066 members of the BeagleBoard [2] forum, which is almost 13 times less. Beagle Board, as said in the BeagleBoard.org website, is a true open hardware, community-supported embedded computer for developers and hobbyists.

Another advantage of the Raspberry Pi was the availability and proof of the possibility to compile and run `openFrameworks` on the platform. Andreas Müller and Jason Van Cleave were giving a workshop with the title “OpenFrameworks on the Raspberry Pi” at the Resonate 2013 new media festival in Belgrade—there one could see that compiling and running `openFrameworks` applications on the Raspberry Pi actually work [53].

Low price was another reason for choosing the Raspberry Pi as the main target platform. As of now (6 May 2015) the price of the Raspberry Pi 1 Model A+ is 22.50 €, but Raspberry Pi 2 Model B—38.50 € [20]. Refer to Table 2.1 for the price of a fully usable Raspberry Pi kit. Even the latest and most powerful version of the Raspberry Pi is cheaper as for example the BeagleBone Black (54.90 €) or CubieBoard (99.90 €). The Banana Pi seems to be a competitor with the price of 36.90 €, but it lacks the previously mentioned community aspect—only 8,343 forum members [22]. All prices taken from Reichelt Elektronik [20].

2.3 OpenFrameworks

Use of a framework was the preferred choice for developing a tool as it would save a considerable amount of development time. I considered the following creative coding frameworks as

valid candidates for the development of `ofxPiMapper`:

- Processing [24]
- OpenFrameworks [45]
- Cinder [3]

All of them are open source and popular creative coding frameworks with stable user and developer base. Processing was one of the first of the so called creative coding platforms and openFrameworks took a lot of concepts from Processing. Cinder, on the other hand, takes a different approach and relies on the knowledge gained from the C++ community. Cinder is also the youngest of the 3 mentioned frameworks, it had time to learn the best from the other two and avoid software architecture pitfalls.

The term “creative coding” means programming something expressive rather than functional [84]. Using programming for creating art is nothing new and it started in the 1960s [80]. Also the “Demoscene” can be considered a form of creative coding, although to gain “respect” from the demoscene community it is not enough with having just a good looking outcome, there has to be a technical challenge, and it has to be well technically executed [54, p. 97].

Processing was not chosen because it proved itself to be very slow on the Raspberry Pi platform as it uses bytecode compiler known as the Java Virtual Machine [34] and that means that only part of the code is translated into machine code while compiling—the other part is being compiled in real-time when the program is run. Probably it would perform better on the Raspberry Pi 2, but still it would perform worse in general than the other frameworks discussed in this section. Also as the development of `ofxPiMapper` started before there were any kind of news regarding Raspberry Pi 2, it was not possible to test the performance on it, therefore that becomes a potential topic for future research.

Cinder, as the newest of the frameworks discussed here, had time to learn a lot from the good and bad sides of Processing and openFrameworks. As stated on the Cinder website, it has been made to be intuitive to the C++ programmers specifically, building on the idioms and techniques the C++ community has developed over its long history. Unfortunately there was lack of information on how to successfully compile Cinder on the Raspberry Pi which could be solvable, but potentially too time consuming. If that was not the case, I would probably choose Cinder as the main development framework for creating `ofxPiMapper`.

I considered openFrameworks being the optimal choice for developing `ofxPiMapper` as it supports the Raspberry Pi platform out of the box, has a well defined add-on system as well as it provides foundation for the types of projects `ofxPiMapper` intended use would be. Furthermore if the tool would work well on the Raspberry Pi, it would run event better on conventional (non-ARM) systems running operating systems like Mac OS or Windows, Linux, iOS and Android not being an exception.

OpenFrameworks was created as a “bare-bones” alternative to other similar frameworks available at the time. It became popular over time and now is a feature rich framework for doing audio-visual work [45]. OpenFrameworks has a considerable online community—6800 forum users [23], and that makes it easier to find help if needed.

2.4 C++

OpenFrameworks is a framework that is made by combining libraries and other building blocks of the C++ programming language. C++ has a long history and even if there are new programming languages that might seem more modern, C++ is still considered the programming language of choice for performance intensive applications [1]. C++ is both low and high-level. C++ offers the possibility to access hardware specific instructions by using C or assembly language directly and then switch back to the object-oriented abstraction level once done.

Since the beginnings of the development of the C++ programming language in 1979 [70], experience and resources have been gathered and stored on-line, as for example Cppreference.com [11] that focuses on providing explanation about the language details with examples that can be edited and compiled directly on the website. There are 367,993 questions on Stack Overflow question and answer site [68] with the tag “C++” (14 May 2015), and there are books on various aspects of C++ available.

GCC or the GNU Compiler Collection [27] is used for compiling openFrameworks C++ applications on the Raspberry Pi. I mainly develop `ofxPiMapper` on my Mac OS X computer with the Mac OS X version of openFrameworks. I do not use cross-compiling tools, once a working state of a feature is developed on my Mac computer, I upload, compile and test it on a Raspberry Pi. This approach works as good as it would with a cross-compiling solution—openFrameworks is build to run same code on all its supported platforms the same way. I decided to use the time that would be needed for setting up a cross-compiling environment for the actual development of `ofxPiMapper`.

2.5 Linux

As stated in the Raspberry Pi website [48], Raspbian is the recommended operating system for the Raspberry Pi. Raspbian [75] is a Linux distribution that is based on the Debian Linux [14]. There are other alternatives, but most of them are based on Linux [35]. Raspbian is an optimized version of Debian, made to run as efficient as possible on the ARM powered Raspberry Pi. By default it starts up in command-line mode and can be configured to start in desktop mode as well.

The Linux kernel is an open source computer operating system core module developed by Linus Torvalds since 1991 [35]. Its development has not stopped ever since and right now it's at

version 4.3-rc2 [76]. Linux is an open source clone of the Unix operating system, developed by Ken Thompson, Dennis Ritchie, and others in the 1970s [59].

The Unix philosophy is an interesting topic on its own. Its focus is on command line applications that can be chained together, thus the quality and performance of the system is defined by how well do the applications collaborate with each other. Instead of having monolithic applications that do all on their own, Unix-like systems tend to have a set of reusable tools that can be chained together in many custom combinations.

2.6 Git and GitHub

Git is a distributed version control system that allows one to create snapshots of software development progress throughout time. The first version of Git was published in 2005 by Linus Torvalds (the same person behind the development of the Linux kernel) and its main use, and the reason for creating it, was the management of the Linux kernel source code [7].

It proved itself to be well built and useful for many other software development projects (additionally to the Linux kernel), its main strengths being speed and distributed, non-linear workflows. That means software developers can clone the project on their local machines and continue working without having a network connection to the main remote code repository. They can easily merge their work with the main code base once network connection becomes available. Git has made the branching and merging features effective and fast. Branching and merging features have been the Achilles heel for other revision control systems, e.g. Apache Subversion [72].

GitHub [26] is a web based Git repository hosting service, which many open source software projects have considered their home. GitHub offers an easy to use web interface for working with Git repositories and a set of web-based software development process management tools. One is able to submit per-repository issues, assign them to collaborators, set milestones, organize releases and more.

The GitHub website (github.com) was launched in April 2008 by Tom Preston-Werner, Chris Wanstrath, and PJ Hyett after it had been made available for a few months prior as a beta period [26]. There are more than 10 million repositories on GitHub currently, according to a GitHub blog post from December 24, 2013 [16]. GitHub has been recognised as one of the 100 most popular websites on earth according to the Wired magazine [38].

Also openFrameworks has chosen GitHub as the main platform for hosting and managing the source code. GitHub is also recommended to developers for hosting their openFrameworks add-on projects [44]. This is also the source for the decision to host ofxPiMapper source code as a repository on GitHub.

CHAPTER

3

EXISTING SOFTWARE

This chapter discusses state of the art in the projection mapping software field. Currently there are different open and closed source tools available, some of them are gratis, others are commercial. This section includes analysis of software tools used by projection mapping practitioners. User and Internet research were used to get a list of software for analysis in Chapter 4. User research was based on a simple user questionnaire and the hardest part was to find enough people who would be knowledgeable to fill it. The Internet research part was based on using web search engines (such as google.com) to find websites that contain information on tools used for projection mapping today.

3.1 User Research

To understand what software tools for projection mapping are being used by the community, a user research was necessary. The research was also needed to understand what features users of the software like and dislike, what could be improved or added. The latter would then benefit further development of the `ofxPiMapper`. Before the user research it was also assumed that some of the most used features could take too much time, or would be impossible to implement.

The user research questionnaire consists of six simple questions (Table 3.1), four of them were supposed to get a broader overview of the tools, features and workflows of current projection mapping software from the user's perspective. It was a challenging task to gather enough people filling the questionnaire published on Typeform website [79] as projection mapping is still a complicated matter and it requires a lot of expensive hardware to get started.

1.	What is your name?
2.	What is your email?
3.	What kind of software tools are you using for projection mapping?
4.	What features do you find the most important in the projection mapping tools you are using?
5.	What features do you think are missing in the projection mapping tools you are currently using?
6.	Please describe your workflow in your favourite projection mapping tool.

TABLE 3.1. Projection mapping software user questionnaire

32 projection mapping practitioners replied and filled the questionnaire thus providing me with valuable data.

The most popular software applications used by the practitioners are MadMapper, Resolume Arena and Millumin, MadMapper being used by 28% of the respondents, Resolume Arena by 13% and Millumin by 9%. You can see the full comparison in Figure 3.1.

The most important features for the users of projection mapping software of their choice are interoperability with other software, ease of use, OSC and MIDI protocol support, triangle warping, masking, edge blending, video playback, high precision adjustments and 3D mesh loading. Syphon was mentioned by six respondents which show that using existing software with projection mapping tools is preferred. Nine users mention terms that stand for interoperability, e.g. OSC, Syphon and MIDI. At least five people mention terms relevant to possibility to use generative sources.

Among the missing features of the software used by the respondents are undo/redo support, animated GIF sources, remote control (via OSC or MIDI), generative sources (FBO sources), improved precision, multiple and easily switchable states of a mask, 3D mesh import and automatic mapping setup/calibration.

Users filling the form also had to describe their workflow in their favourite projection mapping tool. Almost half of the respondents described a very careful and sequenced approach—first sketching, deciding, preparing a 3D scene, then mapping and doing minor adjustments at the final phase. Most of the users tend to like an improvised approach—being able to adjust the final projection mapping output real-time with the final image, video or generative source used as texture.

From the gathered responses I can draw conclusions that users want flexibility, interoperability, precision and ability to improvise without risk. The projection mapping software they choose has to be easy to use, stable and fast. It needs to accept random decisions and let the user to undo mistakes or wrong decisions.

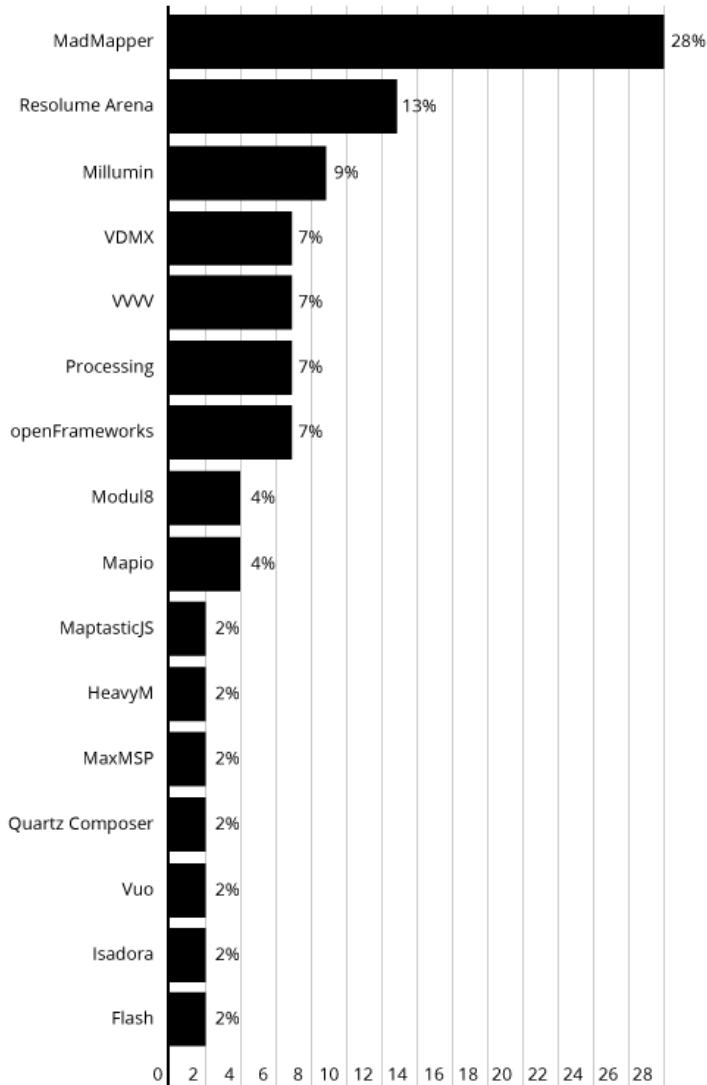


FIGURE 3.1. Projection mapping software used by the community

3.2 Internet Research

There are Internet resources dedicated to projection mapping where one can find recommendations on software and hardware for the task. Some information about current software tools used by the community can be found there. One of the Internet resources is Projection Mapping Central [32]. As said in the About section of the website, it is a community resource to learn about, promote and experience projection mapping. Articles mostly about different projection mapping projects can be found there along with list of tools. The Tools section of

the website has a comprehensive list of software that can be used for projection mapping. I split the list in 5 categories for a better overview.

Mac OS X projection mapping software

- MadMapper
- Blendy VJ and Blendy Dome VJ
- BlendBuddy
- FacadeSignage
- Visution Mapio
- MWM—Multi Window Mapper
- Splash

Projection mapping software for other operating systems/platforms

- OpenFrameworks Virtual Mapper
- Painting with Light
- DynaMapper for iPad
- Mesh Warp Server
- HeavyM
- VPT
- LPMT
- Torsion Blend

Code/framework/node based projection mapping tools

- Procamcalib
- C++/Qt Projector-Camera Calibration
- VVVV
- Touch Designer

VJ and audiovisual stage design software

- Arkaos
- Resolume Arena
- Millumin
- MXWendler

Hardware and advanced media server software

- Green Hippo
- Avolites
- Vioso
- Coolux
- d3 Technologies

There are other sources of information on the Internet, e.g. projection mapping Wikipedia page [85] or the Video Projection Mapping Tumblr blog [82] which lists videos with different video mapping projects. I did not succeed in finding a better available projection mapping software overview as the one on the Projection Mapping Central website by Jones et al. [32].

As one can see, that is a good list of software that complements the list of tools gathered from the user research. The price of the tools vary—some of them are free (Splash), for others the price varies from 320 € (MadMapper) to \$1,200.00 per year (d3 Designer) and more depending on the licensing terms. All of the software requires powerful hardware to be run on.

In the following chapter I will analyze and discuss the software that I was able to get and try

out with the hardware and operating systems available to me, focusing on the main features advertised by the software producer and the features I would use for a test projection mapping project of mine.

EXISTING SOFTWARE ANALYSIS

In this chapter I will analyze the projection mapping software tools based on the list compiled by the user and Internet research done in the previous chapter (Chapter 3). Some of the tools found I have used myself before, as the MadMapper or VDMX for example.

This chapter can be considered as benchmarking, comparing other projection mapping applications to understand what would be the best practices to use for development of ofxPiMapper and what to develop. In the Benchmarking Book by Tim Stapenhurst [69] benchmarking is said to be “applied to a wide variety of activities that organizations undertake to compare their performance levels with others and/or identify, adapt, and adopt practices that they believe will improve their performance.” Instead of companies we will be comparing different projection mapping applications which, in fact, have a kind of organization behind the scenes.

In the following sections I will describe each of the software from the ones found that I was able to get running on the hardware available. I will start out with a general description of the software, what are the main features promoted by the creator of the software, what were the ways and licensing rules of getting the software, my impressions after launching it the first time, a short review of using it for a simple projection mapping project and a summary of advantages and disadvantages.

4.1 Specialized Projection Mapping Software

In this section I discuss the software whose main purpose is projection mapping. This kind of software does not provide much means for content editing. In some cases it might provide a timeline for video sources and some real-time color correction options, but the most software features described in this section are related to projection mapping specifically. This kind of software would provide you with ways of using specific kind of image or video input and a wider set of possibilities to use the incoming image as a texture for applying it to a real-life object via projector.

4.1.1 MadMapper

MadMapper can be considered as a standard of today's projection mapping software. Not surprisingly, 28% of the user research questionnaire respondents mentioned it as a tool they use for projection mapping. MadMapper is currently developed by GarageCUBE and 1024 Architecture. As said on the website (madmapper.com), the software is built around the idea of sharing video content between applications, a software framework called Syphon is used to use the underlying Mac OS X operating system capabilities of sending moving images real-time from one application to another. MadMapper runs only on computers running the Mac OS X operating system and Mac OS X version 10.6 or later is required to be able to use Syphon sources.

One can download a trial version of MadMapper from the project website. It is fully functional, except there is a watermark all over the main output image that is sent to the projector. Another limitation is that it is not possible to save projects, it can make a big difference if setting up of your projection mapping project takes more than a hour—you will have to redo everything if you exit the application or your computer crashes.

If you intend to use it seriously, it is possible to buy a MadMapper license for 2 computers for 320 € (25 Sep 2015). Various discount options are available. The licenses are transferable from computer to computer and there is a system that prevents you from using more than two copies of the software.

The user interface of MadMapper (Figure 4.1) looks attractive, and after taking a closer look, one can discover keywords and icons that can lead to the right direction in terms of using the software. The user interface is divided into three columns. The leftmost column contains various settings, the main ones being related to source selection. The middle column is where you see your source and select various regions of it for projection mapping in the rightmost column.

After using it for the test project, it was clear that MadMapper is easy to use and as stated in the MadMapper website, it “demystifies” the projection mapping process. The interface is clear and the main features are easily accessible. It is self-explanatory—select the source, then

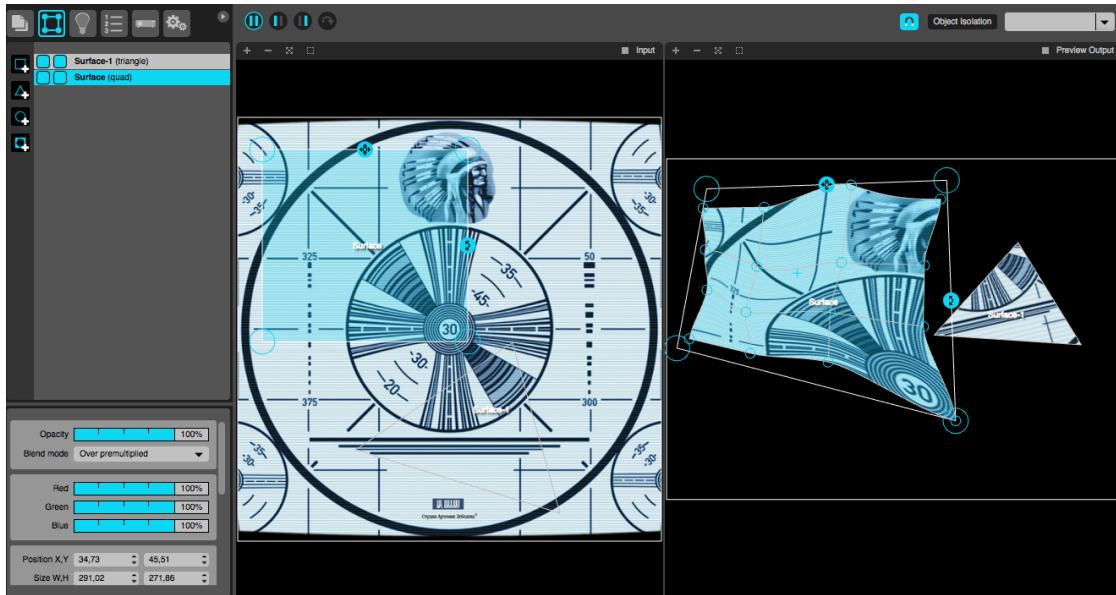


FIGURE 4.1. MadMapper user interface

select/crop parts of the source to be projected on the irregular screen and finally move and transform the selected areas of the source to match the objects in the range of your projector's output beam.

One of the main advantages of MadMapper is its wide application range—one can use it for simple video mapping projects and there is also a possibility to combine it with other software in order to make use of generative content. For example it is possible to create a generative application using openFrameworks and Syphon, then select the Syphon output of the application in MadMapper and projection map it. Additionally one can use Quartz Composer visual programming environment available on Mac OS X to create generative content without the need of Syphon.

MadMapper is almost ideal, but there are also disadvantages as the possibility to select only a single source for projection mapping. Now if one wants to combine multiple sources, the “gluing” process of the sources has to be done beforehand while editing or real-time with other software. MadMapper would be useless for some projects where it might not be possible to choose or afford computers running Mac OS X as it is the only platform it supports.

4.1.2 Visution Mapio

Visution Mapio is another tool that offers a broad spectrum of features for one's projection mapping project. Mapio runs on Windows and Mac OS X and has two variants: Mapio 2 Lite and Mapio 2 Pro. It is being developed by Ivan Ryaboff [61]. Free download is available on the Visution website (visution.com). To use it for commercial projects one has to buy a license. To

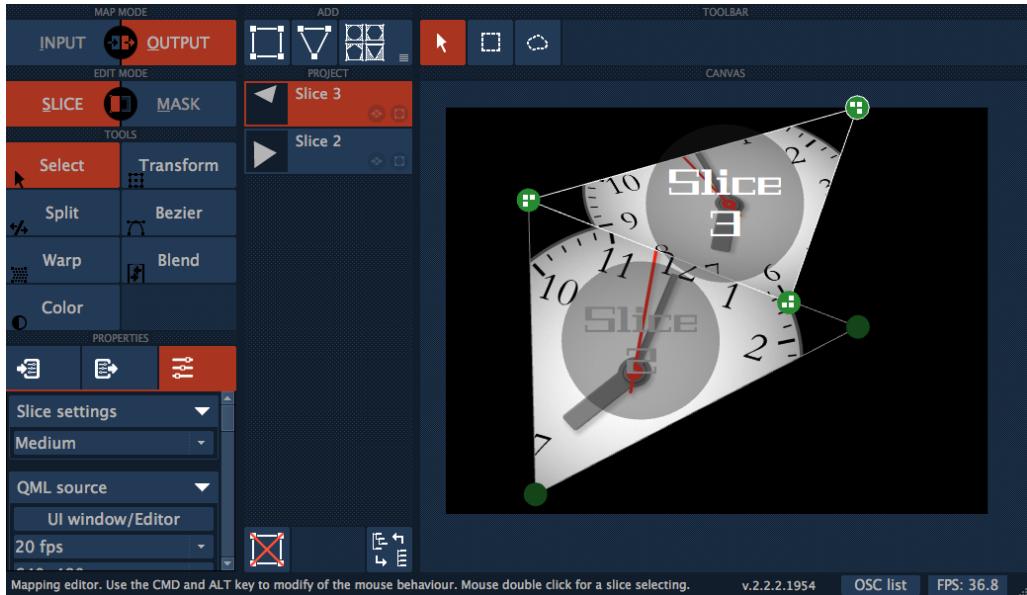


FIGURE 4.2. Visution Mapio user interface

buy Mapio 2 Lite license one will have to spend 150 €. The price of Mapio 2 Pro license is 300 € (25 Sep 2015).

The Lite version offers features like Syphon/Spout support, QML patches, edge blending tool, bezier tool for slices and masks, OSC control and GPU acceleration. Spout is a similar technology to Syphon, but for the Windows operating system [67]. QML is a user interface markup language developed by Nokia for creating highly interactive user interfaces [5]. The Pro version additionally comes with Kinect support, DMX capabilities, output to serial port devices and other advanced features. Kinect is gesture control device by Microsoft that allows one to use one's full body to play computer games [47]. DMX or Digital Multiplex is a digital communication standard that is often used in stage lighting and effects [77].

I decided to go with the Lite version as the supported features seemed more than enough for my simple projection mapping project. Even though the user interface (Figure 4.2) looks nice and not bloated with too much features, it took some time to understand how to approach projection mapping with the tool. The workflow that has to be used with Mapio is different from the one in MadMapper. Projection mapping surfaces are called "slices" and one can assign a source per slice. Sources are texture providers that can be image, video, color, Syphon input, QML interface, connected video cameras or adjustment patterns. A collection of existing QML applications come bundled in with Mapio e.g. analog clock and Twitter feed. It is also to edit the QML source code directly in Mapio. The QML source feature allows to build generative and interactive compositions in Mapio without the need of a 3rd party application.

The overall impression of the software is positive. The website of the tool looks good and lists features that can be useful in many projection mapping project scenarios. The

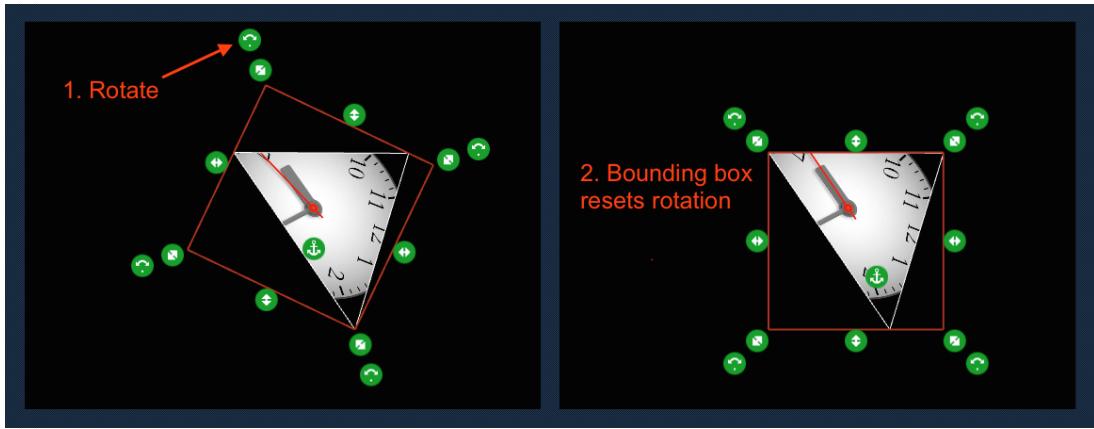


FIGURE 4.3. Visution Mapio rotation tool behaviour

user interface is aesthetically pleasing, but sometimes not too easy to use. For example in comparison with MadMapper, Mapio lacks simultaneous slice (or projection mapping surface) preview on the same screen while adjusting the texture mapping for the selected slice. The slice transformation tool is confusing as it looks similar to the ones implemented in many graphics related applications, but it works different—after using the rotation handle, the transformation bounding box rotates back to its initial angle (Figure 4.3).

Visution Mapio would be my tool of choice if I had to develop a projection mapping solution on a Windows PC and would need to projection map a highly interactive user interface. The ability to use QML applications in Mapio is a big plus. If I were able to influence the development of this software, I would rethink the way the user interface is built. One of the first improvements would be introduction of a split screen that would allow one to see the projected output while editing the texture mapping. I would find a better place for the QML application settings as currently the area for them is too small. I would also emphasize projection mapping surface creation icons over available modifier buttons as one can not use the modifiers before a surface is created. Now the modifiers generate higher contrast than the surface creation buttons.

4.2 Software with Additional Projection Mapping Functionality

In this section I discuss real-time video software where the possibility to use it for projection mapping is just an additional feature. By feature here I mean a tool integrated with the core software that has a predefined user interface. User is able to access a specific projection mapping interface by navigating the menu tree of the discussed application. As projection mapping is closely related to audiovisual performances and Vjing, tools designed for the purpose show an increasing trend to additionally provide projection mapping functionality.

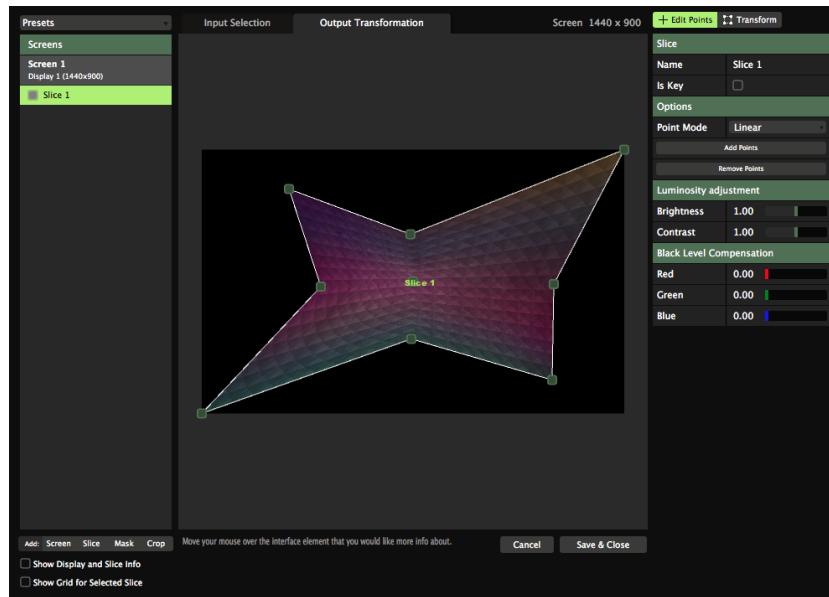


FIGURE 4.4. Projection mapping settings in Resolume Arena

4.2.1 Resolume Arena

As said on the website (resolume.com) Resolume is created by Edwin de Koning and Bart van der Ploeg together with Tim Walther, Joris de Jong, Menno Wink, Reinier Noorda and a team of specialized freelancers. They are based in Hague, Netherlands. Their main motivation of creating the software back in 1998 was just to become better VJs as there were a lot of limitations when using VHS tapes for the purpose. In the Shop section of the Resolume website there is a pricelist and one can buy a licence for one computer for 299 €.

After launching the software for the first time, it was not easy to understand where to begin. It took me some time to understand the user interface of the many features of the software, and it took even more time to find the projection mapping options as the way things work in Mapio is different from the way I am used to by using MadMapper. It appeared that there is a tool for adjusting the projection mapping of the final visual output (Figure 4.4) before it is sent to one or more projectors. The projection mapping tool is based on “slices” or in other words instances of final output. For each slice it is possible to define an input region—the cropped part of the image that will be visible on the slice, and the output transformation or projection mapping.

Only quad regions could be defined as input in the tool, making it complicated to projection-map a 3D object with triangular surfaces. On the other hand, the projection mapping tool seems to be a valuable addition to a VJ software like Resolume, but it has its limitations regarding projection mapping itself.

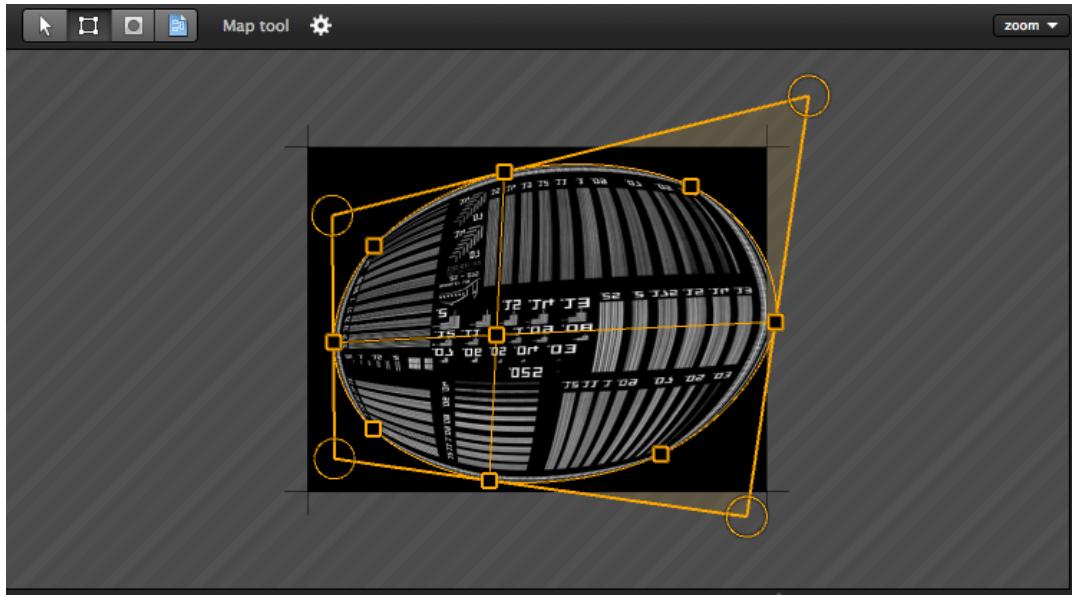


FIGURE 4.5. Millumin projection mapping user interface

4.2.2 Millumin

Millumin is a product of Anomes (anomes.com) and one can get it through the project website (millumin.com). Its main application area is not only projection mapping projects, but audio-visual shows in general. Projection mapping is just one of the features the software offers. The price for the full version of the software is 599 € (tax not included). Two licenses are included—one for the computer you work on, another for the performance.

After opening the Millumin application the first time, it was not clear where to start and the Millumin Tour was necessary to get familiar with the basics of the interface. After finishing the seven step tour, it seemed like the extract of the best audiovisual-show-related features from Adobe AfterEffects. There is a timeline and a possibility to add layers, create compositions with animated source parameters (including animated projection mapping). The output settings panel also seemed very straight-forward and with many useful settings like the choice of output resolution, blending between multiple projectors, soft-edge and color adjustments (in case there are noticeable color difference between the projectors used).

Regarding the projection mapping feature (Figure 4.5), there is only one type of surface available, which is the quad. One can do perspective and mesh warping with the available options. Bezier warping is available that can be very useful for curved screens. The software was not well suited for the test project as it was not possible to successfully cover the triangular surfaces of the spatial object. It would be possible with additional work with the sources, but it slows down the process and thus makes it less fluent (than in the case of MadMapper).

Millumin can be useful for audiovisual shows in general, but if we look at the projection

mapping feature, there is still space for improvement. Nevertheless, with Millumin it is possible to use multiple sources for projection mapping which is not possible in MadMapper.

4.2.3 VDMX

VDMX is professional VJ software for computers running the Mac OS X operating system. It is being developed by a company called Vidivox (vidivox.net), based in the capital region of New York. As said in the website, since 1998 the company has “focused on making outstanding tools for visual artists and performers around the world.” Vidivox is also involved in various open-source projects, as the Hap Video Codec, VVOpenSource, VVFFGL and Syphon.

One can download a demo version of VDMX, but I did use it for commercial projects before and thus have the full version already. The current price (28 Jun 2015) of VDMX is \$349 (313 €) and you can use a single license to install VDMX on three different personal computers.

As described on the website, VDMX is meant to be very flexible as no audiovisual project is the same. It is supposed to work well with other software one might use. Audio reactivity is mentioned as one of the most important features of VDMX, one can expect that syncing visual content with audio can be done in this software. VDMX is able to render Quartz Composer patches—that enables it to be used with generative content. Remote control (via OSC and MIDI) is also promoted as one of the core features of VDMX.

I still remember the first time launching the application for the first time approximately four years ago. Nothing much has changed since then—the interface still looks very complicated and one must spend some time reading manuals, watching tutorials and trying out things in order to understand how to approach VDMX correctly. There are a lot of configurable features and in a way it is a kind of visual programming language on its own.

The visual output of VDMX is organized around the concept of layers. Visual sources as camera input, video files and generative Quartz Composer compositions can be assigned to a layer. One can add numerous effects and change parameters of a specific layer (or layer group) including a simple quad projection mapping setting for adjusting the perspective warping (Figure 4.6). The tool does not offer a possibility to do mesh warping, there is just a panel where one can click and drag one of four corners of the quadratic source layer.

For the test project it is not enough with having just a quad perspective warping feature as that makes it complicated to work with triangle surfaces. On the other hand VDMX is not supposed to be a projection mapping software on its own, but can be used together with an application that is specifically made for the purpose. The projection mapping capabilities that come together with the VDMX software can be useful in situations where the projector keystoning functionality is not enough.

VDMX fits as software being a central part of a complicated audiovisual show. In regard to projection mapping, VDMX alone can not do much and one has to consider using additional software.

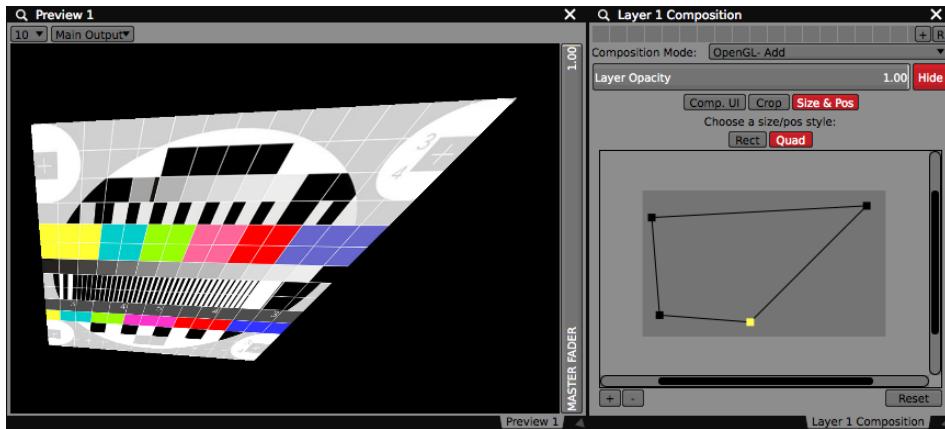


FIGURE 4.6. VDMX projection mapping user interface

4.3 Creative Coding Frameworks

In this section I discuss creative coding frameworks which can be used in order to build custom projection mapping tools or that have one which is integrated or community supported. I use the term “creative coding” here as it is widely used by the community and one won’t find the same information when using a term that could seem more correct—“creative programming”. Daniel Shiffman in his book *The Nature of Code* [63] uses the term “creative coding” when talking about programming environments as openFrameworks, Processing and Cinder. Also in my book *Cinder—Begin Creative Coding* [56] I use the term to refer to the activity of using the Cinder C++ based programming framework.

4.3.1 vvvv

According to the vvvv website (www.org), “it is a hybrid visual/textual live-programming environment for easy prototyping and development.” It is developed by the vvvv group: joreg (full name not available), Max Wolf, Sebastian Gregor and Sebastian Oschatz. They are based in Berlin, Germany. One can use vvvv free of charge for non-commercial projects. Commercial licenses are available starting from 200 €.

After downloading the software and trying to run the setup executable, one will notice that some dependencies have to be installed in order to be able to launch vvvv. After launching it for the first time, a blank window appears which left me thinking, where I have to go next? Probably it was because I downloaded a beta version and it is a bug. The setup process of the beta version was confusing because tools used by the vendor for debugging were showing up.

As the software is supposed to be a live-programming environment [10] which allows one to combine small and specific parts into custom formation, it takes some time to get familiar with the workflow of the software and it takes even more time to understand how to do projection



FIGURE 4.7. User interface of badMapper (Source: <http://vvv.org/contribution/badmapper>)

mapping with it. In short the software is not made for projection mapping specifically, but functionality that one can use to create a custom projection mapping solution is available. It appears that vvv does not provide a projection mapping solution out of the box, one has to design it him or herself. Nevertheless there is a contribution available that is called the badMapper (Figure 4.7), and as said on the contribution's website (<http://vvv.org/contribution/badmapper>), it is inspired by MadMapper.

Among advantages of using vvv for projection mapping is that one can create a very specific setup out of the building blocks available and combine it with generative content directly. A disadvantage of using vvv for projection mapping is the setup complexity and requirement of learning at least the basics of vvv.

4.3.2 Processing

Various open source programming environments exist and Processing is one of them. According to the website (processing.org) "Processing was initiated by Ben Fry and Casey Reas and it is developed by a small team of volunteers." One can use it completely free of charge, but a donation is encouraged to support its further development. Processing runs on Windows, Linux and Mac OS X platforms.

Processing is a simplified programming environment built on top of the Java. There are no out of the box projection mapping solutions integrated into Processing, one has to build it him/herself. One way of approaching projection mapping with Processing is to use solutions

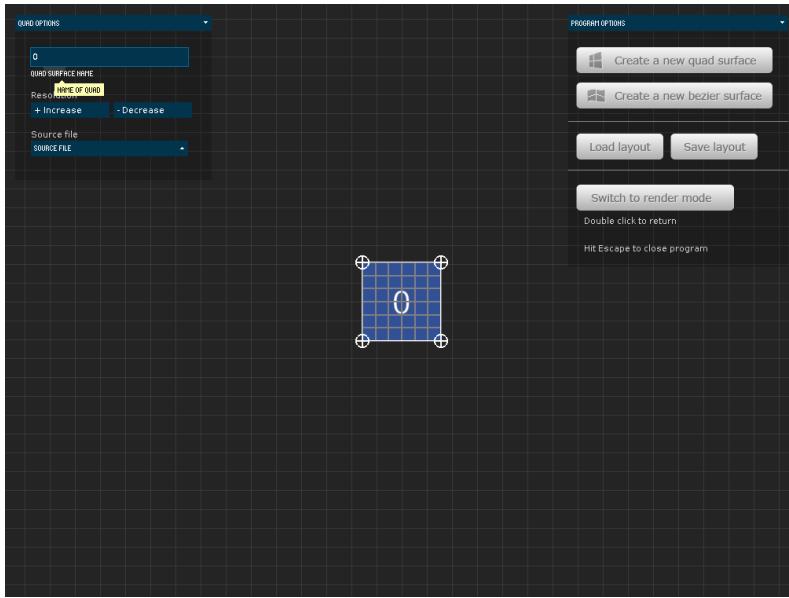


FIGURE 4.8. SurfaceMapperGUI interface

like the SurfaceMapperGUI (<https://github.com/jasonwebb/SurfaceMapperGUI>). It provides the possibility to create quad perspective and quad bezier surfaces, image and video sources can be applied to them as textures. The surfaces can be transformed by using a simple user interface (Figure 4.8).

Processing is intended for inexpensive prototyping, but even though the software is free, a considerable amount of time has to be spent in order to learn how to properly install dependencies of projects like the SurfaceMapperGUI. It took me multiple attempts to get the SurfaceMapperGUI code running, but once I did, I was positively surprised by the clean and “warm” interface of it. Although it does not have a lot of features, the user interface design seems almost perfect for a single-screen application.

4.4 OpenFrameworks Add-ons

The add-ons for openFrameworks are distributed as source code (ofxaddons.com) and have to be copied in the openFrameworks add-ons directory. Software based on openFrameworks is usually available open-source, but in some cases it is also possible to find downloadable binaries that are compiled by the author of the project for a specific platform. Below is a table (Table 4.1) with openFrameworks projection mapping add-ons and their main features.

It is important to mention that one of the most important features of these add-ons are compatibility with other openFrameworks add-ons. As one can see all of the add-ons tend to provide mainly one simple functionality that can be combined with all the other possibilities openFrameworks offers. On the other hand, none of the tools provide a really simple solution

Add-on	Features
ofxKinectProjectorToolkit	Real-time Kinect depth camera-to-projector calibration.
ofxReprojection	Tool for dynamic projection mapping using depth camera and projector.
ofxMtlMapping2D	Quad, quad mesh and triangle surface types, masks, single-screen.
ofxProjectionMask	A projection masking tool that supports texture source masking only.
ofxVideoMapping	Projection mapping tool with the possibility topojection map quad surfaces only.
ofxDome	Add-on for dome projection mapping [13] only.

TABLE 4.1. OpenFrameworks projection mapping add-ons

for using it on the fly, right after downloading. It is possible to run the example applications, but none of them is ready for production use right away. One has to spend some time for understanding the underlying concepts of the add-on and creating a custom solution.

The tools described in this section are applicable for simple projection mapping solutions with no budget, where the main person being in charge is the artist him or herself. Scenarios might include projection mapping prototypes or installations that do not require the ability to stay on for long periods of time. Although, that can be achieved, but it requires additional time and effort.

4.5 Summary

As this analysis shows, there are a lot of projection mapping tools for different kind of scenarios already there. It might raise a question—why another tool? The answer could be related to the fact that none of the tools found is able to run flawlessly on the Raspberry Pi pocket-size computer. Another answer could be related to personal interest of finding out how projection mapping software works behind the scenes. Nevertheless the collection of tools gives me enough ideas on how to realize almost any projection mapping project imaginable, I still believe that a tool like `ofxPiMapper` could make a difference.

None of the tools discussed is perfect. Each of them has an aspect that makes it better than any of the other tools. Most of the tools are suitable for simple projection mapping scenarios. Most of them provide a way of using generative source input. For each of the tools discussed it is possible to add something that is missing, could be improved or cloned from another software.

MadMapper can be considered suitable for many projection mapping project scenarios when using a Mac OS X computer. Visution Mapio can provide great help when using a Windows PC. The tools that can run on Linux (SurfaceMapperGUI) lack production quality features. As stated before—none of them are supposed to run on the Raspberry Pi. OfxPiMapper could be a success if it would implement a mix of overlapping features in existing projection mapping software, would ask for less resources hardware-wise and provide an intuitive way for setting up a production-ready project fast.

SOFTWARE DEVELOPMENT

In the previous chapters supported the idea of creating a new projection mapping tool that would be optimized for low-end hardware. In this chapter I am going to talk about its development.

When you look at the existing projection mapping software, there are many things that users expect to be there. As the undo feature for example—one could not imagine a modern text or image editor without it. Saving and loading a document is as important. From the software development perspective all of the features that are taken granted by the user have to be developed and its time consuming. Furthermore, the more features get implemented, the more lines of code the project contains and it has to be well organized in order to successfully maintain it. How to do that? This is one of the questions that I tried to answer through the software development process of `ofxPiMapper`.

At the beginning of the project I also decided to take a practical approach—I started writing code and tried to reach a state where I could compile and use the software. This approach can be compared to the bottom-up design which in contrast to top-down is focused on starting out from atomic elements of the system and then working towards generalities [37, p. 111]. The top-down design approach is about having a clear plan and complete understanding of the system before the actual development of it has started. It is hard to say which of the approaches is better, but from the perspective of the development of `ofxPiMapper`, bottom-up approach seems to win because of the following aspects. There are many things that have to be tested before and the best way to do it is by creating components that can be reused later if the tests pass. Creating a good software development plan requires a lot of experience in the field; Furthermore—experience related to the projection mapping software development

is needed. I believe that using the bottom-up design approach is less risky and can create reusable knowledge and experience that can be used for other projects in the future even if the development of the current project fails.

From there I tried to identify potential problems and searched for information on how to solve them. This chapter is a summary of my learnings. It is a combination of personal experience and available knowledge of traditional and not so traditional software development processes. It starts out with a Software Requirements section where I describe how I learned to approach the requirements of what the software should be capable of. Next I describe the ofxPiMapper software from the user interface perspective. From there I switch to Software Architecture section where I describe the design of the software, how the source code of the software is and could be organized to make it easier to maintain and add new features in the future.

5.1 Software Requirements

At the very beginning of the development process I did not identify software requirements as a separate phase of the development process. Nevertheless, I had a vision of what the software should be capable of as well as how it should look and work. After numerous iterations I identified that the whole process can be divided into multiple separable parts and one of them would be the software requirements phase. The requirements phase is usually the first step in development of a project in the so called waterfall software development model [40]. In the waterfall model it is followed by the design and implementation phases and that aligns with the top-down approach discussed earlier in the text.

According to a study by Michael Krigsman [33] 68 percent of IT projects fail because of poor requirements. On the other hand, the Agile development process suggests not to focus on requirements at the beginning of the software development process as they will change later anyway.

Requirements cannot be captured at the beginning of a project, because users do not know what they want. Even if one managed to write a requirements document, it would be useless because requirements will change through the project. [39]

Through examining both of the sources I came to a conclusion that both of the hypotheses are extremes and that I should take the best out of both practices—specifying minimal requirements at the beginning of the development process as well as reviewing and updating them on a regular basis throughout the software development process. It was not too hard to define the first set of requirements for the software as it was also clear that it would not make sense to proceed with the project if it was not possible to fulfill them. The first requirements were as follows:

- The tool should have the very basic functionality of a projection mapping software—it should be possible to create a virtual surface and assign a texture to it.
- It should be possible to transform the virtual surface on the screen by clicking and dragging on its corners; the texture assigned to the surface should morph with the surface.
- It should run on the Raspberry Pi.

After successfully fulfilling the first set of requirements it was clear that it is worth proceeding and I continued to add new requirements gradually as the project progressed.

One of the next major steps was to get the software to a state which would allow other people to use it for their projects. That involved creating an already longer list of requirements. I did not go for a full set of features as in the MadMapper software as I am not a team of software developers and thus I can not develop at the speeds. I tried to find a balance between my skills, experience, and imagined list of features a basic user of projection mapping software would want. To move things forward quickly, I put myself in a position of a test user and tried to figure out, what would be the basic features I would need to realize a simple projection mapping project. By assuming that I would like to create a three-dimensional object consisting of flat triangular surfaces as the screen and have animated content as its texture, I created a following list of requirements additionally to the initial ones:

- It should be possible to create multiple triangular projection mapping surfaces and delete them.
- It should be possible to save the composition so the layout and transformations of the surfaces are still there after restarting the software.
- It should be possible to use an animated (preferably generative) source as the texture.

After succeeding to fulfill this set of requirements, there was a next one and so on. It can be seen as a never ending process. I must also add that at the time I was trying to match the first sets of requirements, they were not so clear as they are written in this document. After multiple attempts, failures and successes I finally managed to get in hold of the development process and the one described here is the one I try to use currently.

The current development of `ofxPiMapper` could be characterized as a set of sprints, from the Scrum methodology [74], which are carried out in a random fashion. Each sprint has a specific goal, list of requirements and there are sometimes long unspecified delays between them. Each sprint ends with a working version of the software. The length of a sprint usually is 1 or 2 weeks depending on how much work has to be done. Right now the goal of a sprint usually is one specific feature—like the undo feature (for each of the existing user interaction

related features) for example.

By developing the software over a year now (first GitHub release was on May 10, 2014), a more comprehensive list of requirements has been created. Right now the actual requirements for the next couple of sprints are as follows:

- Users should be able to use software that supports the OSC or Open Sound Control protocol [88] for remote-controlling `ofxPiMapper`.
- Users should be able to do mesh warping and toggle perspective warping for quad surfaces.
- Developers should be able to add new features as easily as possible.

As you see, the software requirements in the case of `ofxPiMapper` do not only specify requirements for the end users of the software, there are requirements for another type of users—developers, which are advanced users with demand for new advanced features and also knowledge about how to add them. For the developers to be able to add new features easily, the code base of the project should be easily approachable. There are many factors that have to be taken into account, one of them is the architecture of the software and I am going to discuss it in greater detail in the Software Architecture section (Section 5.3).

To sum up the Software Requirements section, the learnings from the `ofxPiMapper` project are that it is much easier to build the requirements in an iterative way—start lean and use small incremental steps after each development phase or sprint to improve the overall requirements. There always will be new features or improvements of old ones, and nobody actually knows the ideal state of a software, so spending too much time on planning can be a waste of time. For a lone developer it is also easier to get some satisfaction out of the process if there are more small successful steps rather than one big possible failure in the future.

5.2 User Interface

Be it command line interface or a full-blown windowing system, our first experience and understanding of computers has been built on top of the ways the user interfaces work. If we think of our complaints regarding that something is not working properly or that there is a feature missing in the user interface we are using—if we would be able to fix it, the fix would be initiated from the user interface perspective. I chose to think about the user interface early during development of `ofxPiMapper`. The User Interface section in this thesis comes after the Software Requirements section because the user interface can't be build without a more general and abstract specification that in this case is a list of requirements for the software to be made.

In this section I am going to discuss the user interface of `ofxPiMapper`. User interface is the

main part of a software that provides users with the possibility to adjust it for their specific needs [42]. Projection mapping software is something that lets adjust images to physical surfaces, but by what means, if the limits of screen and projector positioning are reached? It is possible to build software that does not have a user interface—it would just do one thing according to a hard-coded algorithm, for example add a static transformation or effects layer to the existing image that has to be projected.

There are different things that can be perceived as user interface for a software. For a software library that would be an API (Application Programming Interface)—it would allow another software developer to use her programming skills to create new software. In my case, apart from the fact that `ofxPiMapper` is supposed to be an add-on that has a specific API that matches the openFrameworks add-on infrastructure, I wanted to create a software with a graphical user interface mainly and only then think about other ways of interfacing with it. There are four kinds of user interfaces that I will discuss in this chapter.

1. Graphical user interface.
2. Keyboard interface.
3. Open Sound Control (OSC) API.
4. OpenFrameworks add-on API.

5.2.1 Graphical User Interface

I will start with and mainly focus on the graphical user interface as I considered it to be the main user interface for the `ofxPiMapper` software. I also want the other types of user interfaces to depend on the features that I implement on the graphical user interface level. Therefore before I have decided that the graphical user interface has reached a stable state, I do not want to spend time and effort for developing other kinds of user interfaces.

The Graphical User Interface enables the user to accomplish tasks that require a pointing device input. Selecting projection mapping surfaces, transforming them, choosing the right kinds of sources and adjusting which parts of the sources are visible on the projection mapping surfaces can be done through the Graphical User Interface of `ofxPiMapper`. The concept of "direct manipulation" by Ben Shneiderman [64] can be applied here.

As I have used other projection mapping software earlier, some parts of how the graphical user interface of `ofxPiMapper` could look were clear. On the other hand, a graphical user interface design phase was needed in the beginning of the software development process to better understand the structure of the application. In my case it was a very simple sketch (Figure 5.1) where I made a separation between four visual states of the graphical user interface. The four states would then combine layers of layouts of elements that are laid out and displayed in

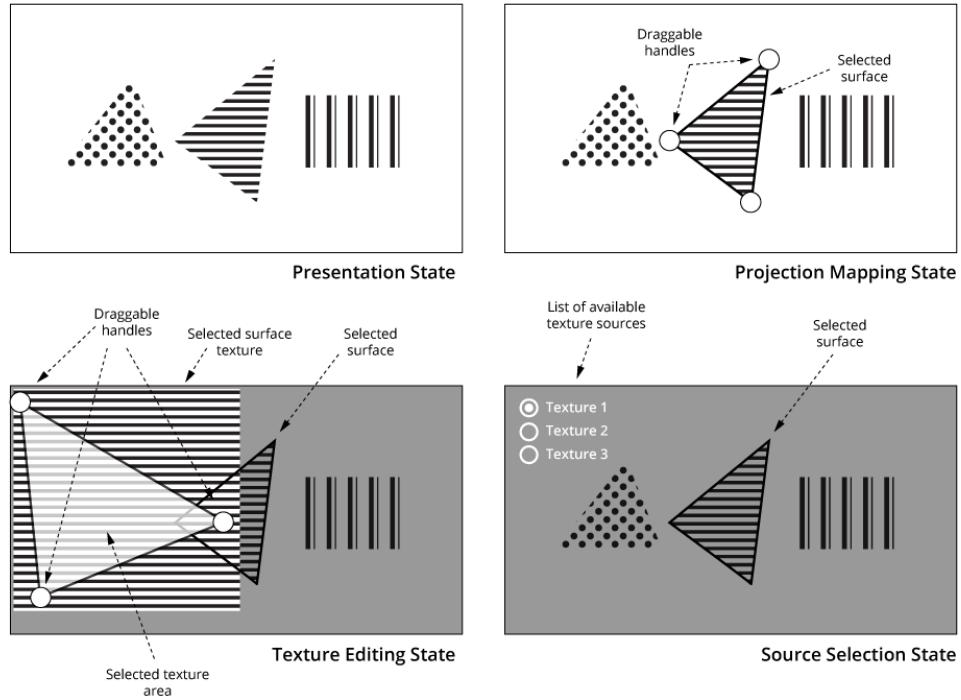


FIGURE 5.1. Initial sketch of ofxPiMapper interface states

different ways according to the graphical user interface state the user would want to use at a given point in time.

As one can see, all of the four states contain repeating elements. Each of the state introduces a new set of graphical user interface elements that allow the user to interact with the software depending on the state it is in.

- The Presentation State consists of plain projection mapping surfaces only.
- The Projection Mapping State already introduces graphical user interface elements that allow the user to see which projection mapping surface is selected and provides draggable handles that can be used to transform the selected surface.
- The Texture Editing State builds on top of the Presentation State by introducing a texture layer which displays full texture below the semi-transparent Presentation State layer and a texture area selector above everything else; the selected projection mapping surface remains selected—it continues to have a white border around it.
- The Source Selection State displays a layer on top of everything else. It consists of selectable radio-buttons—a commonly used interface element.

This simple sketch includes a lot of specific information and solutions to problems that could arise during the actual programming process as for example—what are the building blocks of the graphical user interface, how to logically separate them from one another and how they interrelate. The separation in states of the graphical user interface can also help in the software architecture design phase later. Not a big amount of time is needed to create a sketch like this, but it gives a clear overview of how things should work. Ideally this sketch should be improved in a way it represents the final user interface as it would appear on the user's display when the application is ready. The user interface creation process could also include a phase where the overall style guide of the graphical user interface would be created. This phase would be “hi-tech representation of the design concepts”, would lead to partial or even full functionality and thus could be called hi-fi prototyping [19]. As I wanted to move things forward as fast as possible, I decided to skip this much detail. I was mainly developing the software alone and that made it possible to improvise a bit.

Based on the initial sketch of the graphical user interface of `ofxPiMapper` (Figure 5.1), I am going to discuss the different graphical user interface states in a bit greater detail as in the text before—what is the main functionality of the states and how they are interrelated with each other. Following is a brief summary of the currently available graphical user interface states.

Presentation state

Final rendered output visible only, no visible user interface elements.

Projection Mapping state

Final rendered output with projection mapping surface warping user interface on top of it.

Texture Mapping state

Texture of selected projection mapping surface is visible here with user interface elements that allow to select an area of the texture to be applied to the projection mapping surface.

Source Selection state

Lists available sources for the selected projection mapping surface and allows the user to select one.

The main state of the graphical user interface is the Presentation state. Whenever `ofxPiMapper` software is launched, it automatically enters the Presentation State. In this state only the final output of the projection mapping surfaces is being rendered. No extra user interface elements are visible. The reason for it being the main state is the assumption that the end user would most probably use the software for a project that has to survive multiple shutdown and restart cycles, thus whenever it is turned on, it should be ready for the final show.

To set up the composition, one would first switch to the Projection Mapping state (it is done by using the keyboard interface discussed in the Keyboard Interface section) in order to create,

select and transform projection mapping surfaces. On projection mapping surface creation a default built-in texture is assigned to the surface, the user can then use the draggable handles of the selected surface to transform and distort it.

To adjust what part of the texture is visible on the projection mapping surface, the Texture Mapping graphical user interface state (or mode) has to be used. A state or mode in graphical user interface is a visual representation of a state in the system [51, p. 42]. User input is captured by the user interface mode, interpreted by taking into account the state of the system, and then executed. A projection mapping surface has to be selected in order to be able to adjust the texture of it. The full texture of the selected texture source is being rendered on the screen and a projection mapping surface-dependent area selector allows the user to select a specific area of the texture to be displayed on the projection mapping surface.

Finally, to change the texture, one needs to switch to the Source Selection graphical user interface state to choose a texture for the selected projection mapping surface. The existing texture mapping settings are reused with the newly selected texture.

While developing the `ofxPiMapper` software, I have understood that an iteration of the user interface design phase should follow each software requirements update—especially if a new user interface related feature is needed. Reviewing and improving user interface designs is important also for the software architecture phase later.

There are many levels of complexity that one can add to the user interface design process. The minimum level of complexity that a project should have is wireframes—layouts of user interface elements and a description how they relate to each other. As stated before, this is the approach that I chose for current development flow of `ofxPiMapper`. For the first prototype of a very simple software application it might be enough with a mental image of the interface.

5.2.2 Keyboard Interface

Additional complexity should be added to the graphical user interface if there would be no keyboard interface. To save development time, I decided not to develop another visual user interface layer for on-screen menus. With the help of the Keyboard Interface, the user can switch between the graphical user interface states as well as accomplish different application related tasks as saving the current composition or creating a new projection mapping surface.

The current state of the keyboard interface allows one to accomplish tasks in combination with the graphical user interface. Ideally it should be possible to use either of them separately. New visual user interface building blocks should be created and combined in order to form a full menu-based graphical user interface. The keyboard interface should support switching between unpredictable amount of projection mapping surfaces and other user-defined lists. Additional development time for that is required.

The current key mapping of the keyboard interface is documented on the `ofxPiMapper` GitHub repository (Table 5.1).

Key	Function
i	Show info (help)
t	Add triangle surface
q	Add quad surface
f	Toggle fullscreen
s	Save composition
z	Undo
BACKSPACE	Delete selected surface

TABLE 5.1. Current Key Mapping of ofxPiMapper

5.2.3 OpenFrameworks Add-on API

As `ofxPiMapper` is an openFrameworks add-on—it is a software library that can be used in combination with other openFrameworks projects and add-ons. The example project of the add-on is the main entry point for any user—not depending on whether she is an advanced user or not. The example project can be used for projection mapping projects out of the box, but it is limited for use with image and video sources. By using the `ofxPiMapper` as a library in one's custom openFrameworks project, it is possible to create generative and interactive projection mapping sources for projection mapping.

Currently a reusable C++ base class (`FboSource`) exists that can be extended in order to create custom visual applications that can be used within `ofxPiMapper`. It would be possible to gather data from an online data source and interpret it to create a real-time data visualization. Another example could be data gathering with sensors that are connected to the GPIO (General-purpose input/output) pins of a Raspberry Pi—for instance a PIR (Passive infrared) sensor that would notify the application whenever motion is detected. Kinect or Leap Motion sensors could be used to capture gestures that could be interpreted visually.

The fact that `ofxPiMapper` is an add-on for openFrameworks makes it powerful as it can be integrated with hundreds of other useful add-ons created by the openFrameworks community. This kind of usage of `ofxPiMapper`, however, requires programming skills and that could be one of the main barriers for new users to start using the software.

5.2.4 Open Sound Control API

One of the features that can make a tool like `ofxPiMapper` useful is remote control. There are many ways how to do that and by examining many software tools one can see that new kinds of interfaces and protocols are emerging to allow its users to interact with the tools from other software they are using. One of the most used protocols for remote control used in projection mapping and VJ software today is the OSC [88] protocol. It can be perceived as the successor

of the MIDI or Musical Instrument Digital Interface protocol [73] which has been (and still is) used to control musical instruments and other devices.

I decided to have an OSC API for the `ofxPiMapper` application as that would make it compatible with many other tools currently used in the field. OSC proved itself to be easier to understand than MIDI. Its design also makes it possible to create custom signal (or OSC address) hierarchies that could be useful for controlling complex instruments with hierarchical control parameters.

5.3 Software Architecture

In this section I discuss the software architecture of `ofxPiMapper`. Initially I was not thinking of it as a separate phase or topic that would require much attention, but as the code base of `ofxPiMapper` grew, it became harder to implement new features. I started to search for ways to fix that and stumbled upon the term “software architecture”. Along with the term I found out that software design patterns exist for architecting systems. I found several useful solutions to problems of `ofxPiMapper` code in the book by Erich Gamma et al. [25] I will discuss the solutions found in the following text.

To move forward quickly, it should be possible to implement new features easily, without spending too much time on examining the rest of the source code. Unfortunately this was not true in the initial stages of development. I had to switch from new feature implementation to improving the existing architecture. Which is not bad per se as by using the agile approach—without spending too much time on requirements, a retrospective phase was expected and another software architecture improvement iteration can be considered just normal.

In the following subsections I will discuss the current vision of the software architecture of `ofxPiMapper`. I will start with an overview of the whole system, the main building blocks of it. I will continue with explaining the details about subsystems of the system, their purpose, what were my design choices and why.

5.3.1 Overview

The codebase of the `ofxPiMapper` tool is designed to comply with the openFrameworks add-on structure. This approach has its limitations, but at the same time makes things easier to figure out at the beginning of the project. Fortunately openFrameworks is flexible and after designing the top layer of the system to comply with openFrameworks add-on architecture, one can start building his/her own.

The example application (or the basic `ofxPiMapper` application) starts out as a regular openFrameworks program. A custom class that extends the `ofBaseApp` base class is defined and then allocated in the main function of the application. An instance of the `ofxPiMapper` class is allocated within the created openFrameworks application class defined in `ofApp.cpp` source file

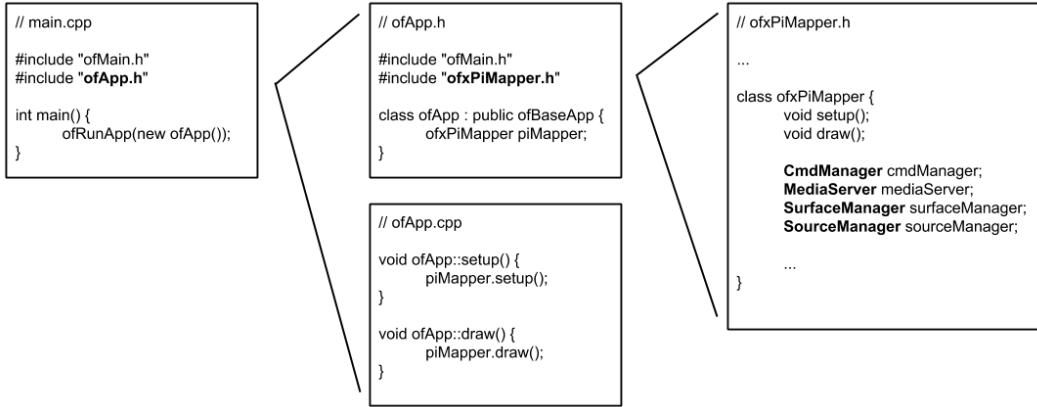


FIGURE 5.2. OfxPiMapper openFrameworks add-on initialization

(Figure 5.2). The created `ofxPiMapper` instance then is configured during the `openFrameworks` application initial `setup()` method call. At the end of the `ofApp` `setup()` method the `setup()` method of the `piMapper` object is called to initialize `ofxPiMapper` with its own `setup()` method. Then the `ofxPiMapper` `draw()` method is called during each `openFrameworks` application `draw()` method call. Everything else happens inside the `ofxPiMapper` object. It basically has to support the `setup()` and `draw()` methods—the rest of the architecture can be customized as needed.

This general structure has been there since the very first version of `ofxPiMapper` as an add-on for `openFrameworks`. The first versions of the add-on example application contained additional code for setting up keyboard shortcuts, now it is not the case anymore. In the add-on initialization diagram (Figure 5.2) all of the subsystems are shown as member variables of the `ofxPiMapper` class, that might not be true in the future versions of the application as some of the subsystems might become singletons [25, p. 127].

5.3.2 Subsystems

In this subsection I discuss the `ofxPiMapper` building blocks and subsystems specifically. It took some time to extract functionality that could be encapsulated in separate components. Some of the subsystems have been there from the beginning of the development, some have been added later and some will be added in the future software development iterations.

In the system overview diagram of `ofxPiMapper` (Figure 5.3) one can see the conceptual separation and hierarchy of the subsystems. On top of all the systems is the user. Underneath there is the user interface layer which consists of the Keyboard Interface, Graphical User Interface and OSC Interface. The user interface layer translates the different types of user input into something the Command Manager can understand. The Command Manager distributes specific commands to the core functionality subsystem layer which consists of

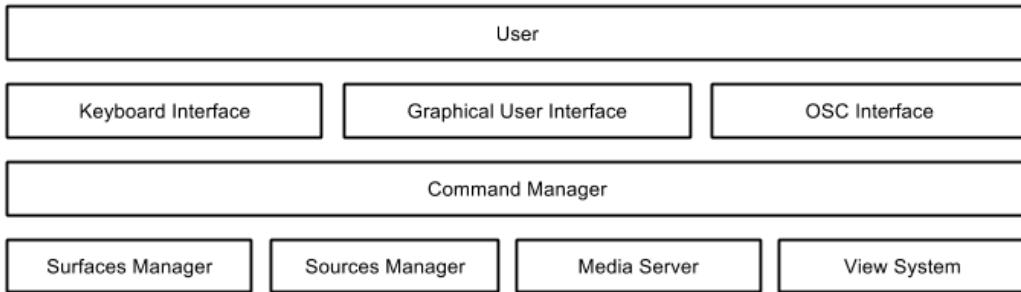


FIGURE 5.3. OfxPiMapper system overview

Surfaces Manager, Sources Manager, Media Server and the View System.

The user interface has been already discussed in the User Interface section (Section 5.2) of this chapter. What I could add here, are some technical details. The Keyboard Interface listens to the keyboard and depending on the set keyboard mapping, creates a specific command to be forwarded to the Command Manager of the application. The Graphical User Interface does the same, but instead of listening to keys on the keyboard, it listens to user initiated events from the View System and maps them to specific commands. The OSC Interface can invoke keyboard and graphical user interface events. In the diagram (Figure 5.3) the OSC interface is on the same level as the keyboard and graphical user interfaces, but in fact it can control both of them.

Command Manager

The Command Manager is responsible for creating and executing commands received from the user and taking care of the undo and redo functionality. The concept of the Command Manager is inspired by the Command design pattern [25, p. 233].

There are two basic types of commands in ofxPiMapper:

Regular commands

They are created and executed, but not stored in the Command Manager undo stack.

Undoable/redoable commands

They are created, executed and stored in the Command Manager undo/redo stack in case user decides to return to the application state before command execution.

Both commands have their own base class—`BaseCmd` and `BaseUndoCmd`. Both of the command classes define interfaces that the Command Manager expects from the commands derived [71, p. 577]. If a command is derived from the `BaseCmd` class, it has to define its own `exec()` method by overriding the `BaseCmd`'s one. If a command is derived from the `BaseUndoCmd` class, it has to define its own `exec()`, and additionally, `undo()` and `redo()` methods.

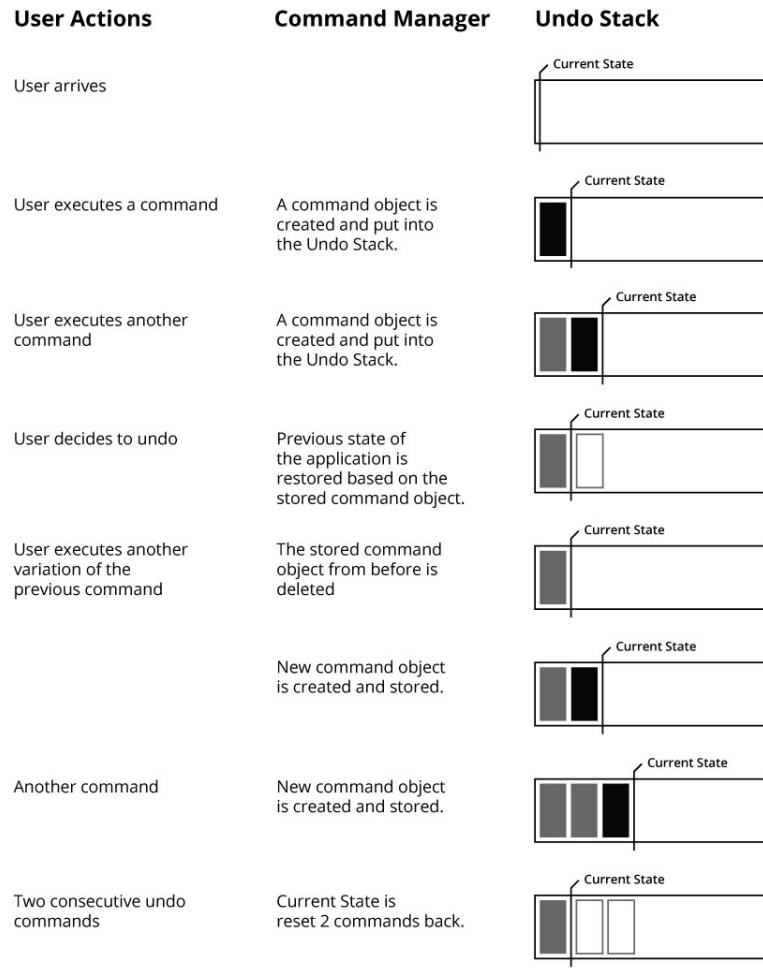


FIGURE 5.4. Undo/redo stack

The `exec()` method executes the actions requested by the user in a way that is acceptable by the system. The `undo()` method rolls back the actions so that the application returns to exactly the same state as it was before the `exec()` method call. If the user changes his/her mind after the `undo()`, he/she can repeat the command by executing the command's `redo()` method (Figure 5.4).

Surfaces Manager

The surfaces manager subsystem is responsible for creating, storing and managing the projection mapping surfaces of the current `ofxPiMapper` application instance. It is based on the concept of the Abstract Factory design pattern [25, p. 87] which means that it is a class

with methods that each can instantiate a specific projection mapping surface class. Each of the methods has different input parameters depending on the type of surface one wants to create.

The Surface Manager keeps track of created projection mapping surfaces and provides methods for manipulating them. A specific “move surface” command might be used to move the currently selected surface by a certain amount of pixels. An instance of the `MvSurfaceCmd` class would then access the Surface Manager to get a pointer of the selected projection mapping surface and call a `moveBy()` method of the selected surface to move it. In another example an instance of the `RmSurfaceCmd` class would access the Surface Manager to delete the selected surface from the projection mapping surface stack, storing the selected projection mapping surface in the undo memory of the command. If the command is being undone, the `undo()` method of it accesses the Surface Manager again and adds the removed surface back to its projection mapping surface stack.

Important part of the Surface Manager are the projection mapping surface classes. They are also supposed to be user interface elements—they have to be selectable, draggable and transformable. The projection mapping surfaces share functionality with conventional user interface elements like buttons. In fact they are buttons, but with shape and texture that is customizable real-time. Surfaces have to detect mouse events and notify the surrounding system whenever user does a mouse click. Projection mapping surfaces also have different states depending on whether they are selected or not and what is the state of the enclosing view.

Sources Manager

The Sources Manager subsystem is responsible for listing the available sources for texturing the projection mapping surfaces. Each projection mapping surface is supposed to use a texture. On projection mapping surface creation a default source with a default texture is assigned to it, but in no case a projection mapping surface can exist without an assigned texture.

The Sources Manager works in tandem with the Media Server subsystem which is based on the Abstract Factory design pattern and is designed in a way that prevents loading a resource twice. Video and image sources depend heavily on the Media Server. The FBO Source, on the other hand, does not use the Media Server at all.

Sources Manager keeps track of what sources are available at application runtime. There are three main source types: the `VideoSource`, `ImageSource` and `FboSource` (or generative source). In the future versions of `ofxPiMapper` additional source types could be added. For example a streaming source that could connect to an online video streaming server and stream projection mapping surface texture over network.

Each of the source classes extend a common `BaseSource` interface that allows the rest of the system handle the different sources in a similar way. Each of the sources provide a pointer to its internal texture. Projection mapping surfaces expect the texture being of the same type [71, p. 135] for all of the different types of sources.

The texture can be perceived as canvas that is painted in ways that are specific to each type of the source. For a video source it will be cleared and repainted on each consecutive video frame. A generative source allows the advanced user to decide in what way and how often the internal source texture is changed.

Media Server

The Media Server subsystem is responsible for keeping track of available images and videos as well as loading and unloading them during application runtime. To save memory resources Media Server reuses already loaded media resources. It also takes care of monitoring the image and video source directories for changes. Whenever a new image or video becomes available, it is able to notify the rest of the application.

There is just one instance of the `MediaServer` class per `ofxPiMapper` application and it uses the Singleton design pattern [25, p. 127]. This allows to be sure that no image and video will be loaded twice and no memory will be wasted even if there are multiple `ofxPiMapper` class instances allocated. This is especially important for a memory limited platform like the Raspberry Pi.

View System

As discussed before in the Graphical User Interface subsection (5.2.1), there are four user interface modes in a `ofxPiMapper` application and there might be more in the future. Each of the modes reuse same datasets, display them in different ways, in combination with additional graphical user interface elements that allow manipulation of the underlying data. The View System is based on the State design pattern [25, p. 305] and it allows to keep each of the user interface modes as a separate state in a separate class. By using the State design pattern, the View System design remains open for improvements in the future.

At the beginning of the project I was trying to separate the graphical user interface from all other visual elements in the application. I was using a set of simple custom graphical user elements for the projection mapping surface user interface and the `ofxGui` graphical user interface openFrameworks add-on for the source selection interface. It proved itself hard to adapt to the specific way I intended it to be used—as lists of radio buttons. I tried to find other openFrameworks add-ons of that kind. I found `ofxUI`, but after trying it out I realized that it consumes too much processor and memory resources and that is what I want to save on the main target platform of this project (the Raspberry Pi).

In the future versions of `ofxPiMapper` the view system will be redesigned. This will allow to not depend on the `ofxGui` add-on dependency and provide better extendability in connection with the evolving `ofxPiMapper` architecture.

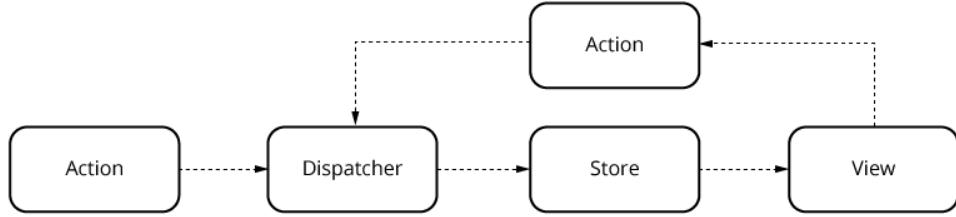


FIGURE 5.5. Flux architecture event flow

5.3.3 Putting It All Together

The `ofxPiMapper` architecture described in this section can be called an unidirectional event flow architecture. Actions from within and outside the system are intercepted by the user interface layer, then passed to the command layer and finally distributed to the separate, but interrelated storage and view components of the system. Some similarities can be drawn with the Flux architecture promoted by Facebook [21]. Flux application consists of three major parts: the dispatcher, the stores and the controller-views. Actions are dispatched through the dispatcher. Stores registered to certain action types are being notified when a specific action happens. The stores emit change events that are handled by the controller-views—they update themselves with fresh data from the stores and re-render. User interface events (when user clicks a specific element in the view, hits a key on the keyboard) intercepted by the controller-views are transformed into actions and sent to the dispatcher again (Figure 5.5). Remotely invoked actions (such as commands sent by a remote control) are intercepted by the dispatcher as well.

The data flow in `ofxPiMapper` is similar in a way that the flow of actions/events is kept unidirectional. Whenever a new event happens in the system, it is handled by a specific (user interface) event handler where it is decided what command to create. The command is then passed to the command manager which executes it, affecting one or more of the underlying subsystems (Figure 5.6).

The view system depends on changes in the media server, source manager and surface manager. This part of the system forms an architectural branch which could seem similar to the store and controller-view relationship in the Flux architecture (Figure 5.7). The view system has to be notified whenever there is a change in the media server, source manager or surface manager that can or should affect the view system.

This approach is clear compared to the MVC (Model-view-controller) architecture where the controller is in-between the model and the view—model being the part storing and/or manipulating application data, view being the user interface and controller the business logic

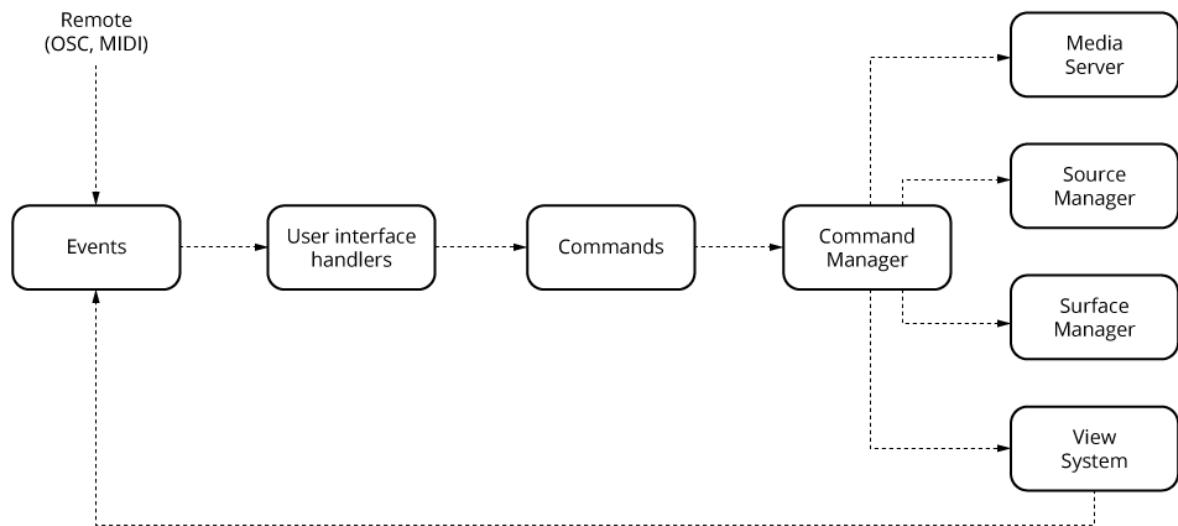


FIGURE 5.6. ofxPiMapper event/data flow

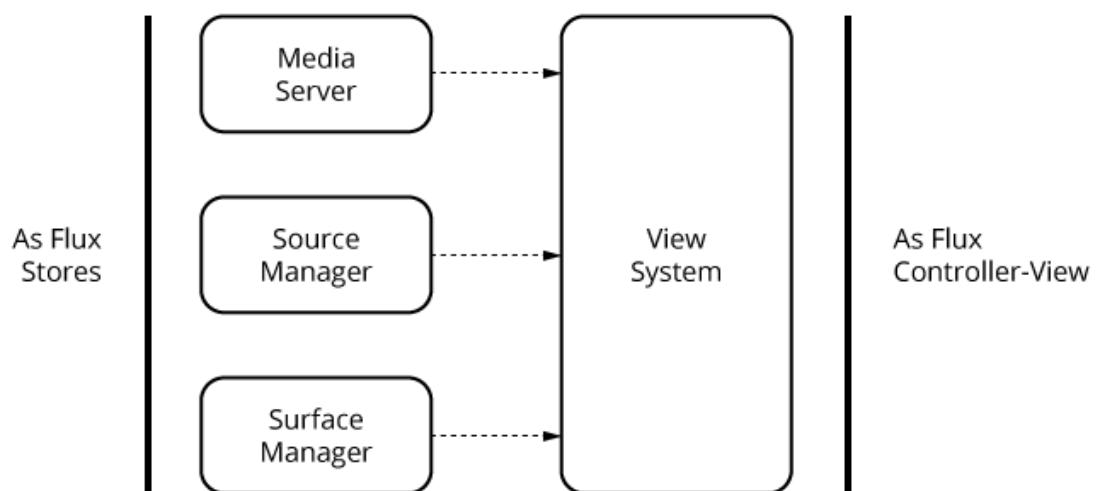


FIGURE 5.7. ofxPiMapper stores and view analogy with the Flux architecture

(Figure 5.8). In MVC controller receives input from model and the view. It also updates both of them. The data, events and actions are running back and forth through the controller. In short—controller has to be aware of the view and the model. In Flux architecture each of

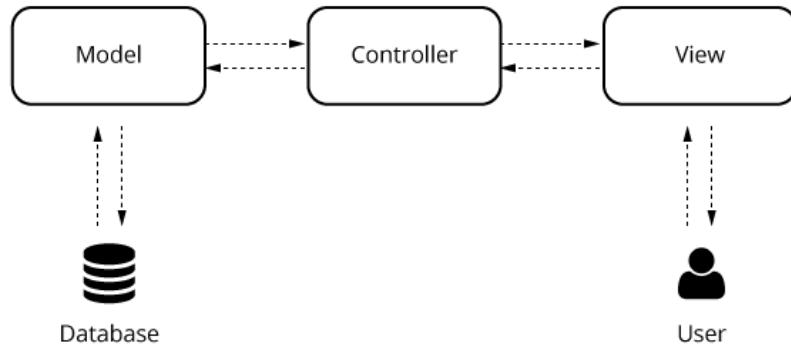


FIGURE 5.8. MVC Architecture

the components have to know only about one kind of participant. For Flux stores it is the dispatcher. For controller-views it is the stores.

Another reason that makes MVC complicated is the many flavours of it. Some of them come implemented as frameworks, but that does not necessarily imply their ease of use.

I did not strive towards reusing existing architectural patterns, but research proved me that unidirectional data flow approach could be easier to maintain. I would probably make efforts towards using the Flux architecture as a starting point if I would need to start the project over again.

HARDWARE-SPECIFIC TESTING

I carried out various tests before and during development of `ofxPiMapper` to understand whether the features that have to be implemented are actually handleable by the Raspberry Pi pocket-size computer. The fact that many people and web resources claimed that Raspberry Pi is able to compile and run openFrameworks applications does not necessarily mean that the specific features work as I imagine them to work in the `ofxPiMapper` context. This chapter discusses the various tests that took place, problems that I faced, and how they were solved.

For each test I made a new openFrameworks application prototype. I tried to implement only the things that are relevant to the test. If things did not work out the first time I tried to find workarounds till I arrived to a working solution. From there I could decide whether the workarounds do not conflict with expected performance and code clarity requirements.

6.1 OpenFrameworks on the Raspberry Pi Test

At the beginning of this project the fact that openFrameworks is able to run on the Raspberry Pi was great news in the creative coding community. During the Resonate 2013 festival in Belgrade, Serbia, there was a workshop called “openFrameworks on the Raspberry Pi” by Andreas Müller and Jason van Cleave [53]. Even the advanced 3D graphics examples seemed to run without problem. After the workshop the Creative Applications Network released an article with a tutorial [12], written by the same people who ran the workshop, with instructions on how to set up openFrameworks on the Raspberry Pi.

At the time various on-line instructions on how to prepare the Raspberry Pi for being able to compile and run openFrameworks applications started to appear, and all that served as

encouraging material to start the development of `ofxPiMapper` as an add-on for openFrameworks.

First test was to get the openFrameworks compile and run on the Raspberry Pi. As there was information on the internet [45], the process went relatively smoothly—I was able to compile openFrameworks on the Raspbian Linux operating system as well as the example applications. It was clear that it is worth continuing the project.

6.2 Texture Mapping Test

After compiling and running the example applications available with the openFrameworks source, it was necessary to understand whether the implementation of `ofMesh` class works as expected. The `ofMesh` class in openFrameworks is used for constructing and manipulating 3D meshes and it supports texturing and texture coordinate mapping. This is the main functional feature that I needed for the `ofxPiMapper` project. If `ofMesh` would not work, I would need to develop something similar from scratch.

The prototype consisted of a triangular mesh and a bitmap image to be used as a texture for the triangle. The vertices of the mesh were supposed to be clickable and dragable. Texture was expected to scale and deform along with the triangle whenever a mesh vertex was moved. At first it seemed that the Raspberry Pi version of openFrameworks has problems with that as the assigned textures did not appear on the meshes. I discovered that the Raspberry Pi openFrameworks `ofMesh` texturing feature works only if the use of normalized texture coordinates is enabled by using the `ofEnableNormalizedTexCoords()` method.

The prototype proved that there are no problems with the 3D graphics tool implementation in openFrameworks and they work well on the Raspberry Pi, providing a good basic foundation for the `ofxPiMapper` projection mapping tool.

6.3 Remote Control Test

Next prototype was made in order to test the remote control capabilities of openFrameworks and how they could benefit the development of `ofxPiMapper`. `OfxOsc` (Open Sound Control protocol add-on for openFrameworks) was used for listening to commands sent over the network. A web server was set up on the Raspberry Pi running Raspbian operating system. User could access the interface of the tool by using a web browser. He/she then would enter the IP address of the Raspberry Pi in the browser's address field. PHP (PHP: Hypertext Preprocessor) module was used on the web server to dynamically forward data from web based user interface to the running prototype application. Open Sound Control for PHP [46] software library was used to send OSC messages to a specific port on the same machine.

The interface featured a quadrangle with vertices that could be clicked (or touched in case of a touch device) and dragged. OSC messages containing the vertex coordinates were sent

to the projection mapping application whenever a vertex moved. The projection mapping application interpreted sent OSC messages and updated its views.

This prototype proved two things:

1. It is possible to use OSC as a general purpose remote control protocol.
2. It is possible to create a web-based remote control interface that would work on any networked device that has a web browser.

A separate web server application was used in the case of this prototype for serving the web interface. There are openFrameworks add-ons that allow to implement web server functionality directly in an openFrameworks application. This prototype was developed during Multimedia Authoring course at the Helsinki Media Lab, December 2013, lecturer Nuno Correia.

6.4 Generative Texture Test

One of the main goals of `ofxPiMapper` was to allow one to use it with interactive and generative imagery. For that to work, the `ofFbo` openFrameworks class was used. It allows to store the generated image into a frame buffer for later use. The `ofTexture` instance (data structure which actually contains and allows to draw the visual data) of the `ofFbo` object can then be used as texture of the projection mapping surfaces laid out in the projection mapping view.

For this test I wrote a simple algorithm for twenty white rectangles on a black background. At application start height and movement speed along the y axis would be randomly calculated. Then on each frame (with a frame rate of 30 frames per second) the y position of each of the rectangles would be increased by its specific speed value (Figure 6.1). The generated animation was rendered on a 500 pixels wide and 500 pixels high `ofFbo` off-screen canvas. Then the `ofTexture` part of it was used as the texture for the projection mapping surface.

This test proved that there are no problems in using `ofFbo` objects as generative texture containers for projection mapping surfaces. The only thing that I noticed is that the `ofFbo` class has to be allocated on the heap or free store (by using the `new` keyword) rather than stack [71, p. 277], order to be able to access and draw data of the contained `ofTexture` object.

6.5 Video Texture Test

One of the most indistinct features was the support of video texture. Fortunately I found `ofxOMXPlayer` [81]—openFrameworks add-on for hardware-decoding H.264 encoded video on the Raspberry Pi, developed by Jason van Cleave, that makes use of Raspberry Pi’s Broadcom VideoCore IV video decoding chip. This helped to develop the `VideoSource` class for the `ofxPiMapper`.

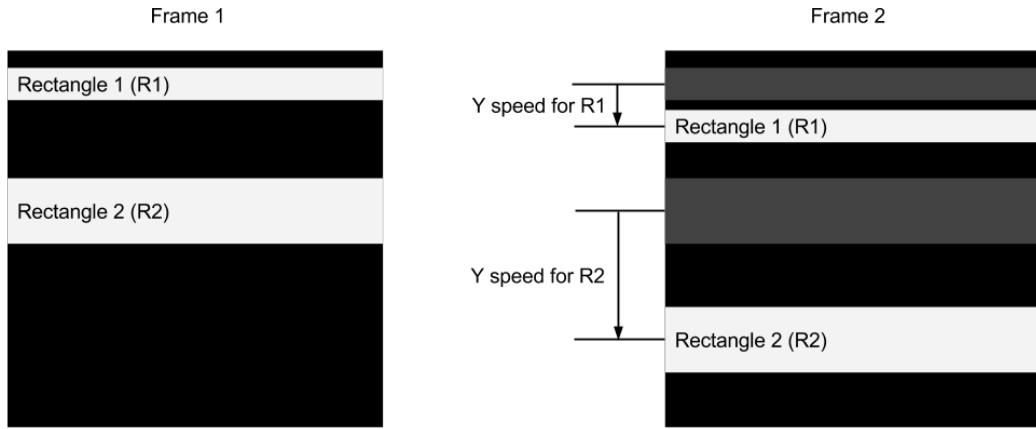


FIGURE 6.1. Generative texture test algorithm

The `ofxOMXPlayer` add-on is capable of rendering the decoded video into an `ofTexture` object. It is more processor-intensive than plain decoding and direct rendering into the Raspberry Pi framebuffer (memory unit of the graphics chip where image data is being collected in a pixel-by-pixel manner before displaying it on a screen), thus the rendering width and height in pixels of the video have to be smaller in order to avoid lags (unwanted decrease of the application frame rate). Sound of the video can be disabled in order to increase the decoding performance.

One of the problems noticed during this test was rendering of grey color instead of the actual texture if the video dimensions exceeded a certain limit. After some experimentation I discovered that it can be solved by increasing the memory available to the GPU of the Raspberry Pi. There are two ways to do it:

1. By using the `raspi-config` command line tool in Raspbian.
2. By manually editing the `gpu_mem` variable in the `/boot/config.txt` configuration file.

Raspberry Pi has a CPU and GPU integrated in a single chip and they also share the same memory pool. It depends on which version of the Raspberry Pi one is using, but for running `ofxPiMapper` one should allocate no less than 128MB of memory for the GPU (which is doable for all versions of the Raspberry Pi). The more imagery one wants to render, the more memory should be allocated for the GPU.

6.6 Summary

The prototypes built for the tests proved that it is possible to develop a projection mapping tool that would run on the Raspberry Pi. The results were positive, each of the prototypes could be improved and transformed into specific components of the `ofxPiMapper` software. The process also provided motivation as it was possible to get feedback from people by showing them something that actually works.

The tests also helped to build up experience that will be useful in building the full version of the software. The Generative Texture Test (Section 6.4) was especially useful as it was the source of the decision for making extensive use of pointers and the heap memory throughout `ofxPiMapper` source code. The success of the Video Texture Test (Section 6.5) showed that the software could reach a level where it could compete with the already available tools.

Because of the initial tests I was also able to get enough feedback to avoid extensive user testing phase at the end of the thesis project. It will be done at some point in the future, but the current status of the software asks for more input regarding source code structure and development. User testing will be planned and done when a stable software architecture state and larger user base will be reached.

RELEASING OPEN SOURCE

There are many ways of releasing software open-source. One of the old ways is putting the source on your own server and then advertise it. The critical part of availability of this kind of project is the advertisement one, the project gets lost as soon you don't spend enough time talking about it. I wanted to decrease the amount of time spent on advertising. My general idea was, and still is, to make the software as good as possible and if users like it and actually use it, it would be the best advertising a software can have.

I also like the idea of opening the source code of an application to its users. It can be especially useful for applications that are developed by a single person, he/she might not have enough time to respond to bug fix requests quickly enough. An advanced user might choose another software where the bug is fixed already or, if the source is available, fix it on his/her own. Ideally the fix would then be merged with the base source code and made available to other users.

Releasing application source code publicly can improve code quality as with each next user and contributor there is a new pair of eyes that can spot problems [52]. It also can be a good way to find new programmer friends or future colleagues.

7.1 Releasing on GitHub

I already explained the reasons of choosing Git and GitHub as tools for managing the source code of `ofxPiMapper` in the Git and GitHub section 2.6. Ideally one should create a Git repository for a project at the very beginning of it and improve on its history with each new feature added. For the first working version of a software it can be enough with having a single commit

(manually added source code snapshot to Git history), but it is recommended [7] to have more than that, for example one or more entries for the source code itself, one for adding the license, one for the readme.

I decided to release the source code on GitHub once I got the very basic functionality working. It was possible to compile and run the application. One had to add custom code to create additional triangle surfaces, load and assign sources. It was possible to use mouse to transform the triangle surfaces by clicking and dragging on their corners. As I have used Git from the beginning of the project, there was no problem uploading the source code to GitHub. I already had an account there, I had to add a new repository entry, add the repository as a remote to my local `ofxPiMapper` Git repository and push the master branch to the remote [7, p. 35].

There are openFrameworks add-on release conventions that require having certain structure for your add-on project. It exists because of the openFrameworks add-on catalogue (`addons.openframeworks.cc`) which is compiled automatically from projects on GitHub that match a pattern. The source code has to be structured into `src`, `lib`, `bin` and `example` directories. You can choose, but it should have a license file, a “`README`” file, which contains basic information about the add-on and how to use it, and a image that can be used by the openFrameworks add-on catalogue.

7.2 Documentation

As for any decent open-source project, documentation is essential. You want people to understand what the code is about and you don’t want to steal their precious time for exploring the maze you made if you want them to contribute to the project. In order to attract contributors, you have to lower the entry barrier for them in terms of the complexity of the structure of the project and good documentation. Good documentation means that it is easy to find, it contains all necessary information and that it is also easy to navigate.

The main documentation entry point of almost every GitHub project is the “`README`” file. It usually contains basic information about the project, its authors, how to use the source code and information for those willing to contribute. GitHub also has a per-repository wiki that can be updated by contributors of the project. When the project grows big, it is common that the “`README`” file is not enough for all the documentation that a project requires.

In the case of `ofxPiMapper`, the “`README`” file was and still is enough for describing information relevant to the users as well as the potential contributors. Still, with each next version of the software, something new is added to the documentation and soon it will be necessary to move parts of the “`README`” somewhere else. My plan is to leave only the documentation that is relevant for the new user of the software in the “`README`”, move the advanced user and contributor related information to the wiki and create a “`CHANGELOG`” file that lists most

important changes per version.

7.2.1 Usage Instructions

The most important thing of an openFrameworks add-on usually is information about how to use it. In this case it is important to tell the users what `ofxPiMapper` is all about, list dependencies and provide with an example project. `OfxPiMapper` example project is made as a stand-alone application that can be used for one's projection mapping project right out of the box. It is important to list the dependencies before pointing to the example project as it won't run if the user won't have the libraries and openFrameworks add-ons required.

`OfxPiMapper` has been made by keeping in mind that it has to be easy to use. Thus the documentation regarding its usage must be as concise as possible. Main graphical user interface states are described along with keyboard shortcuts that provide access to them. Advanced users are provided with code snippets that can help them use `ofxPiMapper` with their custom code.

One of the main requirements to use the software is for the user to know how to install openFrameworks, how to compile and run its applications. The user should also know how to install and use add-ons, although the example applications of the addons are just another openFrameworks applications that have to be compiled and run in the same way as every other openFrameworks project.

7.2.2 Development Guidelines

As the main readme file contains basic information about the add-on and how to use it, there is not much space for information regarding `ofxPiMapper` development. For an experienced developer it is clear that the source code can be a great up to date documentation itself, but sometimes the project becomes more approachable if it has a brief introductory essay with a couple of illustrations about the vision and structure of the project by the author of it. From my experience I can say that I feel welcome to contribute to a project (more often than I actually contribute) as this much effort proves that there is someone who actually cares about the project.

A reasonable place for a developer guide of a project hosted on GitHub is the repository wiki. It can be accessed by navigating to the repository page and then clicking on the Wiki menu item. The GitHub repository wiki is a Git repository itself and can be updated the same way a project containing source code is. The wiki consists of plain text files that usually are formatted using the Markdown [83] syntax. A Git commit can be made with each addition or modification of a file, this makes it possible to follow the history of the wiki development easily.

The developer guidelines of `ofxPiMapper` includes a "welcome" text for potential contributors. It also provides with a brief overview of the system, similar to the description in the

Software Architecture section (Section 2.6). Detailed descriptions of specific topics can be found in subsections of the wiki.

7.2.3 Programming Conventions

Programming conventions are essential as potential contributors should spend as short time period as possible understanding other developer code [37, p. 66]. It is easier to read a familiar handwriting than an alien one. Programming conventions should not be too detailed as following them should be generally easier than solving problems in the source code itself. It makes sense having more complicated and more specific guidelines for a bigger project, but for a project like the `ofxPiMapper` simple ones could suffice.

As `ofxPiMapper` is an add-on, I decided to use the same programming conventions that are used in its base platform—`openFrameworks` [45]. The `openFrameworks` programming conventions are based on Qt code style, but modified to suit `openFrameworks` better [49]. It is stated in the conventions that it is generally advised to keep the source code as simple as possible. Instead of writing clever code, one should stick to writing readable code. It can be a great personal achievement to be able to solve problem with single compact but unreadable line of code, but it does only harm if someone else has to understand it to improve or debug.

The `ofxPiMapper` project was started without keeping any programming conventions in mind and it started to cause problems when first contributors started to appear. I was using tabs for indentation, contributors used spaces. Along the way the code became harder to read because of bad indentation and additional time had to be spent in order to fix that atop deciding to what coding conventions to stick to.

7.2.4 Collaboration

Releasing the code open source seems easy before one has to think about how to deal with improvements from project contributors. For example, before Git Linux developers worked with patches [7, p. 138] that were sent to one of the managers of the project who then applied the patch to the existing source code and was responsible for releasing a new version of the software afterwards. The authors of Git and GitHub know that, and this is why there are tools that can be used to make the management of a project easier.

GitHub introduced a feature that allows the one who owns a GitHub account to clone an existing project and copy it to his/her GitHub account. The process is called “forking” [7, p. 113] and it allows one to safely work with the project source code and its Git history as if it was his/her own. This way one can create a new feature or fix a bug locally, push it to the his/her own forked version of the project on GitHub and then create a “pull request” to the original project with the commits relevant to the latest changes. The developer of the original project then can make sure if the code works, and if it does, merge it easily with the existing source code.

Another approach is to add collaborators (other existing GitHub users) to the project GitHub repository. The collaborators then do not need to create pull requests and wait for them to be accepted—they can merge changes with the original code base whenever they feel the need to do so. It has good and bad aspects.

Good

- Final rendered output visible only, no visible user interface elements.
- One does not need to spend time on reviewing each pull request.
- The development of the code base can happen faster this way as it is not dependent on a centralised decision maker who might become sick or out of time.

Bad

- Code is not being double-checked and can lead to decreasing quality of the code.
- Collaborators can harm the code based on personal conflicts or human error.

For a project where collaborators do not share a room or office the “pull request” approach seems a generally wiser choice. Nevertheless project development can have slowdowns because of the central developer not having time to review pull requests, it leads to better code quality as there will be always at least two pairs of eyes going through the code before releasing it.

GitHub comes with an integrated issue tracking tool. It allows to create issues - notes about changes that have to be made, new feature requests or bugs in the software. Each issue can be tagged with a label, e.g. bug, question, enhancement. There are default labels there, but one can create custom ones if the specific project needs it. Issues can be assigned to collaborators and grouped in milestones. Milestones can be seen as collections of issues and once all issues of a milestone are solved, the milestone can be considered to be reached. Milestones are usually future versions of the software, e.g. if the software is at version 1.1, the next milestone can be version 1.1.1 or 1.2 depending on the type and amount of changes to be done.

OfxPiMapper benefits from the use of semantic versioning [62] and the correct implementation of the specification is being polished along the way. The numbering scheme consists of MAJOR, MINOR and a PATCH number. MAJOR version changes once there are incompatible changes to the API. MINOR version number changes once new, backwards-compatible functionality is added. PATCH version number changes once bugs are fixed in a backwards-compatible manner.

For every collaborator it is important to master Git to a level one understands branching. “Commiting” often and being able to clearly define what next commit is going to be is important

for Git-centered development. The use of a tool like Git does not say that human contact is obsolete—if you are in trouble, ask for help as soon as possible unless you like and are able to solve your problems on your own, therefore you are sure that you won't break the project, which is unlikely when using Git.

OfxPiMapper advertises the Git branching model introduced by Vincent Diessen [43]. For each new feature, bug-fix or refactoring section one should make a separate branch. Therefore it is important to keep your development branch and it should be separate from the main or master branch. Once a feature is added, it is supposed to be merged into the development branch.

Once a group of features and bugfixes are done, a new version branch can be created out of current development branch to adjust documentation, bump version in source files, etc. The new release branch then is merged with master and also development branch. Master branch is being tagged with the new version number.

CONCLUSIONS

The making of `ofxPiMapper` was an interesting and valuable process as I had to learn many new concepts throughout the way. It made me appreciate other open source software much more, especially the fact that there are people behind the projects who really care and spend their free time in long-term to keep the development of the software alive. There are ethical issues in open-source software development—it is about not letting your users down. They are waiting for new features and with each next user your responsibility towards all of them grows.

I think that I chose the right way to release `ofxPiMapper`. I published the source code on GitHub as soon I got the very basic features working. I then added feature by feature and as the project is online for more than a year now (since 9 May 2014) [57], it is possible to evaluate how useful the project is to users by looking at the GitHub repository statistics. As of now (26 Sep 2015) the `ofxPiMapper` GitHub repository has 70 stars and it is being viewed by more than 10 unique visitors a day which might seem not much for a product website, but for a source code repository of a specific niche project like `ofxPiMapper` it is successful statistics.

8.1 Evaluation of the Initially Set Goals

Regarding the software development goals of the thesis, to develop new projection mapping software for low-end computers with features that have been accepted by the projection mapping community through existing software and new, specific, ones. The goal has been reached. The software is available online, it is being used, proving that it has features others find useful. It also has features other projection mapping software does not. One can use `ofxPiMapper` as a part of his/her software as a library, extend its functionality by editing source code and, most importantly, run it on a Raspberry Pi.

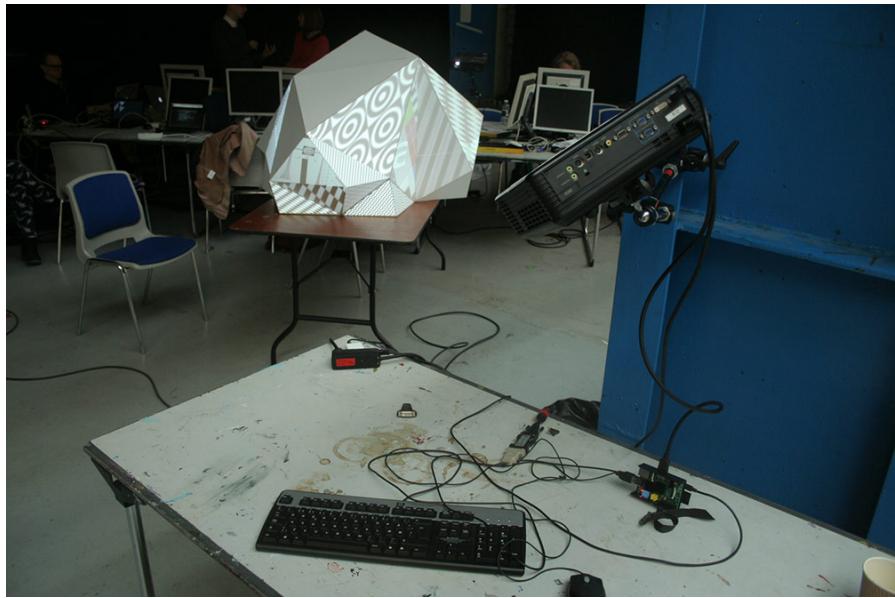


FIGURE 8.1. Projection Mapping with the Raspberry Pi workshop at the CLICK Festival 2013

Hardware-wise Raspberry Pi was chosen as the primary ARM based platform to ofxPiMapper develop against. The primary goal was being able to compile and run the software on the pocket-size computers. I was using the Raspberry Pi 1 model B for the initial tests and they proved positive. I was also able to compile and run ofxPiMapper on Raspberry Pi 1 model A+ which was especially interesting as the model A+ is almost half the size of model B and B+ therefore having half of the RAM of the latter. fxPiMapper has also been tested and is compatible with the Raspberry Pi 2 model B.

Open-sourcing the project was a good idea as it brought me closer to the projection mapping community. Every question or suggestion from a user increased my motivation to continue working on the project. The addition of ofxPiMapper to one of the biggest projection mapping related resources on the internet (projection-mapping.com) was also a great motivation towards finding more time and energy to continue developing the software. Using Git and GitHub helped me to keep the project organized and available to other potential developers. I also learned that keeping direct contact with the users makes new friends and increases the value of the software. Software is made by people and the way we mostly perceive applications today makes us forget that it is actually made by human beings. When the user can talk with the developer directly, the software becomes more personal as the human being behind it can create emotions and experiences the software will never be able to create alone.

I did workshops during the development of ofxPiMapper. The first workshop took place during the CLCIK Festival 2013 [9] in Helsingør, Denmark (Figure 8.1). Along with introducing participants to projection mapping with the Raspberry Pi and ofxPiMapper, I also taught

openFrameworks and a little bit of Git. It was a valuable experience as I saw the users getting very excited about being able to create custom generative textures with openFrameworks to be used for projection mapping with `ofxPiMapper`.

8.2 Future Research and Development

The software development done as a part of my master's thesis solved problems related to code clarity. A clear separation in subsystems and components was done, specific design patterns were used to make the source code easier to understand for potential contributors. One of the future goals is to improve the architecture even more, the ultimate goal being able to improve any subsystem of `ofxPiMapper` without the need to understand how other parts of the code work. For example one should be able to add a new type of projection mapping surface by copying and modifying an existing projection mapping surface type class and just that.

`ofxPiMapper` should have more of the best features from existing projection mapping software while keeping it simple and easy to use. For example implementing bezier warping for projection mapping surfaces could be a challenge. Custom mesh import would be also something to consider. Video sources that one could synchronize among many devices is another feature that could prove itself useful in different contexts.

Improving user interface and user experience is another important goal for the future. Including comprehensive usability and user experience research was beyond the scope of my master's thesis. The GitHub repository statistics and users trying to communicate with me directly prove me that I am heading in the right direction. There are still user interface related bugs that I see myself.

Additionally to `ofxPiMapper` running on the Raspberry Pi I would like to try to use it on other ARM based platforms like the Banana Pi or Cubietruck. Mac OS X was used as the main operating system for development and it is one of the environments `ofxPiMapper` could target in the future along with Windows. There are operating system specific features to be added as the Syphon source for Mac OS X computers and Spout source for Windows. Syphon and Spout allow to share video real-time among running applications. That would allow to create or mix video in one software and use `ofxPiMapper` on the same computer to projection map it.

Workshops and small festivals could be organized to showcase interesting projection mapping projects made with the software. Workshops take place for basic and advanced users. Basic user workshops could include installing `ofxPiMapper` on a Raspberry Pi or using a previously installed one to projection map a simple three-dimensional object. Advanced workshops would require some programming skills and one could learn to use `ofxPiMapper` for custom generative art projects or how to add new features and contribute them to the main source code.

BIBLIOGRAPHY

- [1] G. AŽMAN AND J. KALB, *C++ Today: The Beast Is Back*, O'Reilly, 2015.
- [2] BEAGLEBOARD.ORG, *Beagleboard.org - community supported open hardware computers for making*, 2015.
[online]. Available at: <<http://beagleboard.org>> [Accessed 6 May 2015].
- [3] A. BELL AND H. NGUYEN, *Cinder | the library for professional-quality creative coding in c++*, 2010.
[online]. Available at: <<http://libcinder.org>> [Accessed 15 Mar 2015].
- [4] O. BIMBER AND R. RASKAR, *Spatial augmented reality*, A K Peters, 2005.
- [5] J. BOCKLAGE-RYANNE AND J. THELIN, *Qt5 cdaques — 5 cdaques book v2015-03*, 2015.
[online]. Available at: <<http://qmlbook.github.io>> [Accessed 25 Sep 2015].
- [6] T. BUTTERWORTH AND A. MARINI, *Syphon*, 2010.
[online]. Available at: <<http://syphon.v002.info>> [Accessed 2 Sep 2015].
- [7] S. CHACON, *Pro Git*, Apress, 2009.
- [8] D. CHISNALL, *Understanding arm architectures*, 2010.
[online]. Available at: <<http://www.informit.com/articles/article.aspx?p=1620207>> [Accessed 2 Sep 2015].
- [9] CLICKFESTIVAL.DK, *Click festival*, 2015.
[online]. Available at: <<http://www.clickfestival.dk>> [Accessed 29 Sep 2015].
- [10] N. COLLINS, A. MCLEAN, J. ROHRHUBER, AND A. WARD, *Live coding in laptop performance*, Organised Sound, 8 (2003).
- [11] CPPREFERENCE.COM, *cpreference.com*, 2015.
[online]. Available at: <<http://cpreference.com>> [Accessed 24 Sep 2015].
- [12] CREATIVEAPPLICATIONS.NET, *How to use openframeworks on the raspberrypi - tutorial*, 2013.
[online]. Available at: <<http://www.creativeapplications.net/tutorials/how-to-use-openframeworks-on-the-raspberrypi-tutorial>> [Accessed 15 Mar 2015].

BIBLIOGRAPHY

- [13] J. CREMIEUX, *Fulldome content production: A bricoleur's approach*, Master's thesis, Aalto University, 2011.
- [14] DEBIAN.ORG, *Debian—the universal operating system*, 2015.
[online]. Available at: <<http://www.debian.org>> [Accessed 24 Sep 2015].
- [15] DEVELOPER.APPLE.COM, *Introduction to quartz composer user guide*, 2007.
[online]. Available at: <https://developer.apple.com/library/mac/documentation/GraphicsImaging/Conceptual/QuartzComposerUserGuide/qc_intro/qc_intro.html> [Accessed 2 Sep 2015].
- [16] B. DOLL, *10 million repositories*, 2013.
[online]. Available at: <<https://github.com/blog/1724-10-million-repositories>> [Accessed 14 May 2015].
- [17] E. DUBROFSKY, *Homography estimation*, Master's thesis, The University of British Columbia, 2009.
- [18] B. EDELSTEIN, F. WUNSCHEL, A. RECHE, M. BEGHIN, D. GELDREICH, G. ABEGG-GAUTHEY, E. MORZIER, I. KATIN, P. SCHNEIDER, AND C. E. A. BENOIT, *MadMapper*, GarageCUBE and 1024 Architecture, 2010.
- [19] F. N. EGGER, *Lo-fi vs. hi-fi prototyping: how real does the real thing have to be?*, 2000.
- [20] R. ELEKTRONIK, *Raspberry pi pricing*, 2015.
[online]. Available at: <<http://www.reichelt.de>> [Accessed 6 May 2015].
- [21] B. FISHER, *Flux: A Unidirectional Data Flow Architecture for React Apps*, ACM, 2015.
- [22] FORUM.LEMAKER.ORG, *The open source sbcs community*, 2015.
[online]. Available at: <<http://forum.lemaker.org/forum.php>> [Accessed 6 May 2015].
- [23] FORUM.OPENFRAMEWORKS.CC, *Openframeworks forum*, 2015.
[online]. Available at: <<http://forum.openframeworks.cc>> [Accessed 24 Sep 2015].
- [24] B. FRY AND C. REAS, *Processing*, 2001.
[online]. Available at: <<http://processing.org>> [Accessed 6 May 2015].
- [25] E. GAMMA, R. HELM, R. JOHNSON, AND J. VLISSIDES, *Design patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley Professional, 1994.
- [26] GITHUB, *Build software better, together*, 2015.
[online]. Available at: <<https://github.com>> [Accessed 14 May 2015].
- [27] GNU.ORG, *Gcc, the gnu compiler collection*, 2015.
[online]. Available at: <<https://www.gnu.org/software/gcc>> [Accessed 23 Sep 2015].

- [28] F. HARARY, *Graph theory*, Addison-Wesley Pub. Co, 1969.
- [29] J. HARDY AND J. ALEXANDER, *Toolkit support for interactive projected displays*, ACM, 2012, p. Article No. 42.
- [30] ISO.ORG, *Iso/iec 15948:2004 - information technology - computer graphics and image processing - portable network graphics (png): Functional specification*, 2004.
[online]. Available at: <http://www.iso.org/iso/catalogue_detail.htm?csnumber=29581> [Accessed 2 Sep 2015].
- [31] B. JONES, *The illustrated history of projection mapping*, 2013.
[online]. Available at: <<http://projection-mapping.org/the-history-of-projection-mapping>> [Accessed 20 Aug 2015].
- [32] B. JONES, R. SODHI, K. KARSCH, M. CASPERSON, AND C. HENKE, *Projection mapping central*, 2012.
[online]. Available at: <<http://projection-mapping.org>> [Accessed 24 Sep 2015].
- [33] M. KRIGSMAN, *Study: 68 percent of it projects fail* | zdnet, 2009.
[online]. Available at: <<http://www.zdnet.com/article/study-68-percent-of-it-projects-fail>> [Accessed 1 Jul 2015].
- [34] T. LINDHOLM, F. YELLIN, G. BRACHA, AND A. BUCKLEY, *The java® virtual machine specification*, 2013.
[online]. Available at: <<http://docs.oracle.com/javase/specs/jvms/se7/html/index.html>> [Accessed 24 Sep 2015].
- [35] LINUX.COM, *What is linux: An overview of the linux operating system*, 2009.
[online]. Available at: <<https://www.linux.com/learn/new-user-guides/376-linux-is-everywhere-an-overview-of-the-linux-operating-system>> [Accessed 23 Sep 2015].
- [36] D. MANIELLO, *Augmented Reality in Public Spaces: Techniques for Video Mapping*, Le Penseur, 2015.
- [37] S. McCONNELL, *Code Complete*, Microsoft Press, 2 ed., 2004.
- [38] C. METZ, *How github conquered google, microsoft, and everyone else*, 2015.
[online]. Available at: <<http://www.wired.com/2015/03/github-conquered-google-microsoft-everyone-else>> [Accessed 14 May 2015].
- [39] B. MEYER-STABLEY, *Agile! The Good, the Hype and the Ugly*, Springer, 2014.
- [40] S. M. MITCHELL AND C. B. SEAMAN, *A comparison of software cost, duration, and quality for waterfall vs. iterative and incremental development: A systematic review*, IEEE Computer Society, 2009, pp. 511–515.

BIBLIOGRAPHY

- [41] M. NAIMARK, *Displacements*, San Francisco Museum of Modern Art, 1980.
- [42] J. NIELSEN, *Usability engineering*, Academic Press, 1993.
- [43] NVIE.COM, *A successful git branching model*, 2010.
[online]. Available at: <<http://nvie.com/posts/a-successful-git-branching-model>> [Accessed 15 Mar 2015].
- [44] OFXADDONS.COM, *Addons howto*, 2015.
[online]. Available at: <<http://www.ofxaddons.com/pages/howto>> [Accessed 14 May 2015].
- [45] OPENFRAMEWORKS.CC, *openframeworks*, 2015.
[online]. Available at: <<http://openframeworks.cc>> [Accessed 14 May 2015].
- [46] OPENSOUNDCONTROL.ORG, *Open sound control for php*, 2008.
[online]. Available at: <<http://opensoundcontrol.org/implementation/open-sound-control-php>> [Accessed 15 Mar 2015].
- [47] A. PHAM, *E3: Microsoft shows off gesture control technology for xbox 360*, 2009.
[online]. Available at: <<http://latimesblogs.latimes.com/technology/2009/06/microsofte3.html>> [Accessed 25 Sep 2015].
- [48] R. PI, *Raspberry pi - teach, learn, and make with raspberry pi*, 2012.
[online]. Available at: <<https://www.raspberrypi.org>> [Accessed 6 May 2015].
- [49] QT-PROJECT.ORG, *Qt_coding_style*, 2015.
[online]. Available at: <http://qt-project.org/wiki/Qt_Coding_Style> [Accessed 15 Mar 2015].
- [50] R. RASKAR, G. WELCH, M. CUTTS, A. LAKE, L. STESIN, AND H. FUCHS, *The office of the future: a unified approach to image-based modeling and spatially immersive displays*, ACM, 1998, pp. 179–188.
- [51] J. RASKIN, *The humane interface*, Addison-Wesley, 2000.
- [52] E. S. RAYMOND, *The Cathedral and the Bazaar*, O'Reilly Media, Inc., 2008.
- [53] RESONATE, *Belgrade new media festival: Education*, 2013.
[online]. Available at: <<http://resonate.io/2013/education>> [Accessed 6 May 2015].
- [54] M. REUNANEN, *Computer demos – what makes them tick?*, 2010.

- [55] B. RIDEL, P. REUTER, J. LAVIOLE, N. MELLADO, N. COUTURE, AND X. GRANIER, *The revealing flashlight: Interactive spatial augmented reality for detail exploration of cultural heritage artifacts*, Journal on Computing and Cultural Heritage (JOCCH) - Special Issue on Interacting with the Past, 7 (2014), p. Article No. 6.
- [56] K. RIJNIEKS, *Cinder: Begin Creative Coding*, Packt Publishing, 2013.
- [57] K. RIJNIEKS, *Ofxpimapper github repository*, 2014.
[online]. Available at: <<https://github.com/kr15h/ofxPiMapper>> [Accessed 1 Jan 2015].
- [58] K. RIJNIEKS, I. SPICAKA, AND P. BURAVICKY, *MetaspHERE*, 2013.
[online]. Available at: <<https://vimeo.com/88571959>> [Accessed 26 Apr 2015].
- [59] D. M. RITCHIE AND K. THOMPSON, *The unix time-sharing system*, Bell System Technical Journal, 57 (1978), pp. 1905–1929.
- [60] L. E. ROSEN, *Open source licensing*, Prentice Hall PTR, 2005.
- [61] I. RYABOFF, *Visution mapio 2*, 2010.
[online]. Available at: <<http://visution.com>> [Accessed 25 Sep 2015].
- [62] SEMVER.ORG, *Semantic versioning 2.0.0*, 2009.
[online]. Available at: <<http://semver.org>> [Accessed 15 Mar 2015].
- [63] D. SHIFFMAN, *The Nature of Code*, Daniel Shiffman, 2012.
- [64] B. SHNEIDERMAN, *Direct manipulation: A step beyond programming languages*, ACM, 1981, p. 143.
- [65] R. SODRE, *Blendy VJ*, Studio Avante, 2012.
- [66] P. SPINRAD, *The VJ Book: Inspirations and practical Advice for Live Visuals Performance*, Feral House, 2005.
- [67] SPOUT, *Realtime video sharing framework for windows*, 2015.
[online]. Available at: <<http://spout.zeal.co>> [Accessed 25 Sep 2015].
- [68] STACKOVERFLOW.COM, *Stack overflow*, 2015.
[online]. Available at: <<http://stackoverflow.com>> [Accessed 14 May 2015].
- [69] T. STAPENHURST, *The Benchmarking Book: A How-to-Guide to Best Practice for Managers and Practitioners*, Elsevier/Butterworth-Heinemann, 2009.
- [70] B. STROUSTRUP, *Stroustrup: Faq*, 2010.
[online]. Available at: <http://www.stroustrup.com/bs_faq.html#invention> [Accessed 23 Sep 2015].

BIBLIOGRAPHY

- [71] ———, *The C++ Programming Language*, Addison-Wesley Professional, 4 ed., 2013.
- [72] SUBVERSION.APACHE.ORG, *Apache subversion*, 2015.
[online]. Available at: <<https://subversion.apache.org>> [Accessed 24 Sep 2015].
- [73] A. SWIFT, *A brief introduction to midi*, 1997.
[online]. Available at: <http://www.doc.ic.ac.uk/~nd/surprise_97/journal/vol1/aps2> [Accessed 22 Aug 2015].
- [74] H. TAKEUCHI AND I. NONAKA, *The new new product development game*, 1986.
[online]. Available at: <<https://hbr.org/1986/01/the-new-new-product-development-game>> [Accessed 25 Sep 2015].
- [75] M. THOMPSON AND P. GREEN, *Raspbian*, 2015.
[online]. Available at: <<https://www.raspbian.org>> [Accessed 24 Sep 2015].
- [76] L. TORVALDS, *torvalds/linux*, 2015.
[online]. Available at: <<https://github.com/torvalds/linux/releases>> [Accessed 23 Sep 2015].
- [77] TSP.PLASA.ORG, *Ansi e1.11 - 2008*, 2013. *entertainment technology - usitt dmx512-a, asynchronous serial digital data transmission standard for controlling lighting equipment and accessories.*, 2008.
[online]. Available at: <http://tsp.plasa.org/tsp/documents/published_docs.php> [Accessed 25 Sep 2015].
- [78] J. TURRELL, *Afrum*, Guggenheim Museum, 1967.
- [79] TYPEFORM, *Forms done awesomely*, 2015.
[online]. Available at: <<http://www.typeform.com>> [Accessed 14 May 2015].
- [80] VAM.AC.UK, *A history of computer art*, 2015.
[online]. Available at: <<http://www.vam.ac.uk/content/articles/a/computer-art-history>> [Accessed 29 Sep 2015].
- [81] J. VAN CLEAVE, *jvcleave/ofxomxplayer*, 2015.
[online]. Available at: <<https://github.com/jvcleave/ofxOMXPlayer>> [Accessed 15 Mar 2015].
- [82] P. VIDANI, *Video projection mapping*, 2010.
[online]. Available at: <<http://videomapping.tumblr.com>> [Accessed 24 Sep 2015].
- [83] W. WANG, G. POO-CAAMAÑO, E. WILDE, AND D. M. GERMAN, *What is the Gist?: understanding the use of public Gists on GitHub*, IEEE Press, 2015, pp. 314–323.

- [84] WIKIPEDIA, *Creative coding*, 2015.
[online]. Available at: <http://en.wikipedia.org/wiki/Creative_coding> [Accessed 14 May 2015].
- [85] ——, *Projection mapping*, 2015.
[online]. Available at: <https://en.wikipedia.org/wiki/Projection_mapping> [Accessed 24 Sep 2015].
- [86] ——, *Raspberry pi*, 2015.
[online]. Available at: <http://en.wikipedia.org/wiki/Raspberry_Pi> [Accessed 6 May 2015].
- [87] A. D. WILSON AND H. BENKO, *Combining multiple depth cameras and projectors for interactions on, above and between surfaces*, ACM, 2010, pp. 273–282.
- [88] M. WRIGHT, *The open sound control 1.0 specification*, 2002.
[online]. Available at: <http://opensoundcontrol.org/spec-1_0> [Accessed 19 Jul 2015].

