ČVUT
ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE

ZADÁNÍ DIPLOMOVÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení:	**Petrov**	Jméno: **Vadim**	Osobní číslo: **406747**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra počítačové grafiky a interakce**

Studijní program: **Otevřená informatika**

Studijní obor: **Počítačová grafika**

II. ÚDAJE K DIPLOMOVÉ PRÁCI

Název diplomové práce:

Pokročilý nástroj pro produkci vizuálních efektů pro živá vystoupení

Název diplomové práce anglicky:

Advanced tool for production of visual effects for live performances

Pokyny pro vypracování:

Na základě poznatků z bakalářské práce navrhněte, realizujte a otestujte nástroj pro vizuální doprovod živých hudebních vystoupení, řízený datovým tokem MIDI. Identifikujte slabá místa v předchozím řešení a popište možnosti jejich vylepšení. Seznamte se s dostupnými technologiemi a platformami a diskutujte zasazení nového nástroje do stávajících pracovních postupů vizuálních umělců (VJs), hudebníků a dramaturgů elektronické hudby (DJs). Nástroj otestujte z technického a uživatelského hlediska pomocí některé z metod testování uživatelských rozhraní (usability engineering) a zhodnoťte i z hlediska uměleckého srovnáním s existujícími VJ nástroji.

Seznam doporučené literatury:

Rick Snoman (2008) Dance Music Manual. Focal Press.
MIDI Manufacturers Association (1996) MIDI 1.0 Detailed Specification.

Jméno a pracoviště vedoucí(ho) diplomové práce:

doc. Ing. Adam Sporka, Ph.D., Katedra počítačové grafiky a interakce

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) diplomové práce:

Datum zadání diplomové práce: **03.12.2019** Termín odevzdání diplomové práce: **24.05.2019**

Platnost zadání diplomové práce: **30.09.2021**

[redacted]	[redacted]	[redacted]
doc. Ing. Adam Sporka, Ph.D. podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	prof. Mgr. Petr Páta, Ph.D. podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Diplomant bere na vědomí, že je povinen vypracovat diplomovou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v diplomové práci.

11.12.2019	[redacted]
Datum převzetí zadání	Podpis studenta

FACULTY
OF ELECTRICAL
ENGINEERING
CTU IN PRAGUE

Master's thesis

Advanced Tool for Production of Visual Effects for Live Performances

Bc. Vadim Petrov

Department of Computer Graphics and Interaction
Supervisor: doc. Ing. Adam Sporka, Ph.D.

January 7, 2020

Acknowledgements

Thanks to Adam for having patience during the lengthy process of settling on a topic, to Malcuth, David and Hepex for the peer reviews, to Human Ketchup and Malcuth again for the opportunities to experiment, to Marijana for support and to everyone who came to the shows.

Declaration

In Prague on January 7, 2020 . . . .

Citation of this thesis

Petrov, Vadim. *Advanced Tool for Production of Visual Effects for Live Performances*. Master's thesis. Czech Technical University in Prague, Faculty of Electrical Engineering, 2020.

Abstrakt

Díky probíhající proměně koncertů a hudebních vystoupení obecně do podoby multimediálních zážitků se otevírá nový prostor pro zkoumání vztahu hudby a obrazu. Nástroj Skinny Mixer nabízí míchání videí, použití vizuálních efektů a synchronizaci s hudbou. Nástroj je navržen pro živá vystoupení a během vývoje byl opakovaně nasazován v rámci audiovisuálního vystoupení Simulacrum AV.

Klíčová slova audiovisuální, vystoupení, MIDI, VJ, C++, openFrameworks

Abstract

As live concerts and music performances generally transform into multimedia experiences new space opens for the exploration of the relation between sound and image. Skinny Mixer is a tool offering to mix multiple videos, apply visual effects and synchronize them to music. Intended for live shows it has been employed repeatedly during the development process as part of a live audiovisual performance Simulacrum AV.

Keywords audiovisual, performance, MIDI, VJ, C++, openFrameworks

Contents

List of Figures

List of Tables

Introduction

This work is a spiritual successor to the author's bachelor's thesis *Tool for visualization of a MIDI data stream for live performances* [1]. The central idea connecting both works is to build a unique audiovisual live performance. *Audiovisual* as in using both sound and image in unison, interconnected and dependent. While both elements could be theoretically dependent on each other to form a truly synesthethic experience, in reality one usually leads and the other follows. Such is the case in both the bachelor's thesis (which will be extensively scrutinized in chapter 2) and this work where the image depends on the sound. And *live performance* as in the developed tool is intended for stage usage in front of live audience in a semi controlled environment. Such environment sometimes offers plenty and usually not nearly enough time to prepare the show and the set, other performers or even members of the audience can mingle with the setup (check out all the online videos of someone spilling their drink onto the DJ...) and so it needs to be robust and simple enough do diagnose and fix problems in a hurry.

While there were some tools suitable to prepare such a performance at the time of writing the bachelor's thesis and there are even more now (and the older ones are better, more developed and more expensive than they used to be) there still remains a lot of room for experimentation and unique self expression. The bachelor's thesis described the Czech local scene as plagued with basic setups without creativity. Despite some venues and promoters managed to build stages with impressive amounts of light fixtures, their control often had close to none relation to the music playing. This situation improved with events described in chapter 1, which managed to both open new opportunities and raise the expectations of select audiences. While performing with the system from the bachelor's thesis the author had many opportunities to check the crowd reaction, consider technical improvements and plan the next iteration of a custom audiovisual performance.

Goals

1. **Research state-of-the-art solutions** The situation has changed since the publication of the bachelor's thesis and a further and more detailed investigation into what is available both from the technical and the social aspect is required. Designing a live audiovisual performance is a complex task with many options to consider and the solution developed as part of this work should fit reasonably well into the currently used workflows.

2. **State benefits & investigate drawbacks of the bachelor's thesis** Why is another system required, if at all? Retrospectively evaluate the standpoints and the results of the bachelor's thesis, pinpoint key elements to reuse or the discard and in a symbolic way end the run of the system.

3. **Design & implement an advanced system** Take inspiration from the conceptual, design and technical decisions made as part of the bachelor's thesis and improve upon them. Specify requirements that the system should fulfil, consider available development environments and put them to use.

4. **Evaluate the system** The evaluation should consider multiple standpoints including system performance, usability engineering values and artistic impression.

Development

The system in question started development long before the writing of this document. The ideas that later turned into the analysis (chapter 2) part were already brewing while performing with the system from the bachelor's thesis. Having to endure its many drawbacks personally and trying to overcome them has brought inspiration and motivation to pinpoint which parts make the performance, which are redundant and which proved to be unimportant and can be dropped entirely.

The new system has been put to real life usage right from the first prototype which organically prioritized the most missing features. This resulted in a somewhat chaotic development which led to the many minor and few major issues described in chapter 5. To avoid using the rather cold phrase "the system" all the time, this time the system got a name. At first it was called *TVAEM – Triggered Video & FX Mixer*, a rather terrible name later replaced by *Skinny Mixer*, the name used throuout the entire text.

Chapter 1

State-of-the-art

There are many options to choose from when preparing an audiovisual performance using today's technologies. This chapter covers some of them however does not try to achieve total coverage for multiple reasons. Especially with experimental arts the border between audio, visual, performance, installation and generally art and technology becomes very blurry. The case is usually that a single performance combines many elements described below and the real challenge is not to acquire them, but to use them in an artistically sensible manner.

1.1 Hardware

The hardware used for live performances is perhaps the easiest to track. Traditional parabolic light fixtures with light bulbs have been mostly replaced by the more versatile and often robust LED fixtures. A standard protocol for light fixtures and other stage equipment is the DMX-512. A single *universe* offers 512 channels of control with the option (and often need) to link multiple universes with custom devices. A simple modern LED light fixture can be set to an address (a number up to 512) with usually three to ten parameters linearly addressable relative to the base address. Those parameters can control the red, green and blue (or even white) color channels, total opacity (power output), fade or strobe speed etc. More advanced light fixtures often move in usually one or two degrees of freedom. DMX devices are usually connected in a linear fashion ("daisy-chained") as shown on Figure 1.1. Another popular DMX fixtures include a gobo which shines stencil shapes and smoke machines – often used in combination to create 3D shapes. RGB lasers have their own standard of communication – ILDA – which uses the common DB-25 connector.

The wide variety of DMX devices however come with a disadvantage: they are so advanced that they tend to turn out expensive and require know how to operate. This in turn makes them infeasible for smaller artists and leaves

Figure 1.1: DMX connection example. On the top there is a DMX controller with the light fixtures on the bottom. Note that unlike MIDI notes, DMX addresses start from 1 and that each device has 14 channels available. Taken from [2].

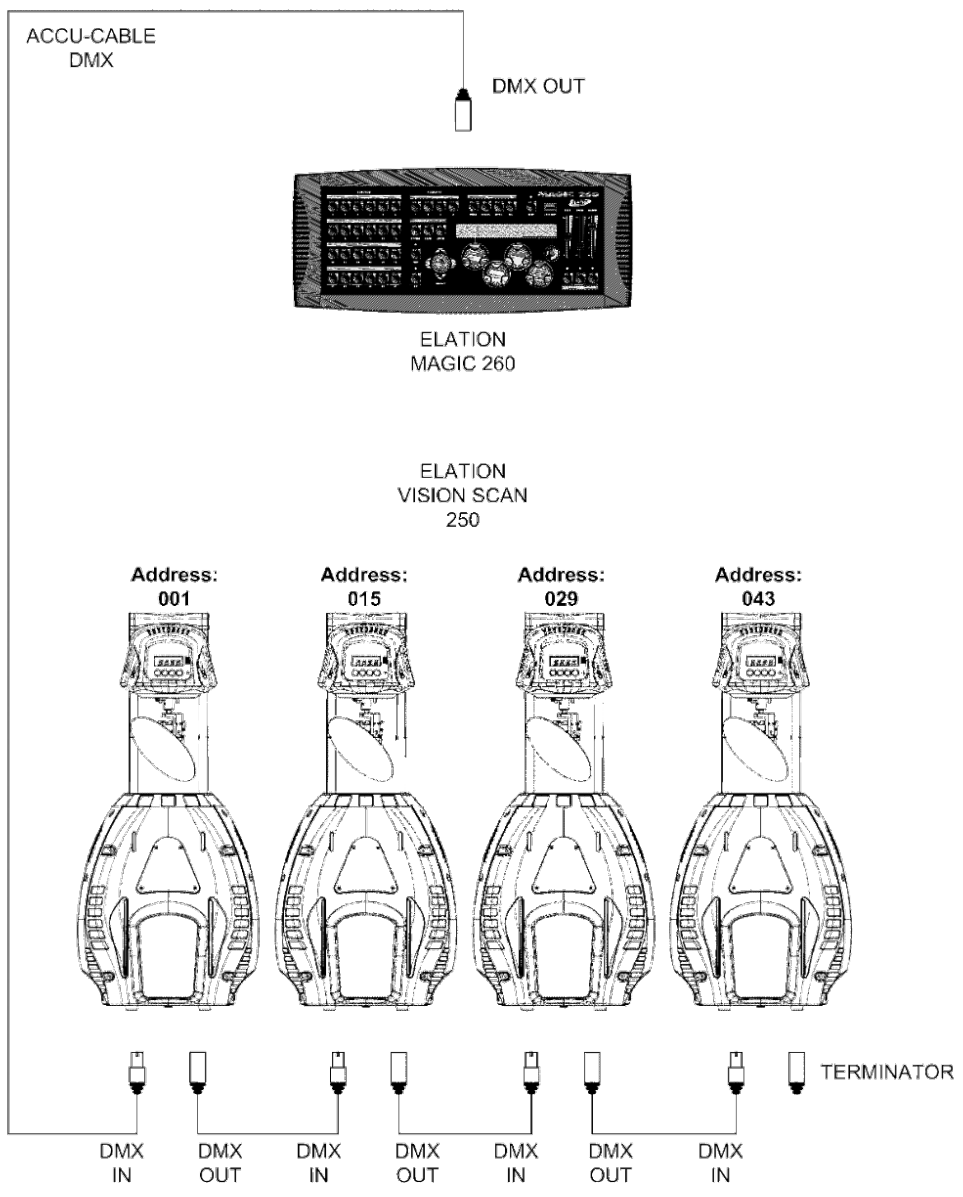
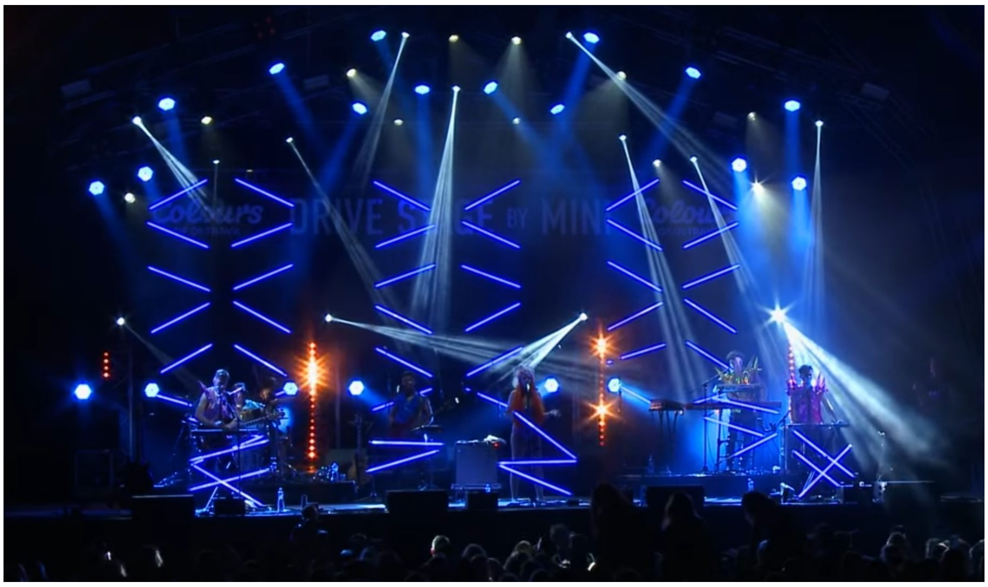


Figure 1.2: Mydy Rabycad stage setup with neon tubes [3].



only large projects and venues with enough budget to afford them. A popular alternative turned out to be simple neon tubes, which can be programmed to light up in simple patterns. These can also be arranged into various shapes to fit on both smaller and larger stages. Case in point is the current stage setup of live electro swing band Mydy Rabycad with rather opulent shows including many lights, fireworks, confetti and those custom tubes programmed by their guitar player Ondřej Slánský.

The biggest recent advancements have been recently made in LED screens and projections. While the trend of video mapping seems to be becoming less popular in stage design, LED screens small and large have been used extensively. Take for example Noisia's live performance Outer Edges at Rampage festival 2017: although the artists claim that the performance is more focused on playing electronic music live unlike in a DJ set, the state of the art visual side has been improved by the massive amount of LED screens accros the entire venue which hosts an audience of thousands. See the screenshot 1.3 from a 360 degree video for reference.

The last notable piece of hardware used in recent audiovisual performances is the now discontinued Microsoft Kinect. A line of motion sensing devices originally meant as a gaming accessory to the Xbox line of game consoles quickly found its place among experimental visuals developers. A good example of its potential was the performance of Sinjin Hawke & Zora Jones at Sónar festival 2018 when the performers recorded themselves playing their music and dancing to it and used various effects to obscure and deform the shapes of their bodies.

Figure 1.3: Noisia performing the Outer Edges live show at Rampage festival 2017 in Belgium [4].

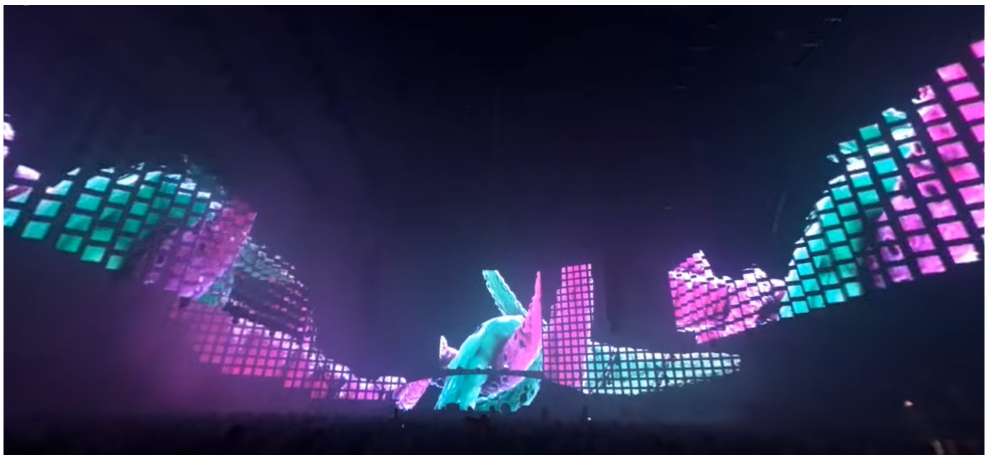
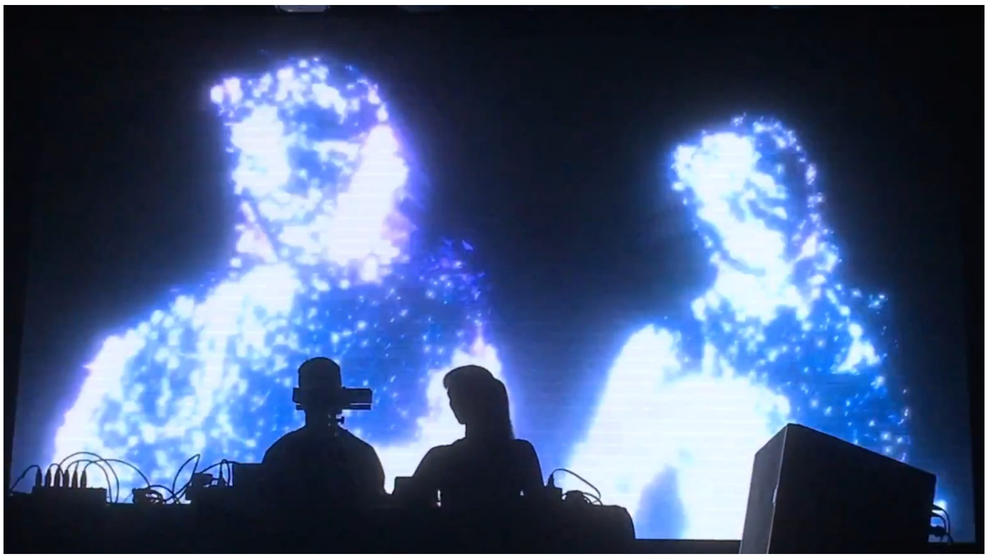


Figure 1.4: Sinjin Hawke & Zora Jones performing their AV Live at Sónar festival 2018 [5].



1.2 Software

Complimenting the hardware equipment there are many software options. Some software is directly tied to hardware, most notably drivers but also tools that change and store button mappings, presets, update firmware etc. Those do not need to be covered here.

Probably the most common tool for a live audiovisual performance is a DAW such as Ableton Live or Bitwig (both of which are used as part of Simulacrum AV on separate machines). A DAW is not only a studio tool for music production but often doubles as an essential part of a live electronic music setup. It hosts VST plugins which implement a wide variety of functions. Apart from virtual versions of standard audio studio gear (mixers, compressors, sequencers etc) and even virtual modular synthesizers they can provide access to network or provide input / output to custom hardware such as the aforementioned Kinect or perhaps physical motors. Indeed a DAW is often the centerpiece for an entire set.

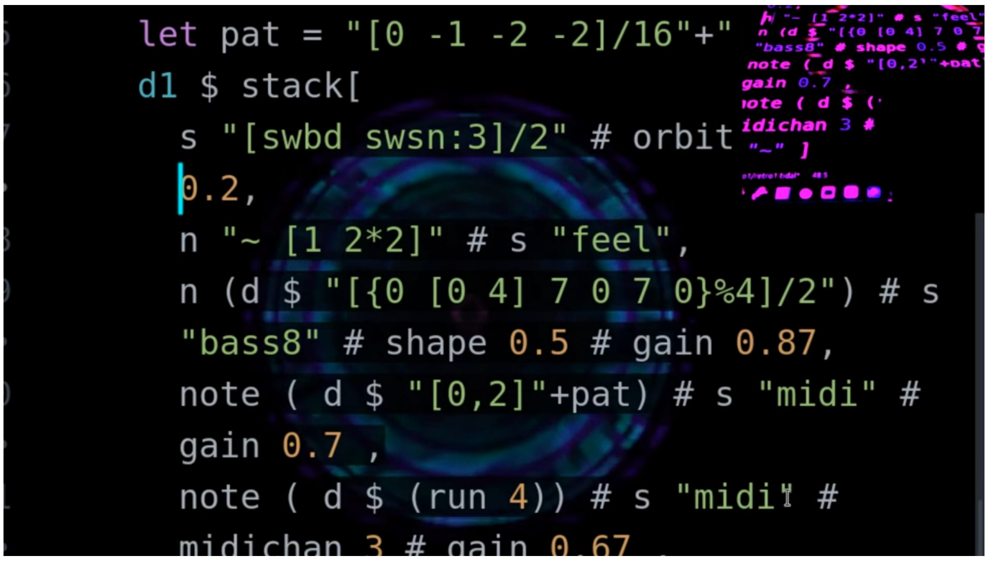
A unique feature of Ableton Live is its very close relation with Max/MSP, a visual coding language for music and multimedia. Ableton Live hosts custom Max plugins similar to running a VST without the typical overhead of building a standard C++ application. It also allows for editing said plugins on the fly which further improves its creative potential.

Visual coding languages can of course be run standalone. A free alternative to Max/MSP is Pure Data or simply Pd. Another option is vvvv – a multipurpose toolkit. All the visual programming environments share some benefits and some drawbacks. The workings of the program and the GUI are a single thing – literally what you see is what you get. Once a user is accustomed enough those environments allow for fast prototyping, small scale experiments and generally tend to boost creativity. Without proper "cable management" and effective usage of screen space however, larger projects tend to get cluttered and confusing rather fast. These specialized environments also lack the wide options of a general purpose programming language.

An interesting approach is the one taken by live coding enthusiasts. A movement with love for both art and engineering coming from universities and research institutes all over the world, it is an inclusive group balancing between art and activism. Live coders write improvised code on stage with the audience being able to witness both the source and the output in one or multiple projections. Those can be sound or visuals, often both. One or many persons can form a single performance, sometimes with coders joining or leaving as if on a music jam. Many languages and environments exist, such as TidalCycles (based on the Haskell programming language) [6], Gibber (based on JavaScript and running in a web browser) [7] or LiveCodeLab (heavily using WebGL) [8].

Some of the live coding environments even tend to get into the last area to be described here and that is VJ and mapping tools. A projection mapping

Figure 1.5: Diego Dorado performing with TidalCycles and the Atom text editor [9].



```
let pat = "[0 -1 -2 -2]/16"+"
d1 $ stack[
  s "[swbd swsn:3]/2" # orbit
  0.2,
  n "~ [1 2*2]" # s "feel",
  n (d $ "[{0 [0 4] 7 0 7 0}%4]/2") # s
  "bass8" # shape 0.5 # gain 0.87,
  note ( d $ "[0,2]"+pat) # s "midi" #
  gain 0.7 ,
  note ( d $ (run 4)) # s "midi" #
  midichan 3 # gain 0.67
```

tool's defining features is the ability to warp videos in order to project them onto complex real life surfaces and blending the images of multiple projectors to gain larger resolution. The purpose of a VJ tool is to mix and combine videos, not unlike Skinny Mixer. While many such tools exist (Resolume Arena / Avenue, ArKaos GrandVJ or VDMX to name the biggest players), offering plethora of features, an out-of-the box solution to start and stop video layers is missing. Some have analytical sound engines capable of syncing video effects to music, however the options are limited. The most versatile solution would be to combine Skinny Mixer and an established VJ tool and it is indeed a part of the plans for Simulacrum AV.

This list of hardware and software solutions to deploy an audiovisual show is by no means complete, rather a selection of some of the options the author has encountered and experimented over the recent years.

1.3 Events

Many events have recently been paying more attention to or have been specifically designed to present modern audiovisual performances. Take for example Lunchmeat Festival which has been running in Prague since 2010. The team presents it as an event for "Live AV Experiences. Distinctive dramaturgy and curating. Sound and visual art in symbiosis. Experimental digital media art works from all over the world" [10]. Similar are Unsound Krakow, a member of Shape Platform which "consists of 16 festivals and art centres and aims to support, promote and exchange innovative and aspiring musicians and inter-

Figure 1.6: Aid Kid & Pavel Karafiát performing at Lunchmeat festival 2019 [12].

disciplinary artists with an interest in sound" [11], and Sónar, Barcelona with its own educational platform happening side by side with the main event.

Whereas the former events still focus mostly on (albeit often very progressive) music, events such as Life Performers Meeting [13] or Generate! [14] are designed to offer a save space for artists to meet, learn, perform and generally share and develop ideas on the edge of art and technology.

The last event that requires mentioning is Signal Festival Prague. While its focus is more on static installations than live performances, due to it being in the streets of Prague and available mostly for free it had a major impact on demonstrating the possibilities of current technologies to the general public. As the website proudly says, "From videomapping show and light design, it has grown to become a well-respected digital art festival and art/tech event which currently belongs to the most significant ones in the Czech Republic" [15].

1.4 Bachelor's thesis

A piece of software requiring special attention is the author's bachelor's thesis. This work draws a lot of inspiration from it and is in fact a spiritual successor to the "system" as it was plainly called in the bachelor's thesis. The entire project was much more primitive and independent. It was based less on proper research but on creativity and the desire to create something new, something unique.

The basic idea was to use realtime 3D graphics (OpenGL specifically) and make use of the creative freedom this area offers. Synchronized with live music using MIDI just as in this work (see below) the goal was to develop a custom audiovisual performance offering a synesthetical experience to the audience. A custom two man music group was founded to demonstrate this system on stage, drawing inspiration a wide array of music genres such as techno, drum & bass or black metal. Three songs were originally written with custom visuals for each and performed live at Klubovna in Prague on 9. 5. 2016 [16].

While the initial goal was mostly achieved the system proved to have many design and even concept flaws and a decision was made to drop it entirely. That is not to say that the performances were not worth developing further – in fact the system and the performance continued to run for more than a year after the initial release with new music, new visuals and new additions and improvements to the already developed material. An EP was even recorded (albeit not in a professional music studio) and was not far from release. Eventually however the development costs exceeded the artistic gain and need for a new approach arose.

Further information on the system from the bachelor's thesis offers in much more detail chapter 2.

Chapter **2**

Analysis

Since this work is spiritually a continuation of the project developed for the bachelor's thesis this section will focus on the previous tool. The goals of the bachelor's thesis were successfully met however some unexpected problems arose and were not properly described in the previous text. Those were mostly discovered after the first public performance using the tool which served primarily as a proof of concept and did not cover all use cases.

As a reminder the bachelor's thesis specified the following technical requirements [1]:

Performance	The system will use appropriate algorithms, workflows and recent versions of libraries to reliably run on the testing device.
Resources management	The system will properly manage acquired resources and minimize memory requirements.
Communication	The system will display its state and notify about events and errors. This communication will be bidirectional if required and will not disturb the visuals.
Configurability	The system will have an option to be manually configured based on the conditions it's being run in. The system will support both traditional and widescreen displays.
Stability	The system will be stable enough to be reliably used during live performances.
Platform	The system will be developed for Microsoft Windows family of operating systems to ensure the possibility to be run without hardware equipment using common DAWs.
Structure	The source code will be extendable and logically structured.

Testing The system will be tested with specified songs. The system will appropriately react to performer mistakes, irregularities or improvisation.

Workflow The system will fit the described creative workflow.

The bachelor's thesis then describes a workflow for the development of music and visuals in one process. The workflow itself is rather complicated and will be discussed below.

2.1 Bachelor's thesis topics

In the analysis section the bachelor's thesis covered various topics relevant to the subject of live music visualizations. Since the situation has not changed all that much we can just briefly skim over those.

The first covered topic was the definition of traditional versus electronic music. The idea was that traditional music relied on what most people understand under a music instrument i.e. percussion, brass or string instruments and such whereas electronic music comes from synthesized or sampled sounds. The point of this distinction was that musicians of the respective kinds rely on different visual forms to accompany their performance. Traditional musicians would use stage props and costumes, electronic music artists would prefer projections and both groups would often use lights and seldomly dancers. In the end the text arrived at performers who connect both worlds such as Pendulum or The Glitch Mob.

While this distinction can still be applied it is not as relevant as the work suggested. Both the original text and Skinny Mixer rely on MIDI input and while that is most easily acquired from electronic or virtual instruments, it is also possible to use sensors to analyze traditional instruments (such as drum kit triggers or custom guitar pickups) or signal processing algorithms to obtain MIDI messages as well. In the end the visualization system does not know or care about the source of the information.

The bachelor's thesis also commented on various approaches on how to synchronize image to sound. The most common one remains control by a one or more technician, usually provided by the venue. This brings a problem, actually one of the major motivating factors for both this and the previous work: the technician often does not know what the musician is about to do and any bold or radical changes in visuals, which should accompany dynamic changes in music, are based on guessing. More experienced light technicians and VJs do tend to take better guesses but they still miss. This can be improved if the visual crew prepares a custom visual show in cooperation with the musicians, but the most common tools still limit what they can achieve. More on this in chapter 1.

Another option for synchronization is signal analysis. This was dismissed in the bachelor's thesis as too messy and often simplistic, however was part of the original proposal for Skinny Mixer. It was dropped from the original prototype for two reasons. The first was time constraints, the priority was just not high enough. The other one was the fact that many other (and better) tools for this task exist, often present in DAWs as a factory default. That means the job can be delegated and the tool itself can focus on its core functionality.

The last mentioned synchronization option was using CV/gate known from the world of modular synthesizers. This kind of signal is however convertible to MIDI.

2.2 Bachelor’s thesis benefits

A major benefit of the work was its artistic success. The duo *Rejfpank* assembled specifically to write music and perform with the developed system performed at multiple occasions for various kinds of audiences. To point out a few of those: the initial test performance at Klubovna supported by a jungle music two piece band Human Ketchup who later also accompanied the first public performance with Skinny Mixer, the Mecca of Prague underground electronic music Cross Club during Žižkovská noc festival or Nod Café as part of support program during Signal festival, the annual event for light shows and video mapping.

What really made the system stand out were unique visuals for each song. Every sound was mapped to a custom visual element, be it transformations in 3D space, color changes, post processing effects, 2D overlays etc. This level of detail is rather uncommon even today for a good reason: it comes at a steep price as discussed of the drawbacks section.

Another benefit of the system was its fast response time. As table 2.1 shows the latency averaged 3.3 and 5.2 frames at 60 frames per second which translates to around 0.1 s or better. Given the low light conditions such performance is designed for, this result is more then enough, especially on the mid to low tier hardware the tests were performed on.

Table 2.1: Bachelor’s thesis latency test [1]. The number of frames (at 60 fps) gives the latency between pressing a physical button and the visual on-screen response.

MIDI device	Frames										Avg
Alesis SR-16	7	6	7	4	5	4	3	5	6	5	5,2
Akai Professional MPX16	4	3	4	3	3	4	2	3	3	4	3,3

2.3 Bachelor's thesis drawbacks

The biggest issue and a major reason for a clean start with Skinny Mixer was the cumbersome development process of new visuals. At the time of publication three songs were finished with fourth on its way. The fourth, titled Hrdinové modulární scény, was not developed during the somewhat stressful period of thesis finalization and had much more time and energy at its disposal, which resulted in the musically and visually most complex piece of work of all the songs produced. However the amount of resources spend at this production meant that no further songs were later developed and the project was eventually dropped.

The problem lies in the core architecture of the system. Developing a new C++ class for each song with a number of custom features is unreasonably cumbersome. Even an early prototype takes long to create and most changes, even minor ones, require a rebuild. The potential of having to do a full rebuild, including the proper development tools and libraries is an unreasonable requirement for such a system and seriously limits its possible application. About the only change one could make without a rebuild was swapping texture files while keeping the original file name. Everything else, including file names, MIDI mappings, even the st list (order of the songs) was hard coded.

This also meant that each show had its specific build and using the latest build (with new features, bug fixes etc) was not an option for replaying a past show.

Another obstacle is the challenging nature of 3D graphics development. Creating the models themselves takes time and considerable skill so even the very basic shapes used in the system required external help. Using OpenGL and having control over everything – the render pipeline, lightning, texturing, draw calls etc. presents an option for a lot of creative freedom however the responsibility is immense and makes the development process exhausting. Giving up some of this freedom in exchange for a quicker process seemed like a necessary step to make as described in chapter 3.

Chapter 3

Design

The basic concept follows from the bachelor's thesis: Skinny Mixer receives MIDI messages as its input and outputs a stream of visual data fit for projection. Instead of producing all the visuals as a combination of 3D and 2D objects in 3D space, the system will use standard videos and mix those together based on the MIDI input, hence the word "mixer" in the name. This major limitation was well thought through and comes as a solution to the problems described in chapter 2. This design is significantly more generic: it allows for configuration completely independent of the source code of Skinny Mixer, solving the *one class – one song* problem. It also delegates the responsibility of content creation – the visuals no longer need to be produced completely by this system. The videos can be reused shots from already published works or custom ones. Skinny Mixer then loads the required videos, interprets the input MIDI stream and plays or manipulates the videos accordingly.

This general overview is further described in more detail according to various methods of system design.

3.1 Main features

- Skinny Mixer triggers videos on or off based on MIDI *note on* and *note off* messages. Each video is assigned a single MIDI note.
- The videos are presented in layers fully overlapping one another. Each layer also has additional properties (see section 3.3)
- Skinny Mixer also offers graphics effects applied over the blended video layers. Those effects are also triggered using MIDI *note on* and *note off* messages and parametrized using MIDI *CC* messages.
- A collection of layers and effects is called a *scene*. A sequence of scenes is called a *show*. A show can be saved, loaded and manipulated.

- Skinny Mixer offers two windows: A control window with the GUI and an output window with only the visuals. If no videos are playing, this window is fully black.

3.2 Use cases

On-line slave	A performer is producing music live on stage and sending MIDI data to another machine running only Skinny Mixer. This machine is operated by another performer, whose task is to monitor the visual quality and make minor adjustments in the GUI or major ones using another MIDI device. This can be triggering of additional layers (such as masks) or effects, forcing or disabling layers, or choosing video files on the go. This is the way Skinny Mixer has been used as part of Simulacrum AV.
On-line master	A single performer operates a single or multiple machines running a DAW and Skinny Mixer. The DAW can have audio inputs to analyze or produce MIDI messages on its own. This is the way Skinny Mixer was used during the Human Ketchup release show.

3.3 Functional requirements

Status indication	Current FPS, video load success / failure and loading information are present in the GUI. Additional information is sent to the console.
Layers	The final image is comprised of two kinds of layers, video layers and visual effect layers, which can be triggered on or off by MIDI *note on* and *note of* messages.
Video layer properties	Each layer can be assigned a single video and has the following additional properties available: alpha opacity, blending mode, re-trigger toggle (whether further triggers continue playback or re-trigger the video from start), force play toggle, mute (disable) toggle.
Blending modes	Each video layer can be assigned one of a subset of blending modes described in [17]. Such a blending mode is defined by a separable blending function $c_r = B(c_b, c_s)$ where c_b is the backdrop color, c_s is the source color and c_r is the result. This function is separable because it is applied independently for each color component. The subset consists of the following:

Normal	$B(c_b, c_s) = c_s$
Multiply	$B(c_b, c_s) = c_b \times c_s$
Screen	$B(c_b, c_s) = 1 - [(1 - c_b) \times (1 - c_s)]$
Darken	$B(c_b, c_s) = min(c_b, c_s)$

Lighten $B(c_b, c_s) = max(c_b, c_s)$

Linear Dodge $B(c_b, c_s) = c_b + c_s$

Difference $B(c_b, c_s) = |c_b - c_s|$

Exclusion $B(c_b, c_s) = c_b + c_s - 2 \times c_b \times c_s$

Effect layer properties Each effect layer can be assigned one of the available effects and has these additional properties: Custom effect parameter, force play toggle and mute toggle.

Custom effects Each effect layer can be assigned a single effect from the following list. Those effects work in a similar manner to the blending modes specified earlier. They are defined by an effect function $F(c, p)$ with color c and a custom parameter p to dynamically alter their behaviour with c_{max} being maximum color value and p_{max} maximum value of the parameter. To fit the 7bit / 8bit MIDI paradigm, 127 has been chosen for p_{max}.

Solarize $F(c, p) = \begin{cases} c_{max} - c \text{ for } \frac{c}{c_{max}} \leq \frac{p}{p_{max}} \\ c \text{ otherwise} \end{cases}$

Posterize $F(c, p) = \frac{\lfloor c \times p \rfloor}{p-1}$

Color Shift This effect is not separable – it does not operate on independent color channels. $F(c_{rgb}, p) = \begin{cases} c_{brg} \text{ for } p \equiv 1 \mod 2, \\ c_{gbr} \text{ otherwise} \end{cases}$

Overdrive $F(c, p) = (c \times \max(1, p)) \mod c_{max}$

Mappability The following is mappable to MIDI messages: Video and effect triggers, video opacity, custom effect parameters, scene change, master alpha opacity.

Scene definition A scene can be assigned a name. A scene contains setup information for all the video layers.

Show definition A show contains information regarding the effect layers (they are shared for all the scenes). A master MIDI channel can be set and is applied to every layer in every scene of a show. A scene switch MIDI note can also be set to move the show one scene forward.

Show manipulation A show can be saved, loaded and can have a scene appended to and deleted from. A performer can move through the sequence of layers either sequentially or randomly.

Figure 3.1: Three video layers blended using the Difference and Exclusion blending modes. The blending mode of the first active layer is always disposed of.

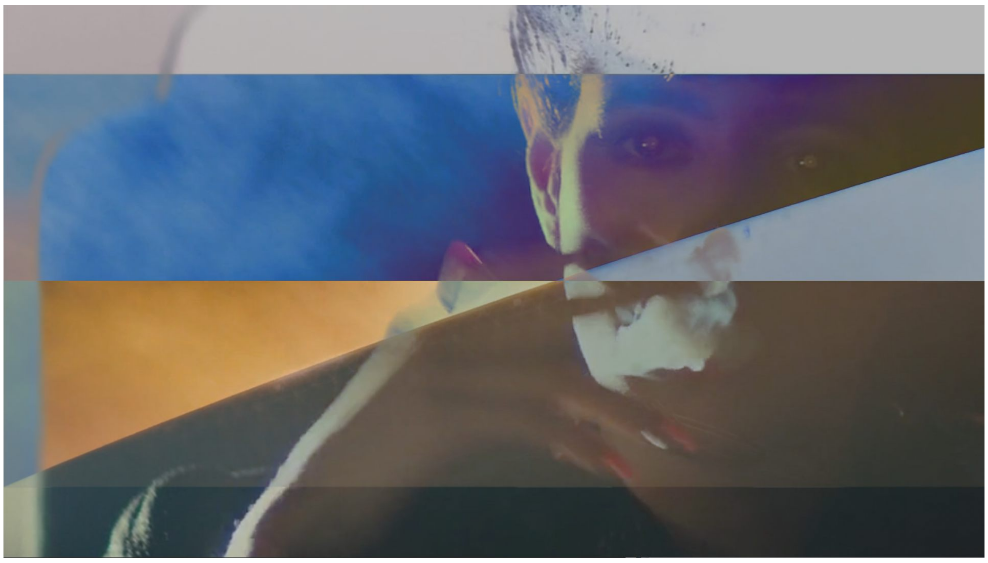


Figure 3.2: Three video layers blended using the Darken and Multiply blending modes.

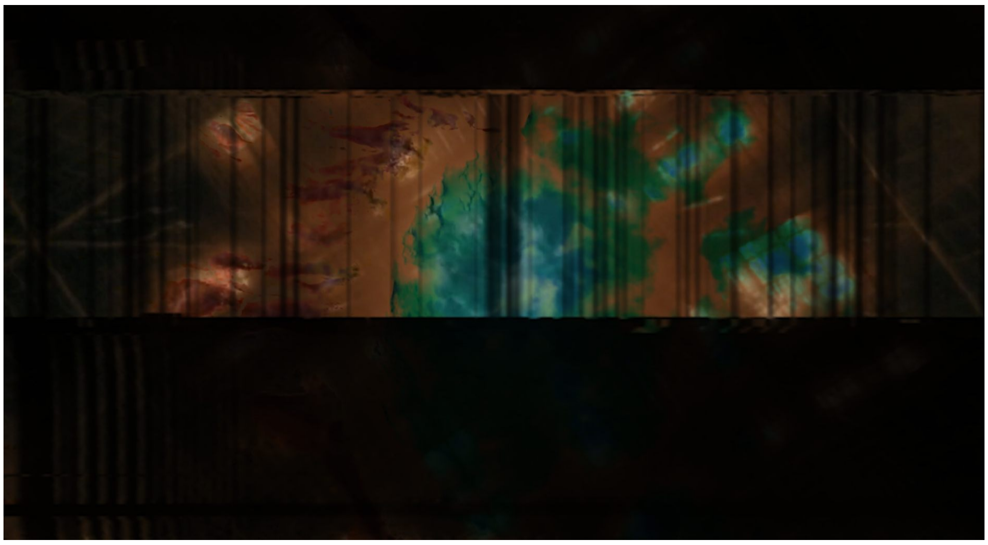


Figure 3.3: The Solarize effect applied to a still shot from the official video to Britney Spears – Toxic.

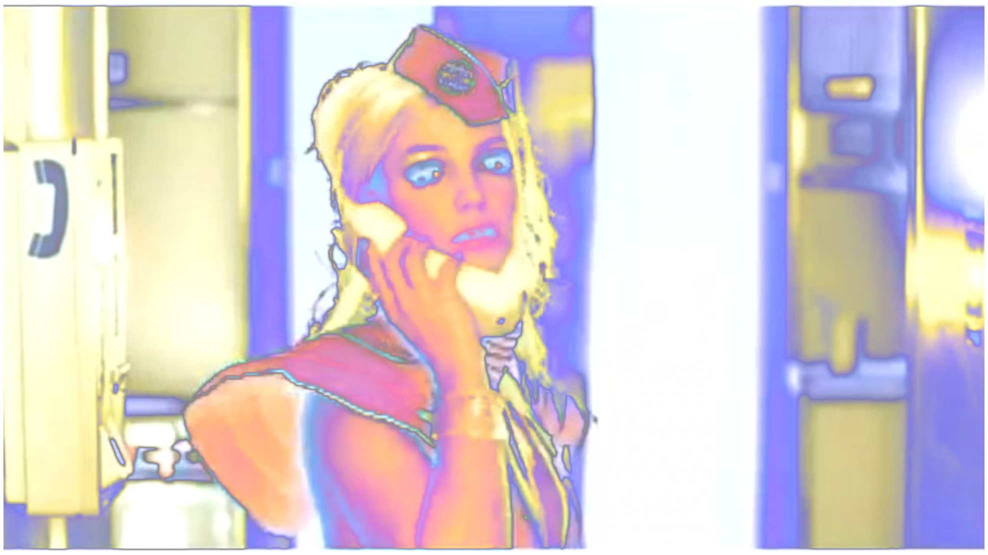


Figure 3.4: The Posterize effect applied to the same shot.

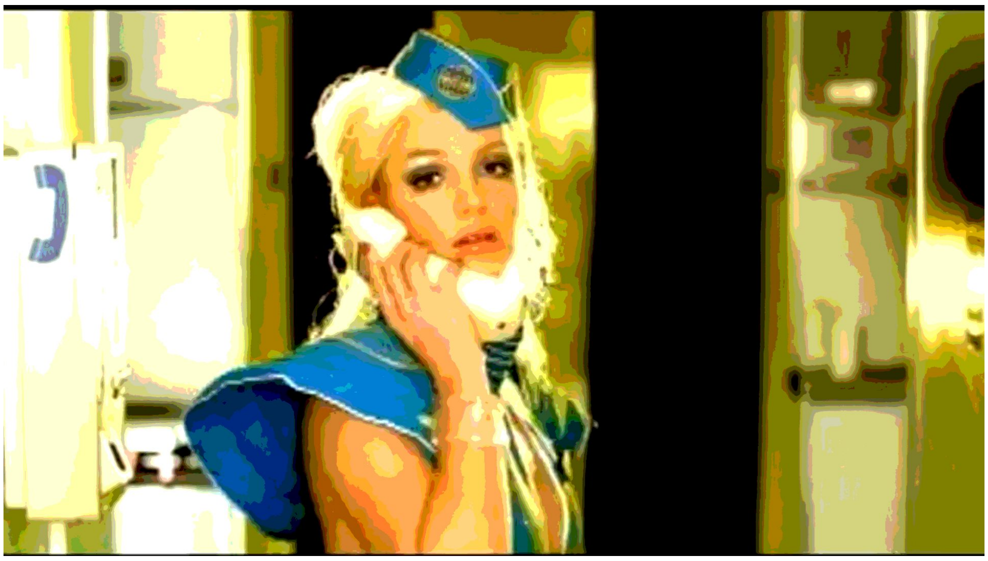


Figure 3.5: The second version of Color Shift effect – *rgb* to *gbr*.

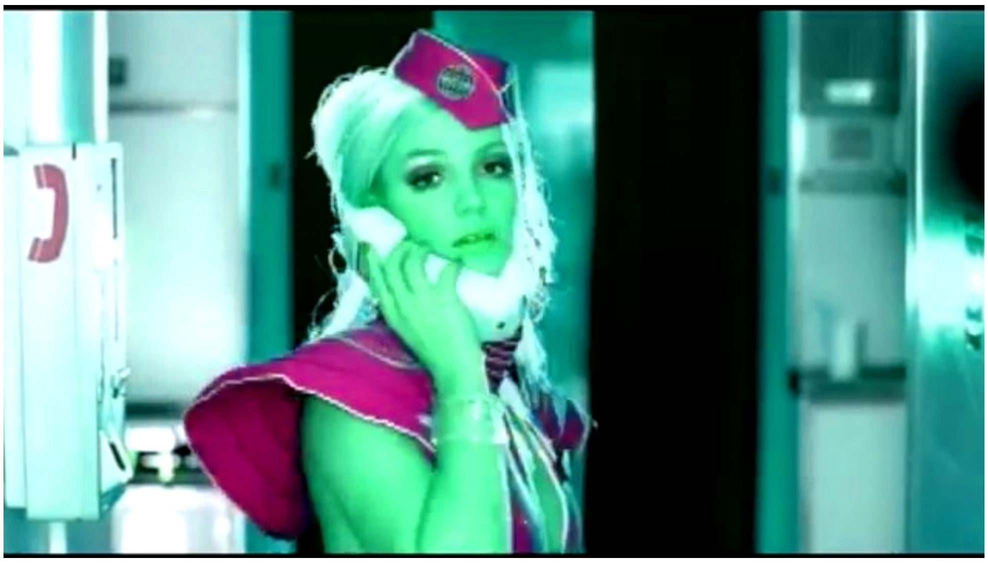


Figure 3.6: The Overdrive effect applied to the same shot. The number of artifacts rises quickly with the parameter, here $p = 2$ (see Figure 3.8 for comparism). The effect works well on upscaled videos with many compression artifacts such as this one.

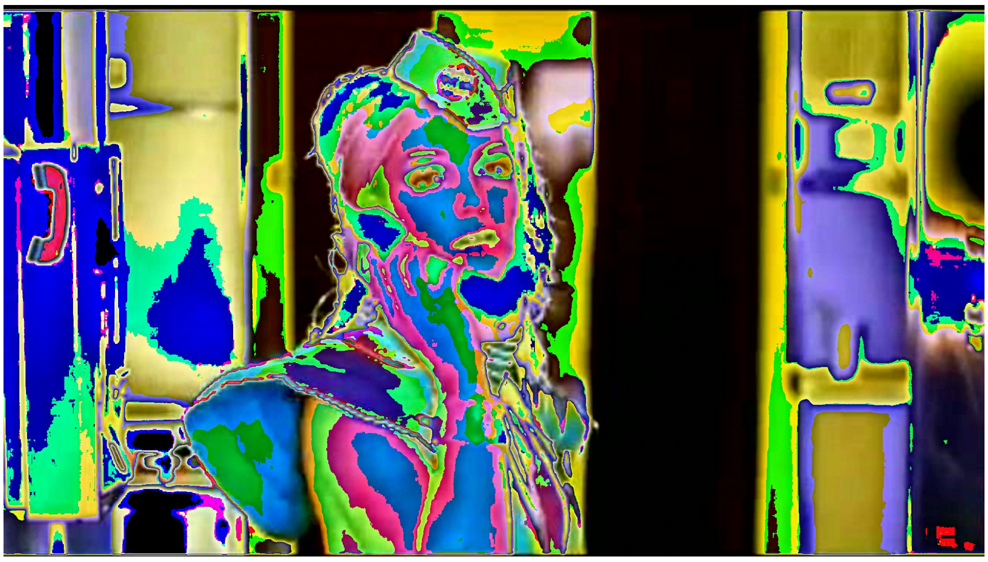


Figure 3.7: All of the previous effects combined. Switching effects on and off, combining them and dynamically changing p helps achieve rich visuals even with minimal video input.

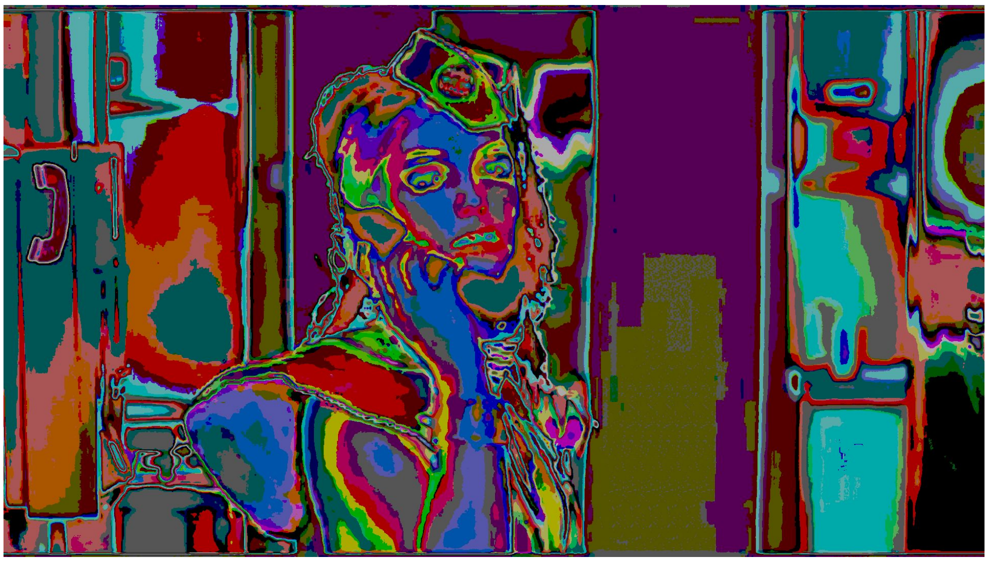
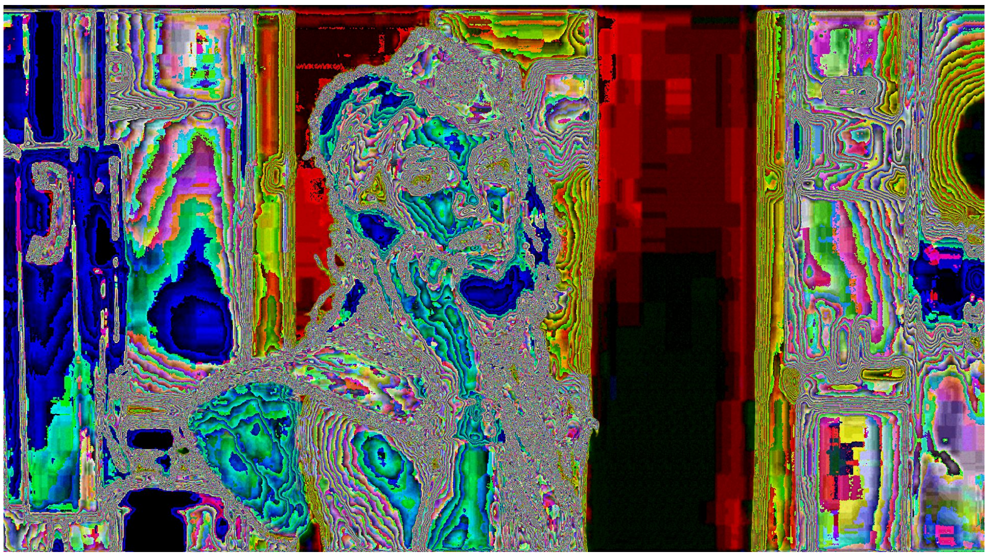


Figure 3.8: The Overdrive effect again with $p = 20$.



3.4 Non-functional requirements

Input	Multiple common video formats are supported. Videos of different lengths and display resolutions can be seamlessly mixed.
Platform	Skinny mixer is primarily designed to run on the MS Windows family of operating systems, however should contain as little platform specific code as possible to prepare for a macOS (and possibly Linux) ports.
Interface	The operation interface enables users to quickly develop a show without any knowledge of the system internals or any programming skills.
Serialization	The entire show can be saved and loaded to / from a file.

3.5 Performance requirements

The following performance requirements consider a standard output resolution of 1920 x 1080 pixels (Full HD).

Load times	Loading a show is fast enough so a switch from one show to another does not hinder the workflow.
Switch times	Switching to another scene is fast enough so the audience does not notice.
Latency	The latency between pressing a button on a hardware device and a visual on screen response is under $^1/_{30}$ s.
Fluidity	The target FPS is 30. To ensure fluidity, average FPS should stay over 25, worst case peaks over 20.

Chapter 4

Implementation

This section provides particular information about the implementation of Skinny Mixer. First a Unity3D prototype was quickly developed and shown to potential partners, however later it was required to decide on a more fitting technological development environment. The current architecture is described in detail and then the major iterations Skinny Mixer went through during the development process until it arrived at this current version – albeit still considered an unreleased alpha.

The source code is available publicly in a GitHub repository [18].

4.1 Unity3D prototype

As a proof of concept the first prototype of Skinny Mixer was created using Unity3D, a common multi purpose game engine. This engine allows for fast prototyping of 3D or 2D applications, skipping a lot of basic boiler code, however it is not as well suited for the specific behaviour Skinny Mixer requires. The engine offers a number of features with no benefit for Skinny Mixer (algorithms and tools for 3D graphics, physics simulation, artificial intelligence etc...) while at the same time some processes can be tedious or requiring too much overhang, such as fast configuration change or GUI creation.

This prototype offered some basic features: Mapping one or more MIDI messages to a video layer, 5 layers in total and the Linear Dodge blending mode (a simple sum of color components). Even with such limited features, the prototype proved the strength of the basic idea while presented to some potential partners. Those included a live jungle music band Human Ketchup and neurofunk DJ/producer Malcuth.

4.2 Development environments

The C++ programming language was chosen as a basic technology. Similarly to the system in the bachelor's thesis the plan was to use the graphics API OpenGL for all graphics computations, however ideally not in a such bare bones manner. A third major component required, is a tool to manage loading and most importantly decoding videos. The following toolsets and libraries were all considered.

FFmpeg is foremost a command line utility for video recording, conversion and streaming on multiple platforms. A set of development libraries is also available for the C++ language. Those low level libraries offer all the functionality of the utility, but their usage is poorly documented. [19]

OpenShot is a simple multi-platform graphic tool for video editing. Just as FFmpeg it is also available as a C++ library, however more high level and with a richer documentation. The declared multi-platform availability is however misleading since the MS Windows manual literally offers good luck [20] and list some more dependencies.

GStreamer is an open source multimedia interface. It allows for building multimedia applications using modules which it combines and groups together to achieve many different configurations. It comes from the Linux environment and shares design principles of the Gnome project. Although a number of the offered modules fit Skinny Mixer requirements rather well, the MS Windows manual is 8 years old and does not reflect today's standards. [21]

JUCE is a platform for multimedia application development in the C++ language. It offers modules to work with sound and MIDI, window creation and management, hardware communication and others. A noteworthy feature is the so called Projucer, a tool to generate build projects for multiple platforms. This was successfully tested by the author during the development of *Patterns*, a probability based MIDI sequencer in the form of a VST plugin, to deploy the plugin both on MS Windows and macOS. [22]. However the options JUCE offers in regards to video are very limited – it only loads videos in the `.avi` format, which is completely unsuitable for a larger amount of video files due to high memory requirements. There exists a module to connect JUCE and FFmpeg [23] but it is not very well implemented and also not sure to be still supported in future versions of JUCE. Also while the documentation of JUCE is mostly top quality, it s far from complete. [24]

openFrameworks is a set of C++ multimedia development tools conceptually very similar to JUCE. A main difference is its decentralisation – independent developers offer hundreds of modules (albeit of sometimes questionable quality). The official modules offer video decoding and playing in many formats using the K-Lite codec pack, OpenGL abstraction simplifying common use cases or window management and creation of simple GUIs. Some of the non official ones offer MIDI in/out software interface, DMX hardware interface, more complex GUIs and much more. Similarly to JUCE the documentation is often missing parts, due to a simple class architecture, much can be read from the class providing module interface. [25]

After a number of experiments with the listed environments, some of which more frustrating and time consuming than the others, openFrameworks were chosen as the best suitable option. The main benefits over competitors include ease of prototyping, previous experience of the author, wide array of tools fitting requirements and reasonably current and complete documentation.

As previously mentioned openFrameworks consist of addons. Some come bundled with the basic package – those are core addons and are prefixed with "of" such as ofApp, of3d or ofVideo. Many additional addons are available at http://ofxaddons.com and come prefixed with "ofx". Skinny Mixer uses three of those – ofxMidi [26] to provide platform independent MIDI software interface, ofxArgs [27] to provide convenient access to command line arguments and ofxDatGui [28], which is an extended alternative to the core addon ofGui.

4.3 Architecture

An openFrameworks application typically uses a monolithic architecture within an `ofApp` class. This approach soon proved to be too simple and had to be extended. The latest version of Skinny Mixer uses a Model View Controller approach, albeit still contained in the main `ofApp` class. Figures 4.1, 4.2 and 4.3 show all the objects that form the application.

Figure 4.1 shows objects responsible for main functionality and the view component. The class `ofApp`, as with all openFrameworks applications, constitutes the entry point and the main loop with methods such as`setup`, `update` and `exit`. It also offers callbacks that are delegated to the control component such as `keyReleased` or `newMidiMessage`. Furthermore, it contains pointers to the respective components: `gui_`[1] – view component, `show_` – controller component and `showDescription_` – model component. Lastly, it defines some private implementation specific objects and provides the MIDI interface in the `midiInputs_` object. There is also a simple `Status` object implemented following the singleton design pattern to keep track of global information.

The view component consists of a single class only, the `Gui`, which runs in a separate window to the main rendering one. It basically wraps the func-

[1]To indicate private members of an object, the "_" suffix convention has been chosen.

Figure 4.1: Objects responsible for main functionality and view component. Note: Some members of the `Gui` class members have been left out to fit on page, specifically all button callbacks and panel setup methods.

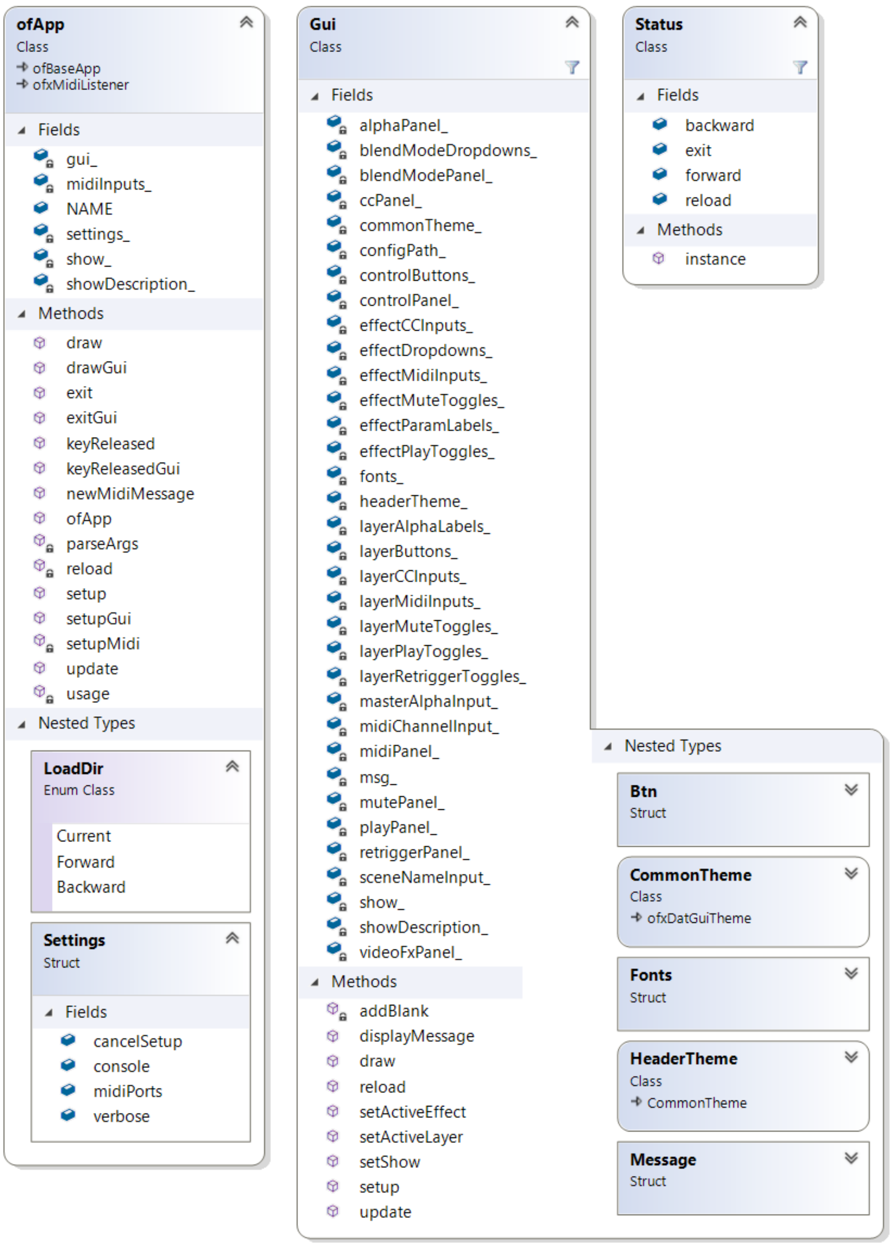


Figure 4.2: Objects of the model component.

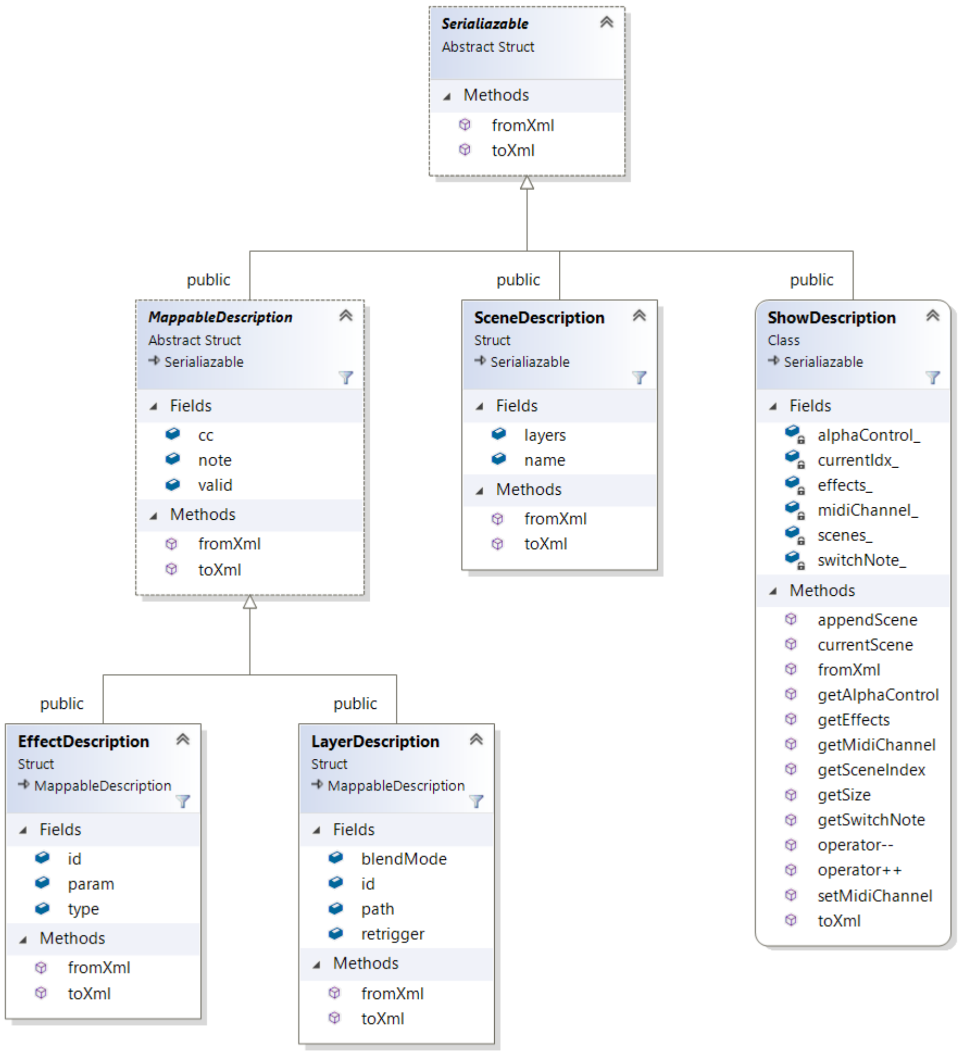
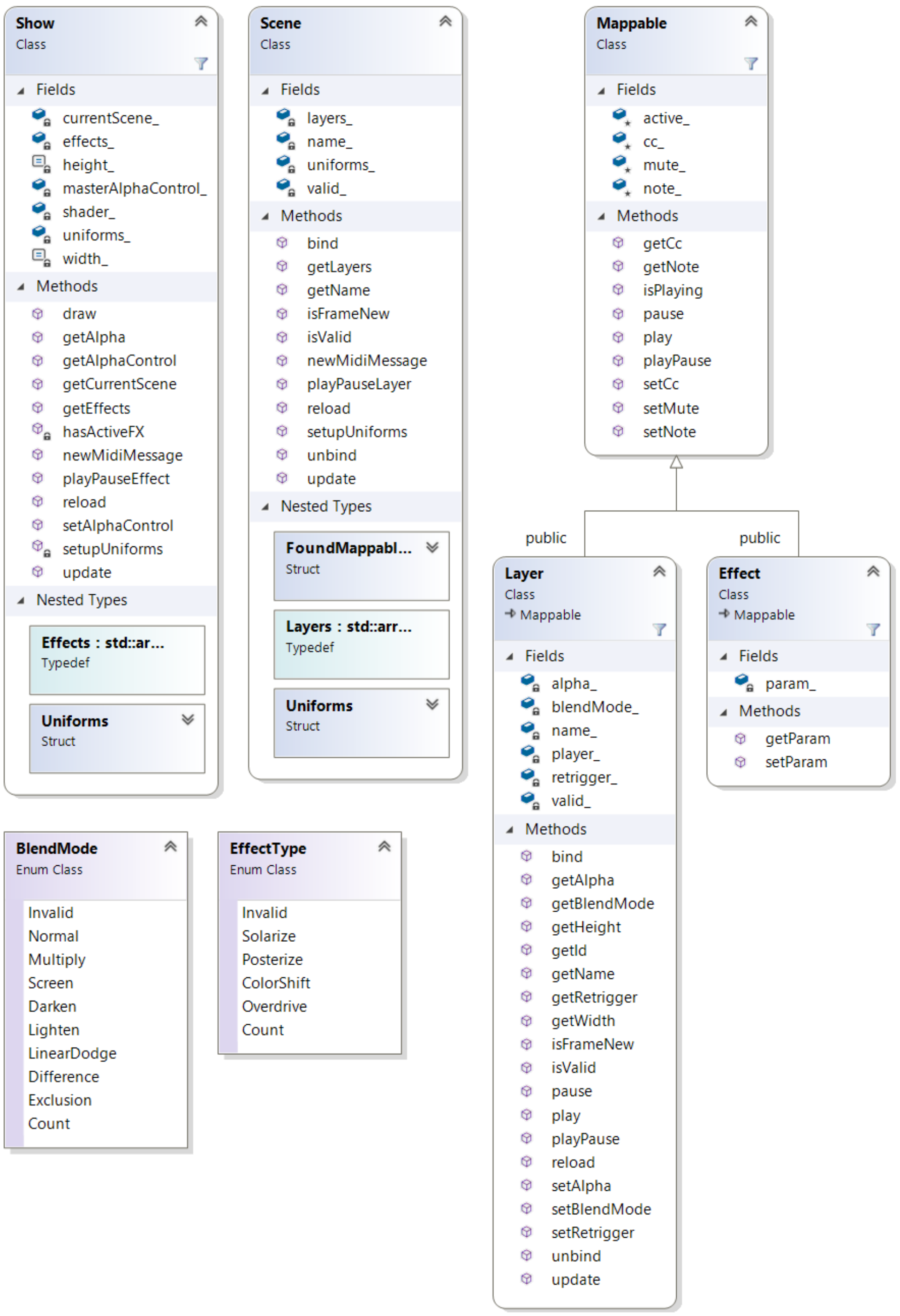


Figure 4.3: Objects of the controller component.



tionality of the ofxDatGui addon – it contains vertical panels with buttons, toggles, text inputs and other standard GUI components and provides callbacks for those, it defines themes with fonts and colors for the addon to use and also displays status messages directly in the GUI window. The `Gui` class also defines some helper objects to take responsibility for specific smaller tasks.

The model component shown on Figure 4.2 consists of a cascade of objects serializable to a XML file. There is always one `ShowDescription` object that holds a list of `SceneDescription` objects and the index of the current one. Apart from that it holds information shared among all scenes – global effects, global MIDI channel, global alpha and scene switch MIDI note. The `++` and `--` operators have been implemented to advance or decrease the current scene index.

A `SceneDescription` object only holds its name and a list of layers. The `LayerDescription` and `EffectDescription` structures inherit from a single interface `MappableDescription` (in fact all the model classes inherit from the `Serializable` interface) as they share some common members, namely the MIDI note, the MIDI CC note and the information whether the description is valid (after a `fromXml` call). The respective objects then hold further data members required for the specific layer type. An example of the serialized data can be seen in Listing 4.1.

Listing 4.1: Example data from the config XML file

```xml
<head>
    <version>0.6.2-alpha</version>
    <switchNote>36</switchNote>
    <channel>15</channel>
    <masterAlphaControl>16</masterAlphaControl>
</head>
<show>
    <effects>
        <effect>
            <id>0</id>
            <type>0</type>
            <midi>8</midi>
            <cc>8</cc>
            <param>63</param>
        </effect>
        <effect>
            <id>1</id>
            <type>1</type>
            <midi>9</midi>
            <cc>9</cc>
            <param>17</param>
        </effect>
        ...
    </effects>
    <scene>
        <name>Demo</name>
        <layer>
```

```
            <id>0</id>
            <path>White Broken Shards.mp4</path>
            <blendMode>0</blendMode>
            <alphaControl>0</alphaControl>
            <midi>0</midi>
            <retrigger>0</retrigger>
        </layer>
        <layer>
            <id>1</id>
            <path>simpleRGB.mp4</path>
            <blendMode>5</blendMode>
            <alphaControl>1</alphaControl>
            <midi>1</midi>
            <retrigger>0</retrigger>
        </layer>
        ...
    </scene>
</show>
```

Figure 4.3 shows the remaining objects present in the system which are responsible for the controller component. Those basically mirror all the objects from model, but implement their actual behaviour. Again there is `Show` which unlike `ShowDescription` does not hold all the scenes but only the current one, but it does hold all the global effect layers. It also holds the shader program (consisting of the traditional vertex & fragment shaders) and defines a structure to hold global uniform variables. The `Show` transmits MIDI messages it receives through the `newMidiMessage` method to scenes and effect layers. A `Scene` object contains video layers, transmits MIDI messages to them, checks for new frames and sends all layer related uniform variables to the GPU. To communicate with the view component it defines a `FoundMappables` object which holds all currently active layers. Similarly to the model component, the classes representing video and effect layers inherit from a common predecessor. They contain all the previously described properties while the `Layer` class (representing a video layer) also holds an `ofVideoPlayer` object available from openFrameworks. This video player plays, pauses and stops based on interpreted MIDI messages and renders potential new frames to a texture. This texture is then accessed in the fragment shader and blended with all the other layers.

To blend all the layers and apply per-pixel effects Skinny Mixer uses traditional GLSL vertex and fragment shaders. The vertex shader is simple: it only transmits the position and computes texturing coordinates based on screen size and video layer dimensions. This effectively means that all layers are transformed to fill the entire screen independently on the actual video dimensions. However it does not provide any solution for letterboxing already present in the videos. The fragment shader is a little more complex: first it accesses all the texturing units related to the video layers and blends them one by one using their respective blending modes. Then, if any video layers

$F(c,p) = \begin{cases} c_{max} - c \text{ for } \frac{c}{c_{max}} \leq \frac{p}{p_{max}} \\ c \text{ otherwise} \end{cases}$	`float solarize(float c, float p)`
	`{`
	`    return c <= p ? 1 - c : c;`
	`}`
	
	`vec3 solarize(vec3 c, float p)`
	`{`
	`    return vec3(`
	`        solarize(c.r, p),`
	`        solarize(c.g, p),`
	`        solarize(c.b, p));`
	`}`

Table 4.1: Example mathematical definition & shader code for the Solarize effect. Note that the floating point variables in the shader are already normalized to values contained in $[0, 1]$.

are active at all, it applies all the active effects, one by one. All the layer properties are given using uniform variable arrays.

4.4 Alpha releases

Skinny Mixer has been developed iteratively over almost two years prior to writing this text and while the motivation, core values and design ideas remain the same, the implementation and variety of features have changed significantly over this time period. The version number uses the semantic versioning concept, [29] however all the versions developed so far are considered alpha releases or pre-releases.

4.4.1 Version 0.1.0: First performance with Human Ketchup

After the Unity3D prototype, two more prototypes with lower importance were created. One used the JUCE platform, which is great for building audio applications such as the Patterns VST [22], but its video options are very limited. The other one was based on FFmpeg and seemed to work rather well before problems with video looping and frame skipping were encountered. Even though it was not yet explicitly mentioned, when a video on a layer reaches its end, it should play right away from the start, and this exactly was for some reason problem to achieve seamlessly with FFmpeg.

After the switch to openFrameworks the work could finally properly begin. The first public performance marked the deadline and was held on 8th June 2018 at the release party of a jungle live music band Human Ketchup. A video recording is available at [30].

The first release included the basic version of most features: seven video layers, 4 simple effects (Inverse, Color Shift as two separate effects and Color

Rounding), all the currently available blending modes, MIDI mapping n:1 (a single layer could have multiple assigned MIDI notes), and other minor features.

This release used the rather uncommon compute shader (unlike the more usual combination of vertex & fragment shaders). It allowed to skip the entire 3D paradigm and work in 2D right away, but had to use one of the textures available on the GPU as a target. It also complicated things a little on the CPU side of the code so this was later reversed. The shader is run every time an active layer has a new frame.

At this point in time Skinny Mixer did not yet offer a GUI and was a completely console based application. Instead the interface describe din subsection 4.4.1 was used. With the development of the GUI, those features have been slowly deprecated and those marked in gray are already removed in recent versions. Currently, the option `--midiports-all` is always used with plans to move the entire MIDI ports configuration to the GUI.

`-h, --help, --usage`	available commands
`--list-midiports`	list available MIDI ports
`--midiport <number>`	open MIDI port `<number>`
`--midiports-all`	open all available MIDI ports
`--config <file>`	load config from `<file>`
`--console`	log to console instead of a file
`-v, --verbose`	activate verbose communication

Table 4.2: Command line arguments

4.4.2 Version 0.2.0: Simulacrum AV at #1113

Simulacrum AV is a joint audiovisual project by DJ/producer Malcuth and Vooku (artistic alias of the author). It combines raw and loud electronic music that focuses on bass sounds and rapid drum beats (more specifically, combines genres the likes of drum & bass, dubstep, IDM, synthwave and others) with sci-fi and abstract themed visuals inspired by the Blade Runner movie or Ghost in the Shell anime. Both the music and visuals are performed live. Malcuth controls music using Ableton Live with Max MSP while Vooku oversees Skinny Mixer with input from Malcuth's Ableton and sometimes a local instance of Bitwig with additional VSTs. The performance became a platform on which to develop and test Skinny Mixer and has been slowly gaining attention. There has been a performance about two weeks after the performance with Human Ketchup using the same release – no further features were developed until later that summer due to time constraints. #1113 is a code name for a secret festival where Simulacrum AV performed later that summer and served as an opportunity to prepare the next release.

After the first performances it was clear that at least some GUI was necessary to create the entire show – writing XML by hand is hardly a good idea. Therefore, it became the top priority. Apart from the GUI which remained until now, albeit with some improvements, came automatic layer IDs (that also double as a texture unit ID), keyboard controls for the GUI window (which was later removed to allow for writing full scene names), reloading scenes both forward and back and default values for scenes so creating a new one comes faster. Since this release bug fixes became an unavoidable requirement.

4.4.3 Version 0.3.0: Simulacrum AV autumn 2018

All the most important features are now included and the time comes to develop new ideas and improve the present ones. The autumn / winter performance of Simulacrum AV at Klubovna, Prague (the same place where the system from the bachelor's thesis was first presented) brought the first experiments with MIDI CC messages. Those messages include not only a note, but also a value in the 0-127 range and on hardware devices are usually sent by rotating knobs. In Skinny Mixer, the first MIDI CCs controlled alpha opacity on video layers and later on custom effect parameters – when parametrized effects finally came. MIDI mapping 1:n was removed for multiple reasons – its use cases were questionable and there were no real plans to include it as part of the GUI and without proper setup it remained a rather unusable feature anyway. The GUI was overall improved and additional layer properties such as mute or re-trigger (as described in section 3.3) were added as well as periodic FPS logging to get some basic idea about the real performance.

4.4.4 Version 0.6.1: Simulacrum AV at #1114

Multiple performances happened with new versions between the autumn edition and the performance at the secret festival renamed to #1114 for the next occurrence. Due to time constraints those were never released online however the development continued. The largest change was brought to the GPU side of code: the compute shader was exchanged with a combination of vertex and fragment shaders. This allowed for one more video layer since no texture unit was required for the output texture – the output is rendered straight to the back buffer. With the change of shaders the effect layers were also revised. Originally custom for each scene they were made "global" – the effect layers are now shared among the entire show. All the effects were parametrized except for Inversion (which was later changed to parametrized Solarization). This allowed Color Shifts to be joined into one effect and Color Rounding to be turned into Posterization. The Overdrive effect was also added. Finally, the effect layers were made independent and configurable unlike before when they were in set, unchangeable order. This means that now it is possible to have two layers with the same effect, but a different parameter.

This release also brought saving to the original configuration file (with “save” & “save as” options) and MIDI channel choice to filter unwanted messages.

During this longer period a stale branch was developed and eventually abandoned. The goal was to create a list with random access to all the scenes in a show both to provide an overview and more control. It turned out that the ofxDatGui addon has problems with multiple window setup – it requires a mouse click to gain focus and another click to activate the button. However, if the panel is already in focus, only one click is required. This is not a major issue when selecting a video, but since reloading a scene takes considerable amount of time, mis-clicking proved surprisingly costly. Therefore the entire feature was abandoned with two possible future outcomes: either this issue is resolved in ofxDatGui (with a pull request from the author possibly) or the entire addon is exchanged to something else.

4.4.5 Version 0.7.0: Rebranding to Skinny Mixer

The current release brought mostly bug fixes and improvements to the code architecture and code structure, which became somewhat cluttered with the dynamic changes. The oldest original effect – Inverse – was finally removed and replaced by the parametric Solarize. Long video layer names are now shortened to fit the button width and a mistake in the Exclusion blending mode was fixed.

The biggest change was the rebranding to Skinny Mixer. The original name was *TVAEM – Triggered Video and Effects Mixer* which is hard to both pronounce and remember whereas *Skinny Mixer* is much sleeker. The name is inspired by a cocktail popular in Prague techno clubs of the late 2010s.

Chapter 5

Evaluation

5.1 Performance evaluation

As specified in chapter 3, the following performance requirements consider a standard output resolution of 1920 x 1080 pixels (Full HD).

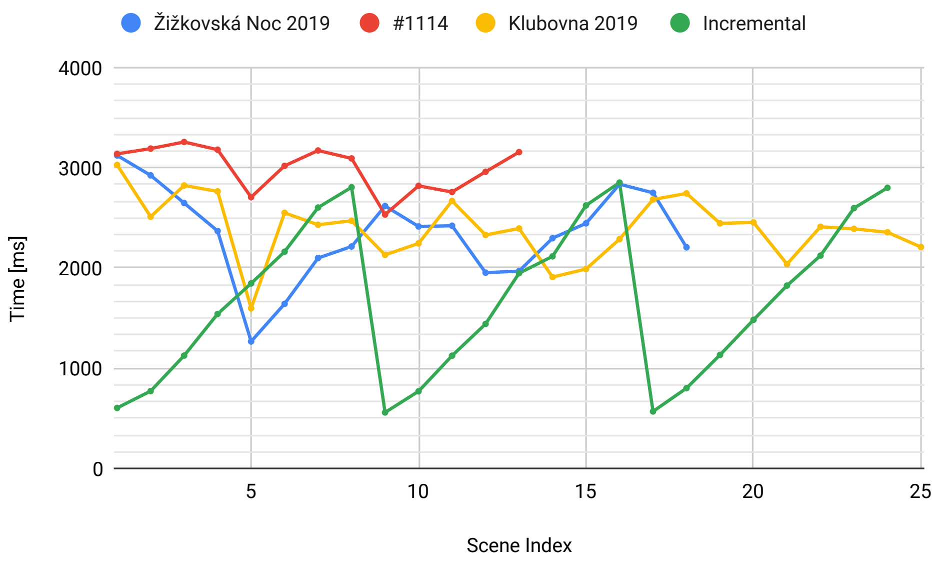
5.1.1 Load times

Loading a show can be split into two parts. Loading the show description is fast – only a deserialization of an XML file, which happens in a couple of milliseconds (see Table 5.1) – an insignificant amount of time from a human perspective. After loading the show however, the first scene is loaded and that actually proved to be an issue.

5.1.2 Switch times

Switching to another scene requires decoding all the videos it contains which proved to take considerable amount of time as shown on Figure 5.1 and Table 5.1. Four measurements were taken: the first three are based on shows from real performances with significant differences. The one at #1114 used the latest version of Skinny Mixer which allowed for 8 video layers instead of the previous 7 in the other two shows. This resulted in overall longer switch times as seen both on the chart and in the table. Each show also consisted of a different number of scenes due to different set times and dramaturgies, this resulted in different amount of data points. There is a major drop in all three shows at scene 5 – this is because some assets from the performances are sometimes reused and all three include the song Poison by The Prodigy with matching visuals at this point. The #1114 show included one more video layer and a long one on top of that which increased the load time of the scene compared to the other two which only differed in about 400 ms.

Figure 5.1: Scene switch times. Blue, red and yellow are shows from actual performances whereas green is a custom show designed to test the dependency between switch times and the number of assigned video layers.



The last show, called “Incremental”, was created artificially to test the hypothesis that switch time depends on the number of videos. The show contained 8 scenes with increasing number of assigned video layers. The videos were chosen randomly from the video library assembled for Simulacrum AV. The show was reloaded three times in total. The results seem to confirm this hypothesis with the amount of time needed to switch increasing almost linearly with the number of assigned video layers. Furthermore, it shows slight deviations in the pattern, hinting at other variables having effect on the final switch time.

Figure 5.2 shows loading times of single videos while the Incremental show was assembled video by video. Multiplying the average loading time of 375 ms by 8 – the number of layers – gives 3000 ms, which corresponds to the average scene switch time of the #1114 show.

All in all this is too long. The switch is by no means instant and 2 to 3 seconds is a long time in the fast paced world of today’s electronic music – 3 seconds mean almost 2 bars at 140 BPM. A workaround used in the past performances was to switch scenes using a total fadeout during the quieter atmospheric parts between songs, which even makes sense from the dramaturgical point of view.

Figure 5.2: The loading times of videos while building the Incremental show. Average loading time of a single video is 375 ms.

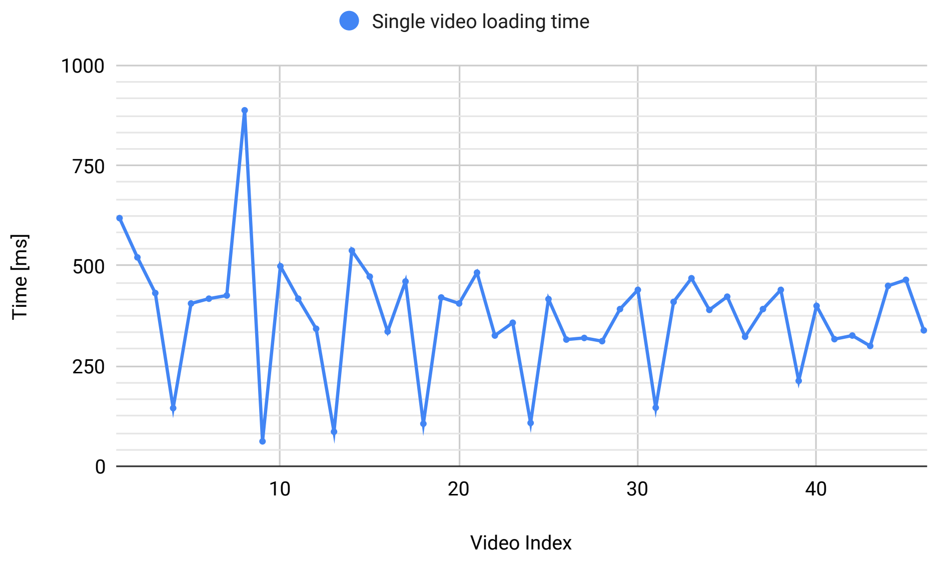


Table 5.1: Average scene switch times

Show	Initial load time [ms]	Average switch time [ms]
Žižkovská Noc 2019	5	2344
#1114	5	2999
Klubovna 2019	7	2394
Incremental	5	1676

5.1.3 Latency

The latency was measured the same way as in the bachelor's thesis. A high frame rate camera was aimed at an area including both the screen and a hardware MIDI device. The camera was set to 120 FPS Full HD recording, the screen was using 60 Hz refresh rate and the device was the same one as used during the later live performances – Arturia Beatstep, which replaced the previously used Akai LPD8 Wireless. Both of those MIDI devices include light up pads which helps identify the exact time of impact. The camera runs at double the frequency of the screen so according to the Nyquist-Shannon sampling theorem it should be fast enough to reliably measure its response. The base system runs on an Intel Core i7-7500U CPU at 2.70 GHz with 32 GB RAM, nVidia GeForce 940MX GPU with 2 GB VRAM and a 256 GB SSD, which is currently medium to low spec laptop, however it is the one used for the actual performances.

Table 5.2: Hardware MIDI device to screen latency measured at 120 FPS

Test	Frames										Avg [frames]	Avg [ms]
1.	10	9	9	10	10	11	8	9	9	8	9.3	77.5
2.	7	11	7	8	9	10	11	9	8	7	8.7	72.5
3.	10	16	13	14	14	15	16	17	13	14	14.2	118.3
4.	14	13	12	11	12	10	14	13	9	11	11.9	99.2

Using the recorded video the latency between pressing a button and the respective response on screen was measured in frames as shown in Table 5.2. As specified in chapter 3, the target was $1/30$ s, or in the units used during evaluation, about 33.3 ms or 4 frames at 120 FPS. Under these conditions the following four tests were performed.

1. Activating and deactivating a single video layer
2. Activating and deactivating a single effect layer over a single active video layer
3. Activating and deactivating a single video layer over 7 active video layers
4. Activating and deactivating a single effect layer over 8 active video layers

While the latency is comparable to that of the bachelor's thesis, the target here was set higher. Even in the best scenarios the latency was at least double the target. Combined with the potential additional latency of a projector with a possible HDMI splitter along the way for better monitoring, the latency definitely requires further improvement. As indicated by the lower response time when activating an effect, the problem lies with the video layers and probably with the continual sequential loading of textures to the GPU.

5.1.4 Fluidity

FPS was measured during four live performances with the results in Table 5.3. The lowest result of 24.52 FPS came after the eight video layer was included at a show which utilised this new feature more than the other performances. Due to the nature of the event, the entire performance was less rhythmic and focused on more atmospheric sounds which inspired the visuals to include fewer dramatic cuts and effect switching and more video blending. The reasoning behind the lower FPS is therefore the cost of multiple texture offload to the GPU.

Note: The sampling rate here is somewhat questionable. To reliably measure if the target FPS of 30 is actually achieved, we should sample at double the frequency. At that rate however logging constantly to a file or just to the console might slow down the application on its own and not only influence

Table 5.3: Average FPS measured during live performances using a sampling rate of 1 s.

Performance	Average FPS
Klubovna autumn 2018	27.34
Žižkovská Noc 2019	26.77
#1114	24.52
Klubovna 2019	26.13

the measuring process, but the performance itself. Also the amount of data grows considerably for a performance that takes an hour to two hours. This could be improved by measuring not during a public performance but while testing in private where the quality of the visuals does not matter. Nevertheless the possible impact on the measuring process itself led to the conclusion that measuring live data is better.

5.2 Usability testing

The graphical user interface was evaluated using two methods from the area of usability engineering: Nielsen's heuristics and users completing scenarios prepared in advance. The command line interface was not evaluated during this phase as it is considered obsolete and is going to be removed completely once all the remaining functionalities have their respective counterparts in the GUI.

5.2.1 Heuristics

1. **Visibility of system status** The currently active or muted layers are indicated using toggles to the left of the layer name. If a layer is not loaded or failed to load, it is indicated on the layer button. The FPS, the MIDI mappings and layer properties are showing at all times. When a show is saved or loaded, a short message displays briefly on the bottom left. A similar message shows when a scene is being switched but not every time. If the MIDI ports setup fails, there is no information in the GUI. Generally, many error messages are sent to the console, but not displayed in the GUI. A better error reporting system is required.

2. **Match between system and the real world** The system uses partly custom language and partly general terms. The custom language used includes the terms *show, scene* and to some extent *layer*. The MIDI terminology should be familiar to users attempting to create an audiovisual show, just as the basic blending modes which are identical to the ones in common graphics software (Adobe Photoshop and the likes). The effect names try to be as descriptive as possible and their workings should be

clear enough when the user tries them out. The layer order is inverted and therefore confusing – what ends up on top shows on the bottom and vice versa. The ranges of numerical properties selected, i.e. 0–127 are debatable, maybe 0–1 or 0–100 would feel more familiar to the users.

3. **User control and freedom** There is no undo or redo. There is also no way to remove a scene from a show or to add a new scene anywhere but at the end of the show. There are buttons next to each layer to stop playback or straight up mute it if it was triggered by mistake. There is no way to change alpha or custom effect parameter without a connected and mapped MIDI device.

4. **Consistency and standards** The video and show loading dialogues are provided by the operating system and are therefore familiar to the user. The FPS displays in a non-editable text box which looks just as all the editable ones. The labels on the bottom of the control panel on the left are not clickable yet they look just like the buttons just above them. The same applies to the headers of the layers table.

5. **Error prevention** Assignable text fields expecting MIDI notes do not allow alphabetical characters, however they do allow some special characters and out-of-range values. The MIDI channel text field is the only one which at least corrects out-of-range values to the limits. Show loading and saving and video loading provide no limits on file types.

6. **Recognition rather than recall** The GUI shows all its elements clearly and all are present at all times, therefore easily recognizable. The elements still missing in the GUI are a different story. While the console interface offers the standard help switches `-h`, `--help` and `--usage`, the console interface itself is barely used and therefore rather obscure. Yet there is no other way to setup logging options or MIDI ports.

7. **Flexibility and efficiency of use** There are no keyboard shortcuts. The save button automatically saves to the most recently open file and if there is none, it prompts the user to choose one. The user is free to tailor their own MIDI mappings and it is even possible to assign a single MIDI message to multiple layers.

8. **Aesthetic and minimalist design** The GUI is purely functional with close to none decorative elements. The colors are dark with contrasting typesetting, suitable for stage use. There is no light theme. The font selection is mostly functional with a hint of attempted styling with the italics.

9. **Help users recognize, diagnose, and recover from errors** If a video layer fails to load it is clear enough but the cause remains unknown.

Most errors are only shown in the console but especially with the verbose mode the information content is rich.

10. **Help and documentation** There is basic help in the console interface. There is no help button, there is no documentation.

Overall the GUI fails in most of the heuristical requirements. It is missing major features and some others are incomplete. The overall design is poor and leaves place for many improvements. However, as this is still considered a pre-release, alpha version, such a state of things is forgivable given that it will hopefully improve or be entirely replaced.

5.2.2 User scenarios

All of the scenarios start with the app running and connected to a hardware MIDI device. The users were explained the basic usage of MIDI devices and have a sheet available showing the mappings of the device. The users were also explained the show – scene – layer paradigm. The windows of the app are on separate screens to show all the information available at once. An assistant qualified to answer all questions is also present during the testing. After completing all the scenarios, a user should be familiar with all of the elements of the GUI. Following are the scenarios and users' comments on them.

1. **Basic setup** Start with a an empty show with only the default empty scene. Load some videos. Play them. Change their MIDI mapping and play them again.

2. **Layer properties** Experiment with the other properties of a video layer. Change blending modes, change the alpha during playback, change alpha MIDI mapping. Mute a video, make it play without input. Setup a video to play from the start every time it's triggered.

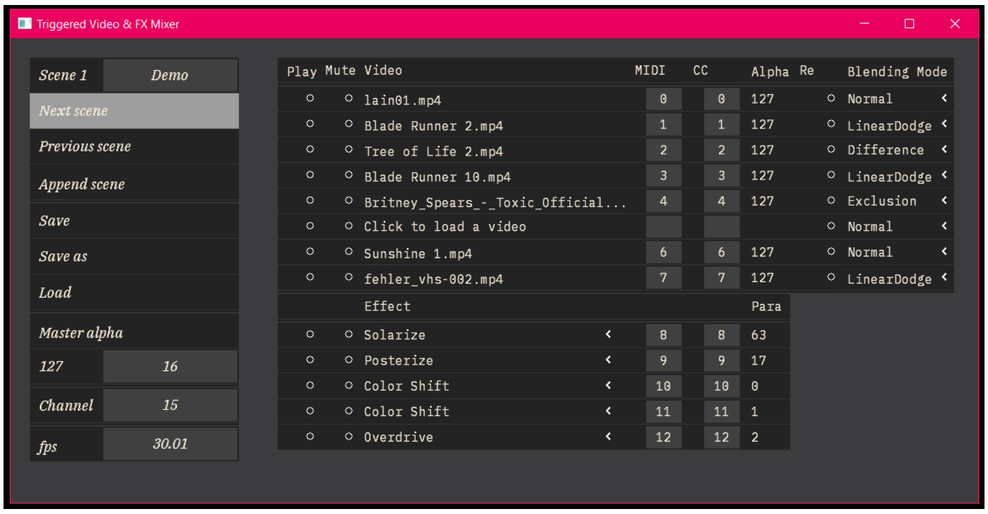
3. **Effects** Apply an effect to a playing video. Apply multiple effects on a playing video. Change the MIDI trigger of an effect.

4. **Advanced effects** Change a parameter of an effect. Change the parameter's MIDI mapping. Have the same effect twice with different parameters.

5. **Control** Add another scene to the show and set up its layers. Change the scene's name. Go back to the previous scene and change its name. Save the show. Close the app.

6. **Advanced control** Load your show. Change the master alpha. Change the master alpha mapping. Change the MIDI channel. Add two more scenes. Go to scene number 2. Go to the last scene. Save the show.

Figure 5.3: The current GUI with the control panel on the left and the layers on the right. Note the unassigned video layer and the redundant Color Shift effect layers with a different parameter.


Triggered Video & FX Mixer

Scene 1	*Demo*
Next scene	
Previous scene	
Append scene	
Save	
Save as	
Load	
Master alpha	
127	*16*
Channel	*15*
fps	*30.01*

Play	Mute	Video	MIDI	CC	Alpha	Re	Blending Mode
○	○	lain01.mp4	0	0	127	○	Normal ‹
○	○	Blade Runner 2.mp4	1	1	127	○	LinearDodge ‹
○	○	Tree of Life 2.mp4	2	2	127	○	Difference ‹
○	○	Blade Runner 10.mp4	3	3	127	○	LinearDodge ‹
○	○	Britney_Spears_-_Toxic_Official...	4	4	127	○	Exclusion ‹
○	○	Click to load a video				○	Normal ‹
○	○	Sunshine 1.mp4	6	6	127	○	Normal ‹
○	○	fehler_vhs-002.mp4	7	7	127	○	LinearDodge ‹

		Effect				Para
○	○	Solarize	‹	8	8	63
○	○	Posterize	‹	9	9	17
○	○	Color Shift	‹	10	10	0
○	○	Color Shift	‹	11	11	1
○	○	Overdrive	‹	12	12	2

5.2.3 User scenarios results

The scenarios were run on multiple occasions with 13 participants in total, each participant individually. When a participant got stuck the assistant showed them how to complete the next step and continue with the scenario. Each run took about 20 to 30 minutes. Below are the most common problems the participants encountered. Successful steps are not mentioned.

1. **Basic setup** A common confusion was with the *Load* button in the control panel – *what* does it load? What kind of file should I select? This could easily be helped by renaming the button to *Load Show* and adding some limits into the load dialogue.

   Most participants tried to work around a specific issue of the ofxDatGui addon responsible for the GUI by double clicking buttons. The issue is that the GUI consists of vertical panels that have trouble with the multiple window setup and the first click into a panel puts it into focus and only the second click activates a button. When you click into the same panel again however the focus stays and the button activates right away. This is very confusing and especially on the *Next scene* button a little dangerous since switching to the next scene takes 2 to 3 seconds as shown earlier.

   When setting up the basic layer properties some users missed right clicking or a drag-and-drop feature.

2. **Layer properties** Blending modes proved to be a major source of confusion to more than half of the participants – all those unfamiliar with graphical software. The blending mode on the first layer does nothing – why is it even there? The Multiply blending mod makes the layer seemingly disappear as it only works over another layer. The Normal blending mode makes all the layers below disappear – a better name would be Override or perhaps Overwrite.

   Another issue of ofxDatGui is the basic implementation of drop down lists. They do not auto close after losing focus and they offer no scroll wheel functionality. The drop downs on the bottom of the interface do not even fit into the window! Only one user successfully attempted to resize the window using the common Windows way.

   All users tried to edit alpha and effect parameter values directly which the GUI does not offer.

   Only three users managed to correctly use the re-trigger toggle and all confessed it was only because it was the last video layer property they have not used yet. The *Re* shortcut lacks any meaning and is misaligned to the toggle making it look strangely disconnected from the entire column.

   There were also some suggestions: The play toggle is too small, the re-trigger toggle should join the other toggles on the left and double tapping a hardware pad should make a video player keep playing. While the last suggestion is incompatible with the basic note on / note off idea, mapping the play and other toggles to hardware buttons is a certain future feature.

3. **Effects** After learning the basic workings of video layers no participants had any trouble with the basic effects.

4. **Advanced effects** During setup of effect properties and especially the double effect step the problems with drop down lists showed in full swing. Setting the bottom effect to anything but Solarize or Posterize is impossible without resizing the window.

   Most users took a while to understand the Posterize effect as it does not really show any major changes until it is set to very low values. Remapping the range should be considered.

   A single user managed to edit both the MIDI and CC mapping of a layer at the same time. This probably has something to do with the library focus issue. Interestingly after a brief confusion the user started liking it and suggested it as a feature.

5. **Control** Right at the top of the control panel there is a text box for the scene name, which defaultly reads “New scene”. Few rows below

that is a *Append scene* button which actually adds a new scene to the end of the show but provides no indication of the fact. Those factors combined resulted in ten participants having major trouble adding a new scene, clicking desperately on the text box and ignoring the unusual word “Append”. Luckily, some improvements to this problem are at hand: change “New scene” to “Enter name” and add indication that a scene was added to the end of the show. The *Append scene* button itself is a tougher problem. Simply renaming it might help as well as jumping to the new scene right away.

While changing the scene name four users kept clicking the *Next scene* button to confirm the name change. Apparently, the interface is too minimal, users do require a submit button and submitting data into a text box by pressing the Enter key does not come naturally.

Since the app does not automatically append a “.xml” ending to the show configuration file while saving, all of the users saved the file without it. While it is not a factual problem as the file is still valid XML, it is not automatically assigned an icon by the operating system and looks out of place.

6. **Advanced control** Apart from the three users that commonly use MIDI and MIDI devices, everyone had trouble with the master alpha and MIDI channel setting. The interface groups those values (and the FPS display as well) into a confusing array of text boxes and labels with inconsistent behaviour and no status indication. With only surface level understanding of MIDI the participants were rightfully misguided.

   Eight participants made use of the *Save* & *Save as* distinction or maybe rather just preferred the *Save* button which is placed higher and appears more simple. When no show configuration file is loaded, both buttons work as *Save as*, prompting a user to select a file or make a new one. After that (or after successfully loading a show configuration file) the *Save* button saves to the file without opening a dialogue while *Save as* prompts the user as before.

   One of the participants suggested a global mute toggle based off the feeling that rotating the master alpha knob is too unwieldy.

Overall running the scenarios with actual users proved to bring a lot of valuable information, even more than the heuristics. Some problems were identified using both methods which hints to their seriousness. All in all the interface needs a lot of improvement. Some problems were known prior to the testing such as the missing features contained only in the obsolete console interface, some discovered ones can be fixed easily and some call for a major redesign, maybe even switching to a different library or at least working on the issues of the current one.

While the control panel was a total failure, the basic layout of the layers and their relation to the hardware mapping proved to be easily understandable, with only minor tweaks required. Since the layers are used much more often, this turned out better than it could.

5.3 Peer reviews

Three experts were asked to write a short review after witnessing the performance to evaluate the system from artistic perspective. These are David Čajčík, music journalist, juror in Vinyla awards and music programmer in Roxy Prague, Bc. Josef Hanzlíček a.k.a Hepex, DJ & music producer, and MgA. Jiří Nižník a.k.a. Malcuth, DJ, music producer, performer and the person responsible for the musical side of Simulacrum AV.

5.3.1 David Čajčík

Simulacrum AV presents a fresh, technologically advanced, sophisticated and, essential to mention, fun approach to presenting electronic music while immersing the audience into an audiovisual world. Whereas the traditional methods of visualising music in live setting operate mainly within traditional VJing programs utilising ad-hoc and "live" controlling of visual schemes in direct response to presented music heard by the VJ, Vooku (the author) utilizes much more sophisticated pre-programmed settings. In that matter Simulacrum AV presents a rather unique approach to music performance, where the relationship between the audio and the visual is united to a much greater extent with longer-term preparations taking place in advance of the performance imposing numerous requirements essentially leading to a more high-quality artistic output.

There is a distinct variety of visual atmospheres presented and modeled by Vooku ranging from video-games cuts, abstract graphics and even science fiction shots combined with promo videos for today's security systems showing an uncanny similarity. Musically Malcuth successfully merges drum & bass, IDM, bass music and other electronic music forms into a coherent flow of both break beats and straight beats. This easily allows the visuals and Skinny Mixer to expose multiple feelings and vibes. The relevancy of this musical approach is well documented by the recent success of Floating Points, British producer composing his live sets with the same feeling by combining several influences and other prolific DJs such as JASSS, Mall Grab or many others. The recent advances in the entertainment industry more or less dictate that the concert experience is complex and absorbing. In that matter Simulacrum AV is ambitiously setting a path in the Czech music scene which is not commonly used and maybe even setting a trend for the future.

5.3.2 Josef Hanzlíček

The entire Simulacrum AV performance was controlled by two people, one performing as a DJ while the other one as a VJ. On first glance they seemed to work independently, however after a while clear rhythmic synchronization between the sound and the playing music can be perceived. Just as the music the visual consists of several layers. The music consisted of pieces mostly from the genres of the bass music region with the main focus on break beat and neurofunk (a drum & bass subgenre), switching at times to more experimental sounds such as ambient or hardly characterisable at all. Two elements distinguish the music from a standard DJ set. Repeatedly two and more tracks were combined together in one moment and thanks to the richness of the selected genres in the set the tempo changed unusually often as well.

This musical variation was accompanied by a graphical variation just as rich – or even richer. The most characteristic element of the performance was the visual base for the musical part, projected behind the performers. I am unsure to what extent were the rhythmical sequences preprogrammed or improvised, but the impact of this synchronization was much larger than the common visuals which can be seen for example during DJ sets at large festivals. The sonic atmosphere also reflected the mood of the visuals. During darker passages with "liquid" bass sounds the projection was dominated by a mix of post-apocalyptic imagery mixed with brief flashes of a logo or shots clearly from the horror, thriller or sci-fi genres from the 80s. I vividly remember a live visual remix of original material with the introduction scene of the 1982 movie Blade Runner, the chimneys of dystopian Los Angeles hurling fire and smoke in sync with the beats, so unlike the original soundtrack produced by Vangelis. During the faster sections the visual took on a straight up abstract character rather than the previous atmospheric one, but the rhythmical patterns kept in sync with the music. The single exception to this was the introduction filled with oscillating bass drones which was accompanied by a slow shot of an island surrounded by calm sea, a strong contrast unlike all the other moments full of dark imagery.

The entire audiovisual work constituted an artistically compact continuum with "audiovisual" being the key term to use to describe this two hour long performance. I am convinced that one of the elements without the other would hardly pass for a convincing work of art. The synchronization was rather impressive from the technical standpoint. Even with multiple connected devices the rhythmical synchronization stands across multiple tempo changes. To borrow some vocabulary from the world of literature – the performance was much more lyrical than epical. In the end it seems that the goal was not to offer the audience a particular cohesive story rather than an emotionally strong message and a clear demonstration of original technical solutions to combine audiovisual elements.

5.3.3 Jiří Nižník

As a composer, DJ, producer and performing artist I've been searching for a tool that would allow my visually evocative musical material to come to life. This software has the means to work with rhythm and image in a very precise and creative way. The best feature is of course it's realtime capability and ability to adapt and even improvise on the go. This tool has been at the core of the "Simulacrum AV" audiovisual live performance where together with the code's author, we strive to create an engaging and thought-provoking visualisation to the underlying sonic barrage.

An interesting aspect of this cooperation is how I as a musician am now able to visualise certain musical elements, instruments, rhythmical patterns, interpunction and more in relation to the software's possibilities. During the creative process of preparing for a performance I'm faced with many aesthetic and dramaturgical choices I can make. After the musical performance and all it's cues are locked I begin to literally "temp" score the visuals[2] using different MIDI notes, triggering lanes of visual information (usually a short loop). These lanes I see as a possibility to represent and accentuate different musical elements. A kick, a snare, vocal phrase, high hat rhythm, bass stab, guitar riff, etc. on one side and chosen visual clip (a colourful animation, black and white dance scene, a sci-fi spaceship, explosion, text etc.) on the other, each in its respectful MIDI lane. A great asset is working within "scenes" with potentially unlimited possibilities for visual dramaturgy and even storytelling. This is but just one way of utilization – a system used for a particular project.

It is not just a versatile tool, but a real instrument which can be learned and improved depending on the user. A modular instrument which can be modified if needed. If we also include the possibility of working with built in effects, lights and procedural animation, the creativity expands even more. The live MIDI controller and parameter mapping allow for a proper live control of incoming and outgoing data with an artist's feel. With all this in mind I can approach my performances either with meticulous preparation or just with a basic concept while the visual artist operating the software organically reacts.

[2]Temp score or temporary score is an expression from film music meaning a rough musical dramaturgy which can (but doesn't necessarily have to) be replaced by composed music from a hired composer.

Conclusion

Goals completion

1. **Research state-of-the-art solutions** Some of the many options to create an audiovisual performance were explored in chapter 1. Especially in the area of software the focus was on the more experimental and innovative solutions. The changing cultural environment was briefly hinted at when mentioning some of today's international events focusing on modern electronic music, multimedia and the relation of art technology.

2. **State benefits & investigate drawbacks of the bachelor's thesis** To the analysis of the bachelor's thesis was dedicated the entire chapter 2. Among the benefits was the artistic success with the *Zmutovaná Veverka* song being a fan favourite, the unique visuals and the fast response time averaging at 83 to 33 ms. The drawbacks included the cumbersome development requiring a new C++ class for each song and the development of brand new visual elements.

3. **Design & implement an advanced system** The specification and requirements for Skinny Mixer were described in chapter 3 and chapter 4 continued on the implementation itself. The development process consisted of iterations, each corresponding to a new live performance to test the new features.

4. **Evaluate the system** Four approaches to evaluation were described in chapter 5. The performance was measured in terms of load times, scene switch times, latency and fluidity. The GUI was examined using Nielsen's usability heuristics and with 13 users completing the 6 predefined user scenarios. Lastly the artistic value was considered by three experts. The performance barely met the specified requirements and the GUI is in need of major improvements, however the peer reviews

agreed that despite the technical issues the audiovisual performance is ambitious, emotionally strong and versatile.

Future plans

Skinny Mixer is very much a project in development. Just a week after finishing this text the next Simulacrum AV is planned on the main stage of Cross Club Prague as part of Future Gate science fiction film festival, with further yet unannounced performances in planning. This means that the motivation to improve and develop the tool remains high.

The performance issues were somewhat known before writing this text and proper measurement helped to confirm them. A big improvement could be obtained by switching to multi threaded approach in two areas. To begin with, some of the messages the GUI is not showing often (but not always) remain hidden because of race conditions as the main app is busy decoding videos. Secondly, OpenGL allows for asynchronous texture loading to the GPU using the `ARB_pixel_buffer_object` extension as shown in [31]. This might not be fully supported by openFrameworks and therefore could require a considerable amount of work – or maybe there is an addon for that.

While some GUI issues were also known beforehand, using proper methods to evaluate it proved invaluable. The simple fixes are expected to be done soon with the more complex ones when they eventually become top priority.

Discussions with the supervisor also brought some inspiration for future modifications. For a while now there has been and idea to create a VST version of the entire app. That would allow not just for using JUCE, which is a brilliant library for VSTs and user interfaces, but also being able to map all parameters to MIDI as that is a common and important feature in DAWs. A problem remained however – what about running an OpenGL context heavy on data transfer out of a DAW? Even without further investigation this seemed like a bad idea. A solution would be to create just a VST interface and based on a client-server architecture utilize a local network to run the video playback independently. This could even be done on separate machines to maximize resources – one machine for control, one for output.

All in all Skinny Mixer and Simulacrum AV have just began to gain traction and will hopefully continue to be a source of challenges to the authors – Malcuth & Vooku – and inspiration to the audiences.

Bibliography

[1] Petrov, V. *Nástroj pro vizualizaci datového toku formátu MIDI pro živá vystoupení*. Bachelor's thesis, Czech Technical University in Prague, Faculty of Information Technology, 2016.

[2] Elation Professional, Los Angeles. *DMX 101: a DMX 512 Handbook*. 2008. Available from: https://cdb.s3.amazonaws.com/ItemRelatedFiles/10191/dmx-101-handbook.pdf

[3] Mydy Rabycad – Where's the fokin' backstage? (Live at Colours of Ostrava). YouTube, 2018. Available from: https://www.youtube.com/watch?v=3Ka6OBZ0qKs

[4] Noisia – "Outer Edges" in 360°– Rampage 2017. YouTube, 2019. Available from: https://www.youtube.com/watch?v=P6rb7Hq_Q0Q

[5] Sinjin Hawke & Zora Jones – AV Live at Sonár Barcelona 2018. YouTube, 2018. Available from: https://www.youtube.com/watch?v=Ul5rTdYo628

[6] McLean, A. TidalCycles. Website, 2020. Available from: https://tidalcycles.org

[7] Roberts, C. Gibber. Website, 2020. Available from: https://gibber.cc

[8] Casa, D. D.; John, G. LiveCodeLab. Website, 2020. Available from: https://livecodelab.net

[9] Dorado, D. Livecoding ∼Ar en Atom 4/8/18. YouTube, 2018. Available from: https://www.youtube.com/watch?v=1OkqxPfKsh8

[10] Lunchmeat Festival. Website, 2019. Available from: http://www.lunchmeat.cz

[11] Shape Platform. Website, 2020. Available from: https://www.shapeplatform.eu

[12] Bandura, L. Aid Kid & Pavel Karafiát performing at Lunchmeat Festival 2019. 2019. Available from: https://www.fullmoonzine.cz/lunchmeat-festival-5-10-2019-narodni-galerie-praha

[13] Live Performers Meeting. Website, 2020. Available from: https://liveperformersmeeting.net

[14] Generate! Festival für elektronische Künste. Website, 2020. Available from: https://generatefestival.de

[15] Signal Festival. Website, 2020. Available from: https://www.signalfestival.com

[16] Petrov, V. Rejfpank Live @ Klubovna 9. 5. 2016. YouTube, 2016. Available from: https://www.youtube.com/watch?v=HzFjP2zBFHc

[17] Adobe Systems Inc. *PDF Reference: PDF Blend Modes: Addendum.* Fifth edition, 2006. Available from: https://www.adobe.com/content/dam/acom/en/devnet/pdf/pdfs/pdf_reference_archives/blend_modes.pdf

[18] Petrov, V. Skinny Mixer. GitHub, 2020. Available from: https://github.com/vooku/skinny

[19] FFmpeg. *FFmpeg Documentation.* 2018. Available from: https://ffmpeg.org/doxygen/trunk/index.html

[20] OpenShot Studios, LLC. *Openshot Installation Guide.* 2016. Available from: https://openshot.org/files/libopenshot/InstallationGuide.pdf

[21] Freedesktop. *GStreamer Documentation.* 2018. Available from: https://gstreamer.freedesktop.org/documentation

[22] Petrov, V. *Patterns: a Probability-based Drum Sequencer.* Ljubljana: Faculty of Computer and Information Science, University of Ljubljana, 2018.

[23] Duracz, J. JUCE-FFmpeg. GitHub, 2017. Available from: https://github.com/c41x/JUCE-FFmpeg

[24] JUCE. *Juce Class Index.* 2019. Available from: https://docs.juce.com/master/index.html

[25] openFrameworks. *openFrameworks Reference.* 2019. Available from: https://openframeworks.cc/documentation

[26] Wilcox, D. ofxMidi. GitHub, 2019. Available from: https://github.com/danomatika/ofxMidi

[27] Mullany, C. ofxArgs. GitHub, 2017. Available from: https://github.com/outsidecontext/ofxArgs

[28] Braitsch, S. ofxDatGui. GitHub, 2018. Available from: https://github.com/braitsch/ofxDatGui

[29] Preston-Werner, T. *Semantic Versioning 2.0.0*. 2019. Available from: https://semver.org

[30] Cross Club – Human Ketchup, Anniversary & Album Release Party. YouTube, 2018. Available from: https://www.youtube.com/watch?v=1jqAS0Mvip8

[31] Ahn, S. H. OpenGL Pixel Buffer Object (PBO). Website, 2018. Available from: https://www.songho.ca/opengl/gl_pbo.html

[32] Snoman, R. *Dance Music Manual*. UK: Focal Press, second edition, 2008, ISBN 0240521072.

[33] Nielsen, J. Enhancing the explanatory power of usability heuristics. *Proc. ACM CHI'94 Conf*, 1994: pp. 152–158.

[34] MIDI Manufacturers Association. *The MIDI 1.0 Specification*. Third edition, 1996. Available from: https://www.midi.org/specifications/item/general-midi

[35] Unity Technologies. *Unity User Manual*. 2019. Available from: https://docs.unity3d.com/Manual/index.html

[36] International Laser Display Association. *The ILDA Standard Projector*. 1999. Available from: https://www.ilda.com/resources/StandardsDocs/ILDA_ISP99_rev002.pdf

Acronyms

BPM Beats per Minute

CPU Central Processing Unit

DAW Digital Audio Workstation

DB-25 D-subminiature connector with 25 pins

DJ Disc Jockey, a performer who mixes records live

DMX-512 Digital Multiplex with 512 channels

EP Extended Play record with more tracks than a *single* and fewer tracks than an *LP* (long playing record)

Full HD Full High Definition resolution of 1920 x 1080 pixels

GLSL Graphics Library Shader Language

GPU Graphics Processing Unit

GUI Graphical User Interface

HDMI High Definition Multimedia Interface

ILDA International Laser Display Association

MIDI CC MIDI Control Change, a type of MIDI message

MIDI Musical Instrument Digital Interface

OpenGL Open Graphics Library

Pd Pure Data, a visual coding language

VJ Video Jockey, a performer who mixes videos live

VST	Virtual Studio Technology
WebGL	Web Graphics Library
XML	Extensible Markup Language