Free / Libre / Open Source Software and Performance Art: Three Case Studies

BA (Hons) Contemporary Performance Practice: Performance Arts

Third Year

Riviera Taylor

Tutor: Adelina Ong

Student ID: 192687

Unit: Dissertation

Word Count: 8480

I hereby declare this work is my own and free from plagiarism and collusion

Acknowledgments

This dissertation would not have been possible without the time generously offered by interviewees. I offer my thanks to Winnie Soon, Nancy Mauro-Flude and Mynah Marie. I would also like to thank Malachy Orozco for putting me in touch with Louis James.

Contents

Introduction

Free / Libre / Open Source Software (FLOSS) encompasses a diversity of concepts. Political and ideological differences exist between free / libre and open source software (advocates). These differences manifest in advocacy campaigns, historical points of disagreements, software licences and contemporary debates. This dissertation explores the relationship between FLOSS and performance-based art practices. It sits in dialogue with and has influenced my practice, which is equally concerned with the intersection of FLOSS and performance art. The main question underscoring my research is: How might practitioners increase knowledge of the intersection of performance art and FLOSS? The literature review outlines philosophical, anthropological and critical approaches to FLOSS. The discussion takes particular interest in how FLOSS relates to creative practices. Following this I undertake three case studies of domains of performance practice which utilise FLOSS. In the first case study, I consider *Vocable Code*, an executable codework written (and occasionally performed) by Winnie Soon and Geoff Cox. The work is a p5.js sketch exploring the concept of queer code and the relationship between code and speech. The debate elaborates on the implications of connections between the execution of source code and the performative status of a codework. In the second case study I discuss live coding. Live coding practitioners describe their craft as a form of performance art. It is a practice I have taken up since I began writing this dissertation. At live coding events performers demonstrate their ability to produce visuals and / or music with specifically designed FLOSS. This involves writing and evaluating code during a performance. Audience members might dance to the music or watch the code being written and edited on a screen

or projector. I provide an account of my own experience of attending an algorave where a variety of live coders shared their work with an audience. I also consider the implications of the content of conversations I have had with two practitioners, Mynah Marie and nunez. The third case study covers the work of two practitioners, Nancy Mauro-Flude and Marloes de Valk. I describe the practice of these individuals as alternative insofar that the works considered are neither live coding nor codework. The work of Mauro-Flude, for example, resembles live coding and codework yet it is neither of these. Each case study considers the work of two practitioners. To that end, collaboration is a crucial element of FLOSS based performance art practices. Feminist practice is another recurring theme.

Methods

Use of FLOSS

Engaging with pieces of FLOSS has been a method of my research into the intersection of FLOSS and performance art. Indeed, the way I have written this text is determined, in part, by the software I choose to utilise. According to Robert Stake '[q]ualitative case study is characterized by researchers spending extended time on site, personally in contact with activities and operations of the case' (2005). I have spent time on site in a holistic, continuous sense by switching my computer's operating system to GNU/Linux Debian. Downloading and setting up Debian on a desktop or laptop is something anyone with an internet connection can do for free. Debian and other free software is something I have been working with throughout my dissertation. Knowing how to use GNU/Linux and the command line is a practical skill which I have been refining in the process of writing this text. Immersing myself in FLOSS, I engaged further with software which I now utilise on a daily basis: GNU Emacs. As a GNU Emacs user I made this dissertation and several artworks by using different pieces of software in tandem with the text editor. Doing so gave me a better understanding of how technologies can function in conjunction with each other. I was inspired by Ilias Bergstrom and R. Beau Lotto's concept of 'code bending' (2015). Code bending occasionally involves patching softwares together. To produce a typeset copy of this dissertation it was necessary to assemble an academic writing tool-chain. My current workflow involves several pieces of FLOSS working in tandem with one another.

Case Study Research Questions

I turned to the field of FLOSS and art in order to develop a research question. Robert Yin advises that

> What's going on in the field might … suggest relevant questions for study. However, be careful with this [as y]ou may be unduly swayed by transient conditions that won't lead to insightful research questions. Also, a lot is going on in the field, so knowing where to focus your attention may be no easier than culling the literature to identify good questions (Yin, 2018).

I strove to take this advice on board in deciding upon a research question and was particularly inspired by my reading of Letizia Jaccheri. During the International Conference on Entertainment Computing in 2011 Jaccheri held a tutorial on open software and art which addressed the following question: 'How can we increase knowledge about the intersection between software and art?' (Jaccheri, 2011, p.468). According to Jaccheri this is an historical question which has influenced research and practice in software art related fields for several decades. I have found it advantageous to ground my research in a question that has historical standing in a closely related field. For one thing my doubts were allayed as to the transience of the question. This dissertation has a particular focus on the intersection of FLOSS and performance art. Accordingly, I have decided to adapt Jaccheri's question and rephrase it somewhat. The question which I consider is: How might practitioners increase knowledge at the intersection of performance art and FLOSS?

In response, I have conducted three individual case studies of genres of work at the intersection of FLOSS and performance art. The cases each cover different practices and practitioners. There is a case study on code work and the work of Winnie

Soon. A case study on live coding which includes a discussion of an Algorave. Finally, there is a case study on alternative software art practices. I decided to conduct case studies and then I chose a 'how' question. The interrogative structure of a 'how' question, according to Yin, is more amenable to 'analytic generalisation' (Yin, 2018). This is helpful when seeking to draw wider conclusions about the field. Each case study is organised around an interview with one or more practitioners whose work occupies the intersection between FLOSS and performance art. A structure comprising several case studies is referred to by Yin and Robert Stake as a *multiple* or *collective* case study (Yin, 2018; Stake, 2005). The collective case study is 'chosen because it is believed that understanding [an array of cases individually] will lead to better understanding, and perhaps better theorizing, about a still larger collection of cases' (Stake, 2005, p.446).

An example of a collective case study from the field is Eduardo Ledesma's 2015 research into Latin American codework. Codework is a 'branch of new media experimental writing' which 'makes exterior the interior workings of the computer' (Raley, 2012). In his text Ledesma carries out research into several practitioners and their creations. The author argues that geographical region has an aesthetic and ideological impact on software art. I found Ledesma's argumentative structure convincing. Similarly, each case I consider covers the work of a different practitioner, sometimes several. My research differs insofar that I focus explicitly on the intersection of FLOSS and performance arts. I was also keen to mix interviews and case studies. To that end, it is hoped that these studies might illuminate the performative elements of other FLOSS-based arts practices.

Interviews

I created and personalised interview invitations early on in the project. I was keen to address ethical matters in these invitations which sought written consent from participants. I provided participants with information about the research, detailed their role, covered some ethical matters and clarified why I was involving them. In line with ethical requirements and best practice I assured participants that their engagement with the research was voluntary and; that their consent could be withdrawn at any time. I detailed that recordings of interviews would be stored securely on my computer and used only for purposes relating to the research. Interviews were carried out and each lasted approximately half an hour. The interviews were semi-structured insofar that I had a question which I put to each participant: What would the consequence be if the free software you use in your practice were not free? All but one of the interviews were recorded and transcribed. Overall, the interviews have formed key elements of each case study. If I had the chance to carry out this research again I would make some methodological changes. For example, I probably would conduct structured rather than semi-structured interviews. Asking the same questions to each interviewee would facilitate further articulating differences between various practitioners. This would be useful in relation to discussing both the practices and artworks associated with each of the practitioners.

Literature Review

Christopher Kelty writes that free software is composed of five elements. The first of these involves 'practices of argument and disagreement about the meaning of Free Software' (Kelty, 2008, p.14). David Berry reminds us '[t]he free/libre and open source movements developed out of debates and practices in hacking, programming and technical groups in the 1980s and 1990s' (Berry, 2008, p.31). Berry draws attention to the points of tension between Richard Stallman and Eric Raymond regarding the difference between free/libre and open source software. Stallman's work is closely associated with the Free Software Foundation (FSF) and the GNU Project. Raymond (1998) was in favour of open source and he co-founded the Open Source Initiative (OSI). Berry recommends that 'the FSF … make a more concerted attempt to deal with the strongly objective modality of the discourse of the open source movement' (2008, p.184). For example, the OSI have criticised the term "free", suggesting that the word is ambiguous. The OSI writes that 'communication is hampered due to arguments over whether a particular piece of software is "free" or not. This is bad enough when the argument is between people who basically agree that source should be available' (Open Source Initiative, 2006; see also Kelty, 2008, p.321–22). For Stallman (2002, p.58), this ambiguity is a debate about the definition of free software.

Kelty's next element is 'sharing source code' (2008, p.14). Consider this practice in the following example of *Free Radio Linux* (Radioqualia, 2001). According to Adrian Mackenzie this 'work consisted solely of the source code … of the Linux kernel … being read out line by line over various radio and streaming web-radio

sites' (Mackenzie, 2005, p.78). The performance utilised a 'speech synthesizer' to translate '4,141,432 lines of code into speech (taking an estimated 593.89 days to read)' (Cox and McLean, 2013, p.64). On the one hand, Kelty's analysis of free software is anthropological. He does not discuss the relationship between free software and art. On the other hand, Kelty draws attention to the dual quality of source code as 'both an expressive medium, like writing or speech, and a tool that performs concrete actions' (2008, p.118). For Alex McLean and Geoff Cox *Free Radio Linux* 'produces an uncertain relation between the code object and code subject—the program and the programmers—and thus challenges property relations bound to the development and distribution of code' (2013). Cox and McLean likely have in mind those property relations which are characteristic of the proprietary software ecosystem. The authors of the Linux kernel have the freedom to broadcast their code publicly. They are bound neither to non-disclosure agreements nor to copyright licences.

Licensing is the medium through which property relations are negotiated in FLOSS projects. Berry remarks that licenses 'function as an organising device in managing people involved in FLOSS projects and a means of social control' (2008, p.18). '[A]pplying copyright (and copyleft) licences' is, according to Kelty, a key FLOSS practice (2008, p.14). It is not possible to have private ownership or to have exclusive control of software distributed under a copyleft licence. Rather, the software is collectively owned. Many, but not all, FLOSS-based performances involve the distribution of licensed source code. I return to this discussion about copyleft software art in chapter three. '[T]he practice of coordination and collaboration' along with 'conceptualising openness' are the final two elements of Kelty's

model of free software (2008, p.14-15).

Kelty articulates FLOSS as a handful of interrelated practices that together form a 'system of thresholds' (p.16). This position is echoed by Jussi Parikka who writes of software art from a Deleuzean perspective. 'Software is a relay between a plethora of cultural practices,' Parikka reiterates (2010, p.120). The application of Deleuzean philosophy to the subject of software art has interesting implications. For example, Parikka elaborates that '[a]ssemblages of software act as *agencements*: agencies, vectors of transmission, always in the midst of becoming something else' (p.121). Live coding, a performance practice which I elaborate upon in chapter two, could be regarded as an *agencement*; a fluctuating process and a medium of communication simultaneously. Parikka takes a "hard" ontological position in the debate about the relationship between code and execution. 'Code exists only in its execution - a theme made evident in the mock 'execution' of code through T-shirts … and oral performance' (p.123). I disagree with Parrika on this point; code can and does exist in a variety of contexts. This, I argue, is evident in examples of non-executable code and I will return to this debate in chapter one.

According to Mark Marino 'the work of identifying ideology at play in source code is the subject of the field of Critical Code Studies' (CCS) (2009). In the same paper Marino argues there are ideological differences between pieces of code. He offers readings of an email worm and two separate yet related artworks to illustrate this point. The artworks which Marino considers are Zach Blas' *transCoder* (2011) and Julie Russo Levin's *Slash Goggles* (N.d.). Both could be described as non-executable codeworks. The title, *Slash Goggles*, is based on fan-fiction involving queer relationships between characters in Battlestar Galactica. The code in *Slash*

Goggles articulates the sexual identification of characters from the television series who are called Cylons. The non-executable algorithms in *Slash Goggles* implement some functions which are described in *transCoder*.

Marino is keen to establish what evidence there is of heteronormativity in code. He contrasts the ideology of these queer codeworks with the ideological structures of the AnnaKournikova email virus. The AnnaKournikova virus was an email worm written by Jan de Wit which surfaced during 2001. It spread by baiting people into downloading a file from an email. This file masqueraded as a photograph of the tennis athlete Anna Kournikova but the attachment contained no photographs. Instead, when executed, the code in the file sends the worm to all contacts in one's email address book. Marino argues that the worm works on a heteronormative basis, luring in middle aged men with the promise of sexualised photographs. The artworks, on the other hand, do not exhibit heteronormative ideologies and go against such codes.

Gerald Stephen Jackson elaborates on the relationship between *transCoder* and *Slash Goggles*. According to Jackson, Marino and Blas *transCoder* is a Software Development Kit (SDK). 'An SDK provides programmers with code interfaces that allow them to produce software for complex platforms' (Jackson, 2017). I find it ironic that this SDK, which is FLOSS, is distributed by the artist in the format of an Apple Disk Image. The downloadable artwork include various plain text files. One of these files contains functions and variables which attempt to implement ideas from critical theory. Accompanying each of the concepts is a description of what the code does. Scholars such as Eve Kosofsky Sedgwick and John Langshaw Austin are implicitly referred to. For example, the library entitled 'executable speech acts'

contains a variable called 'theCloset' which 'will silence whatever it is assigned to' (Blas, 2011).

A considerable part of Jackson's argument takes place through a close reading of the source code which constitutes *Slash Goggles*. Differently to Marino, Jackson is less concerned with the ideological content of source code. Rather, Jackson argues that 'computational performativity is gender performativity when viewed through technical and historical contexts of labor and collaboration in technical fields like computer programming' (2017, p.22). His emphasis on close readings of source code as a means of expanding his argument resonates with CCS scholarship.

In summary, Kelty's five elements of free software clarify what is at stake in discussions of FLOSS. The practices which he examines are thematic components of the case studies which follow. For instance, in each case study I highlight that the distribution of source code is a regular motif of FLOSS based performance artwork. This resonates with Kelty's second element of FLOSS. Overall, the literature is illustrative of some ways in which it is possible to make sense of source code in artistic contexts. The nexus of queerness and code is a salient feature of the latter part of the literature review. These topics are explored from a non CCS perspective in chapter one. In chapter two I discuss live coding. On the one hand, CCS perspectives on live coding performances present opportunities for new research. On the other hand, I do not offer a reading of any artworks from the perspective of CCS. This is mostly due to my lack of technical knowledge of programming languages. Nevertheless, I have included a discussion of CCS within this literature review because it is a nascent approach to code and code-based artworks. Lastly, I return to the topic of software licensing in chapter three.

Winnie Soon and Geoff Cox's *Vocable Code*

What is the relationship between source code, execution and performativity? Occasionally, execution of source code is regarded as a hallmark of performative codework. As Soon points out at the end of our interview: 'If you think about software performance; if you use the technical terms, it is about running and executing code' (Appendix A). The nexus of performativity and execution is critically considered by Inke Arns who writes that pieces of code in '[c]odeworks can *potentially* be executed and thus become performative' (2005, p.8). The assertion that there is no requirement for software to be executed or executable for it to be regarded as performative has several consequences. Firstly, the argument bestows the status of performative upon non-executable codework resulting in an 'expansion' of codework which falls under this heading. Secondly, by identifying the possibility of execution as the locus of performativity Arns challenges the view that codework is only performative when executable. Arns considers Geoff Cox et al.'s view that 'the aesthetic value of code lies in its execution' (Cox et al. in Arns, 2005). She does so in relation to the (false) promise of executability as the basis of performativity. To that end, Arns agrees that the executability of codework reinforces its performative quality.

There is a consonance between the running and executing of source code and the performativity of source code. That is not to say these terms are synonymous. *Vocable Code* (Soon and Cox, 2018) sits in dialogue with other examples of executable codework such as Graham Harwood's *Class Library* (2011) or the anonymous poem *Black Perl* (1990). In the words of Ledesma such works are

'[t]ruly performative'; executable codework 'simultaneously does something (it runs and produces output) and it states something (through both its output and its code)' (2015, p.93). Ledesma's concept is pertinent here because *Vocable Code* is an executable codework. By contrast, Arns is primarily occupied with the performativity of non-executable codework. Her argument that non-executable code is performative implies that non-executable code exists. In this respect, Arns would disagree with Parikka's assertion that code exists only in its execution. Overall, Arns' position reiterates the textual quality of code and thus encourages a rethinking of the basis of performativity in *Vocable Code*. Detaching performativity from technical execution clears a path for imagining different ways in which source code might be executed.

I now turn to the interactive, online artwork which is part of *Vocable Code*. Upon visiting the website hosting *Vocable Code* the browser window is split in two. On the left hand side of the screen is Soon's p5.js code-based sketch for the artwork. On the right hand side of the screen is a black area with centre-aligned off-white text which generally rises up vertically before floating downwards and fading to grey. The code on the left hand side corresponds directly to the functioning of the visuals on the right hand side. By "tweaking" the code one can, for example, edit the transparency of the text colour, or the alignment of the text. One sees this change reflected in the visuals. This ability to modify the artwork captures the open source nature of the software. The artwork features completions of the sentence 'queer is …' (Soon, 2018). These completions have been offered by various participants. Accompanying the visual component there are audio recordings of the phrases which were offered as way of completing the sentence. It is the voices of the

participants that the audience hears. These audio recordings correspond to the textual snippets moving on the right hand side of the screen. The artwork has also been exhibited on several occasions since its inception in 2017. It is due to be shown at the Science Gallery in Australia in 2023. In these exhibitions two monitors are supplied along with a pair of headphones. Audience members wear the headphones and can watch the left and right, portrait monitors. It is not clear from the exhibition photography whether a keyboard is supplied such that the audience could tweak the code.

The code which features in *Vocable Code* was written with p5.js (free software similar to *Processing* but written in the JavaScript programming language). In our interview Soon stated 'I would definitely not use p5 if it were not a free software, it's an important component in my work' (Appendix A). By this does Soon mean that using FLOSS is paramount in their practice? Or does Soon simply mean that p5.js is crucial to their practice? In my view, both interpretations are reasonable although to some extent I assume Soon meant the former. This is mostly because Soon said that 'I think it is quite crucial for me that I try my best to work with free software for any kinds of explicit or implicit production of the artwork' (Ibid.) Soon raised the question of what was meant by free software early on during our conversation. This practitioner emphasised that '[i]t's not just only free in terms of costs. Freedom in terms of the freedom to modify, freedom to make change, freedom to study, freedom to copy, and to produce something on top' (Ibid.). Freedoms such as the freedom to study have been described amongst the 'four kinds of freedom' secured by free software (Stallman in Berry, 2008, p.164).

The FSF have eagerly expanded upon a particular sort of discourse in view of

computer users lack of freedoms. There is reason to be sceptical of the FSF's discourse surrounding free software. As Berry points out the FSF's choice of language is at times evangelical. In 1990 Stallman was awarded a 'genius grant' which according to Berry, 'also meant that now [Stallman] could take more time out to evangelise around the world to increase knowledge and awareness of the GNU project' (p.116). Evangelism is a possible course of action for practitioners grappling with increasing knowledge at the intersection of free software and art. However, such evangelism relates more to free software advocacy than to free software as it intersects with performance art. I am sympathetic to the FSF's causes although I do take issue with such an evangelical approach to free software advocacy. There is a clear "us/them" strand of thought in their concept of free software which mingles uncomfortably with moral puritanism. This combination alienates me somewhat from the ideological campaign of the FSF.

As Berry has also pointed out freedom features in the discourse of the Open Source Movement (OSM). It is not within the terms of this movement that I expect to find recourse to the freedoms outlined by Soon. Nevertheless, 'the OSM regards freedom entirely within the realm of an individual's economic freedom and their freedom to work on projects' (p.176). Berry is as critical of the OSM's neoliberalism as he is of the Free Software Foundation's evangelism. Notwithstanding it seems Soon's emphasis on feminist computing practices and small technology situates her at a distance from this debate between the OSM and Free Software Foundation. Indeed, feminist computing practices offer a real alternative to these often invoked discourses.

Vocable Code was developed during a feminist coding workshop which questioned

whether software could be feminist. The work probes at the history of anti-feminist internet discourses by way of a constrained writing style. Soon and Cox were inspired by the feminist programming language 'C Plus Equality' (C+=). The satirical programming language was instigated by internet trolls and caused offence (White, 2013). In light of this, I am disinclined to agree with Soon that C+= 'is a feminist programming language written by and for feminists' (2018). Notwithstanding, in what I perceive as an act of reclamation, Soon and Cox find a productive tension in applying writing constraints inspired by this programming language. For example, Cox and Soon declined to use binary 0s and 1s in the code and were '[m]indful of all the variable, array and function's naming' (ibid). Perhaps such mindfulness draws attention to the arbitrariness of the language used to denote functions in code. Like Paul B. Preciado, Soon speaks toward the limits of our current vocabularies which surround descriptions of gender and sexuality. 'Homosexuality and heterosexuality, intersexuality and transsexuality do not exist outside of a colonial, capitalist epistemology' writes Preciado (2019). What does it mean to speak 'let queer rights equal' an empty array (Soon and Cox, 2018)? By exploring such things the relationship between language and what is imaginable is brought into question.

'Vocable Code is both a work of “software art” (software as artwork, not software to make an artwork) and a “codework” (where the source code and critical writing operate together) produced to embody “queer code”' (Soon, 2018). The score for the performance-lecture involves two performers reading aloud excerpts of source code, quotes from academics, and other pieces of text written in English. Reading source code aloud is to wrench it from its most conventional context as

something read and dealt with by a computer. The performance lecture has been given on previous occasions by Cox and Soon. Indeed Cox, in collaboration with Alex Mclean, has written about the concept of Vocable Code independently of the artwork.

Arns argues that 'code is not only executable in technical environments, but can become extremely productive as “imaginary software” in readers themselves' (2005, p.8). In this sense, the instruction to vocalise source code can be understood as a form of 'imaginary software', an algorithm to be executed by a performer. To illustrate this, I would like to draw a parallel between Pauline Oliveros' *Software for People* and *Vocable Code* as a performance-lecture. Oliveros' *Software for People* is an early and historical example of what Arns might call 'imaginary software'. 'In the third part of the paper [Oliveros] presents some of the “theory concerning [her] ‘software for people’” in which she aligns ideas from psychology and information processing to explain her model for the organisation of sensory attention' (deLahunta, 2017). 'The program (sic.), or software … is as follows: On cue from the conductor play a very short tone. Each player's partner then tries to react with exactly the same pitch as quickly as possible' (Oliveros, 1984, p.187). Oliveros' Software is imaginary in the sense that the code to be executed is not written in a programming language such as JavaScript; it is executed in the bodies and minds of the performers. In a similar sense the software which the performers act upon in the lecture-performance is the instruction to read symbols aloud. There are thus multiple senses of software at play in *Vocable Code* along with the source code itself. Accordingly, there are various ways in which the code can be executed. This is attested to by the several forms which *Vocable Code* takes.

Live Coding

Live coding constitutes a considerable part of the intersection between FLOSS and contemporary performance practice. In live coding, performance artists write code to produce visuals and music in real-time with purpose built software. The music is played through loudspeakers, the visuals are often projected (See McCallum and Smith, 2011; EHO, 2013). According to Eli Fieldsteel 'live coding is a performance practice that involves writing the source code for a piece of music in real-time as the piece is happening' (Fieldsteel in Nyokee, 2021). Fieldsteel is an academic who has produced several live-streams of semester-length, university courses. In the recordings, which are freely available on YouTube, Fieldsteel teaches his students how to code with SuperCollider. Fieldsteel is not the first to organise a university-level course on this software (Blackwell et al., 2022c, p.19). However, making such knowledge publicly available on YouTube strongly reflects the epistemology and ethos of free software. Notwithstanding, Fieldsteel's description of live coding here is partial because live coding can also be performed with visuals (graphics or video). Still, his words resonate with Dave Griffiths' assertion that live coding is 'the practice of programming as a performance art form' (2010). To be more specific, as Alan F. Blackwell et al. put it, '[i]n asking “What is live coding?,” our intention is not to fix or define but rather to explore how live coding *opens up*' (Blackwell et al., 2022a, p.2). These authors suggest the absence of a 'formal definition' entails that 'established practitioners or institutions' cannot make a claim to own live coding (ibid.). In this sense the practice is opposed to hierarchical control. It reflects the distributive practices of FLOSS more widely and in an often immediate and performative way.

Otherwise referred to as on-the-fly programming, live coding takes place in a live coding environment and particular programming languages. In short, as McLean points out, '[l]ive coding environments make it possible to write code to generate music and video in real-time without having to restart any processes' (2008). Oftentimes it is possible, sometimes it is necessary, to interface with live coding software via a text editor. In this regard live coding environments can span across different pieces of software. In my own creative practice I have experimented with using GNU Emacs as a text editor for creating compositions in TidalCycles and Fluxus. GNU Emacs is the same software which I used to write this dissertation. That it's possible to use one piece of software both for academic writing and for creating algorithmic artworks demonstrates the flexibility of GNU Emacs. Advances in computing technology during the 1980s and 1990s made contemporary live coding possible. Moreover, there are many live coding environments which programmer-artists may choose from. The vast majority are pieces of FLOSS. Some examples of live coding software include Fluxus (Griffiths, 2020), Extempore (Sorensen, 2018) and Threnoscope (Magnusson, 2022). To that end, it is frequent for live coders to create their own live coding environments. Such environments and languages are often distributed over the internet and demonstrated at live coding events.

TOPLAP was founded in Hamburg 2004. It is a network associated with software art and it focuses specifically on live coding. TOPLAP have hosted get-togethers for live coders for many years. TOPLAP also have an "awesome" GitHub repository which contains a list of live coding resources. Amongst this list one readily finds the live coding environments described above. The TOPLAP wiki details members

and bands associated with the organisation as well as a manifesto which outlines what TOPLAP is about. McLean is a key figure associated with the organisation because he founded it (Burland and McLean, 2016). He regularly gives interviews, has written extensively on live coding and developed the TidalCycles live coding software.

Going to an Algorave

Who attends live coding events? Ledesma, along with Burland and McLean, expresses that a significant number of people who attend live coding events are artist-programmers interested in the art form (Ledesma, 2015). Burland and McLean also found that 'liking the artist', 'enjoying high quality music' and 'enjoying previous events' were strong motivating factors for audiences to attend (2016). Live coding often takes place at Algoraves. As part of my research I decided to attend one of these events to witness examples of the performance practice first hand. Thus, on December 3rd I went to IKLECTIK to see a line up of performers. IKLECTIK is a venue located a short walk away from London Waterloo and Lambeth North stations. The line up included the following artists: Tyger Blue, Mxwx + The Animalizer, eye measure, Michael-Jon Mizra, +777000 & nunez, hellocatfood. The atmosphere of the evening was pleasant. People were dancing and I had several conversations with other attendees. According to the event organiser approximately 100 people attended the event. Going to the Algorave was key in the development of my practice as it introduced me to TidalCycles.

At the Algorave two of the acts featured more than one performer collaborating with another performer. The idea of a networked performance arises. In such

performances notions from telematics come together with people to create live artworks which bridge physical distances (Rubio R., 2014). Of course, performers need not be separated by international borders to perform in a networked way. Indeed, all of the performers at the Algorave were physically together at the event. However, other live coders such as Mynah Marie have found networked performance essential as a medium for producing live music. Marie is part of several live coding collectives who produce networked performances. For this, FLOSS such as Estuary has proven indispensable for enabling people to collaborate and live code across distances and time-zones. Supercontinent are an example of a live coding collective based in different places around the world who utilise Estuary for collaborative purposes (4TH SPACE Concordia University, 2021). Marie is part of this collective who have had members from Europe, India and Japan. Marie is also part of a feminist live coding collective called livecoderA. This practitioner detailed that when performing as a collective Supercontinent would have one performer in person representing the group at an Algorave somewhere. Other members of the group would then log in to Estuary from around the world to perform together. Several of the group members would live code audio whilst one member would take care of the graphics. To create audio Supercontinent use Mini Tidal, a lightweight version of TidalCycles and for creating visuals they utilise Punctual. In general '[m]ost of us code in Mini Tidal … and then one person will code visuals' (Appendix B). Marie emphasised that the members of Supercontinent wanted their networked practices to reflect the distributed nature of free software projects; the ecosystem at large. I think such an aesthetic approach is very pertinent. It disrupts what has been described as the fiction of solo genius in favour of collaborative modes of production.

Marie is a 'traditional musician … an instrumentalist and a singer' (Appendix B). During our interview, Marie highlighted that live coding was key for the advancement of their creative practice. This artist records music and utilises Sonic Pi in their solo practice to create ambient sounds. Sonic Pi was written by Sam Aaron and was released as free software in 2012. It runs on Windows, MacOS and Raspberry Pi OS. Marie explained that their relationship to the software is qualitatively similar to working with an instrument. Furthermore, in comparison to other live coding environments Marie stated that

> [t]he code is easier also for people to follow … because the language is closer to English. So even people that don't know code can kind of read what's going on on the screen and have a bit of a sense of what's happening (Appendix B)

Besides some of the reasons given above for attending live coding events Burland and Mclean state the following: 'It is possible that perceptions of an open and like-minded community encourages the possibility of sharing and learning and encourages subsequent attendance and future involvement in coding' (2016). I asked Marie what would be lost if Sonic Pi were not free software. Marie's response was '[t]hat would not happen …. It's just so embedded in the mentality of the community' (Appendix B). In many regards Marie's faith in the community is well-placed. It would be difficult for Sonic Pi to become non-free software because specifically licensed copies of it exist and are collectively owned. No one is in a position to privatise Sonic Pi. Marie briefly discussed the educational potential of live coding. The projection of code constitutes an act of knowledge dissemination. To that end, live coding can offer a means of empowering creative people. For Marie, live coding seeks 'to empower people to understand code so that they can

build their tools' (Appendix B). It is also 'a protest of taking back ownership of something that we believe should be open to all' (Ibid.).

Alternative Practices

Nancy Mauro-Flude

Nancy Mauro-Flude is an academic and artist based in Australia. Mauro-Flude's alternative software art practice engages with some different modes of collaboration. This artist's software art practice is situated within a broader oeuvre of creative works. I discovered this practitioner via Denis "Jaromil" Roio's website. Roio is a maker of free, libre, open source, Rastafari software. His software finds a parallel with feminist practices insofar that both depart from normative assumptions about software engineering. Mauro-Flude and I met for an online interview on December 12 2022. Mauro-Flude has utilised Time Based Text in her performances for over a decade. The software was written by Roio in collaboration with Joan Heemskerk and Dirk Paesmans who both work together under the moniker of Jodi. Mauro-Flude clarifies what this software does in our interview: 'It basically logs your keystrokes into the terminal. It also logs the timing of the keystrokes, obviously. Then you can customise that and then put it into a web browser as a log or an expression, potentially, of how you wrote something' (Appendix C). In other words, Time Based Text is capable of recording sequential keypresses. These can then be played back within the terminal or used in conjunction with HTML code to create playful web pages. It is also possible to edit the way in which the Time Based Text recording gets played back. Several of Mauro-Flude's works are Time Based Text compositions. Mauro-Flude's use of FLOSS illustrates how such technology can fulfil artistic ends across international borders.

Hacking Time Based Text to enable it to function with contemporary technology was something Mauro-Flude spoke about during our interview. 'It's hard to use, there's always glitches. … It doesn't just work out of the box, by any means. It's not like a package', says Mauro-Flude (Appendix C). That's because the software was written several years ago when older technology was available. It's not designed for the most up-to-date hardware and software. Therefore, it's necessary to edit the source code of Time Based Text to keep it functioning as intended. Marloes De Valk writes that

> access to the source code of the software, facilitating the possibility of compiling it with optimizations for specific hardware or purposes, can mean a big improvement in performance as well—never mind the possibility of actually modifying the source code of the software itself to match your needs (De Valk, 2009)

The decision-making freedom engendered by FLOSS guides Mauro-Flude's choices in several ways. On one level, it affects Mauro-Flude's choice of software: she prefers PureData to Ableton. On another more significant level it affects what one can do as an artist and impacts the aesthetics of the work. De Valk is quick to highlight that software artists have specific needs which FLOSS caters well towards, including the ability to make changes in accordance with ones needs. Because Time Based Text is open source any one is able to contribute towards improving the functionality of the software. Thus, Mauro-Flude was able to meet her browser-based playback needs by editing the source code of the software. In my own practice, I have attempted to use Time Based Text and seen it not function as intended with web browsers. Why does this happen? Mauro-Flude alludes to some practical limitations which impact the improvement of FLOSS. Other people will have different computing setups to one's own. 'Even if we're both using the

same MacBook Pros, I've probably set mine up [sic] and have different features and constraints than yours' says Mauro-Flude (Appendix C). When troubleshooting software which was written several years ago one encounters this problem of different computing setups. Thus, improvements which Mauro-Flude has made to Time Based Text over the years might not work for other computer users.

Mauro-Flude's work often combines the poetic with the codified in a way similar to codework. It also experiments with liveness, and textual interfaces in a way similar to live coding. The artist tends to use terminal emulators in performance. As Blackwell et al. note '[c]ommand-line interfaces are essentially live coding interfaces' (Blackwell et al., 2022b, p.245). However, Mauro-Flude's work cannot accurately be described as live coding or codework. Speaking of some of her earlier work with Time Based Text she states: '[i]t's kind of live coding — in that genre — but before I knew what live coding was. It's live typing in a theatre in a way, or text-based performance. I'm using the computer, the web browser as a stage, and obviously logging the keystrokes' (Appendix C). Mauro-Flude's language of revealing the 'inner workings' (Ibid.) of a computer resonates with Rita Raley's discussion of codework. I would be inclined to describe Mauro-Flude's FLOSS-based performances as codeworks insofar they share this ambition. In general, it's a more theatrical take on software art Occasionally theatrical characters are deployed, such as in *Error_In_Time* (2014). The use of character in this performance, which also employed Time Based Text, is a creative decision. During this 30 minute performance Mauro-Flude types into a terminal. She executes commands, runs different pieces of software and performs movement whilst interacting with a computer.

Mauro-Flude's performances further illustrate the enabling potential of FLOSS. However, it is to some extent difficult to understand how Mauro-Flude's FLOSS pieces function. This is because Mauro-Flude tends not to distribute source code for these works. Instead, several of her performances utilise FLOSS. Moreover, Mauro-Flude adjusted the source code of Time Based Text to enable it to function as intended with up to date technology. This exemplifies the ability to make changes in accordance with ones needs. In the context of a rapidly changing technological landscape, Mauro-Flude's work highlights the collective drive behind, adaptability of and issues confronting FLOSS projects. To me, Mauro-Flude's practice also raises technical questions about everyday computer usage. This varies from ones choice of shell to the ways in which various pieces of software are installed. Such details have grown in importance to me the more I have come to utilise FLOSS.

Marloes de Valk

In *Tools to Fight Boredom* (2009) the crux of de Valk's argument is that (software) artists have specific needs which only FLOSS can fulfil. The author writes that 'the need for freedom to express concepts without being hindered by unnecessary technical limitations' is of importance to such artists (p.91). Such a freedom differs subtly to the freedoms championed by Soon and Mauro-Flude. Taking into account elements of the operating system one is using is something which de Valk foregrounds. GNU/Linux is advocated for over proprietary operating systems for several reasons. For example 'commercial software often uses closed formats' whereas with GNU/Linux openness is a core principle. However, moving to an entirely different operating system takes time and can result in a lack of artistic

"output". For this reason, managing the relationship between developing tools and artworks is something which the author recommends. De Valk succinctly states that '[u]sing GNU/Linux and FLOSS in an artistic practice brings an unavoidable political message to the work' (p.98). At the same time, '[t]he political message that all work made with FLOSS carries in it should not be in the foreground, unless this is part of the artistic concept' (p.99).

Several of de Valk's projects have involved performance. Collaborative pieces such as *synth.f* (2007) and *The Silkthreads Mainframe AKA Spiderweb* (2008) have been performed at the Piksel and Transmediale festivals respectively. The former of these works might be better described as hardware art rather than software art. The latter is a 12-hour performance work, directed by Martin Howse and Shu Lea Chang, in which de Valk featured as a player. The artwork involves the building of software. During the performance PureData patches were written to re-imagine the final twelve minutes of Akira Kurosawa's 1957 film *Throne of Blood*. The artwork draws together a variety of practitioners who together produced a durational sonic installation. Audience members were able to watch the development of the PureData code and listen to the sounds produced. Besides working on software artworks, de Valk has endeavoured to enhance knowledge at the intersection of FLOSS and performance arts through writing projects. For example, she co-edited a book titled *FLOSS + Art* (2008). The book features over twenty contributions from practitioners including a text by Mauro-Flude.

De Valk also offers practical and economic arguments as to why software artists might distribute their work under copyleft licences. Copyleft is an anti-copyright practice prevalent amongst developers of FLOSS and some software artists. By

licensing work under copyleft terms, artists and collectives position their works in opposition to international copyright law. In that regard they express a desire for a freer state of affairs and different market dynamics. Permissive licences are distinct from copyleft licences insofar there is no requirement for derivative work to be licensed under the same licence. According to Aymeric Mansoux 'permissive licenses are sometimes referred to as copyfree licenses by their supporters, and the advocates of this term are openly against copyleft' (Mansoux, 2017, p.207). Andrew Katz (2012) writes that permissive licences express the following community sentiment: 'we want our designs to be as broadly used as possible, and we don't care if they are used for proprietary purposes'. Permissive licences are indicative of a "third way" when compared to copyright and copyleft. In practice, de Valk argues, copyleft encourages an ecology based on openness, sharing and peer review while resisting fictions of economic scarcity propagated by copyright.

Hello Process! (2017) is an example of copyleft software art. The artwork was produced by de Valk in collaboration with Mansoux. The installation involves a printer and a computer. The printer prints what it receives from the computer: a file. However, the artists did not simply print images or text. Instead Mansoux and de Valk printed a variable pattern of forty lines. Each line contains 128 blocks and '[e]ach block can contain a small program of up to 1024 bytes. The blocks are executed one after the other. Each program can move, copy, swap or delete any block in the file' (Ibid.). To achieve this the artists 'needed direct access to the printer's graphics mode, so [they] wrote a shell script that directly interpreted numbers and turned them into the right escape sequence for the printer' (De Valk, 2009, p.92). The intention of the work is to illustrate the processes which take place

inside a computer. To some extent, the concept echoes Freidrich Kittler's argument in his 1995 essay *There is no Software*. Software, Kittler argues, can be reduced 'to signifiers of voltage differences' and therefore, in a sense, there is only hardware. However, as Matthias Felleisen et al. point out 'a raw computer is a useless piece of physical equipment' if there is no software installed (2014). Kittler's argument reiterates that software, if it exists, puts hardware to work. This is indicated to in the artwork by the sounds of the printer doing its job. Furthermore, Source code for the artwork is licensed under the GNU General Public Licence. Anyone with access to the source code can recreate the artwork with the appropriate requirements in place.

Conclusion

Ascertaining what would be lost if the FLOSS which practitioners utilise was no longer free was a question I put to each interviewee. Marie stood in disbelief at the idea of FLOSS being or becoming non-free. Soon, I gather, uses free software a lot so non-free software has a limited place in Soon's practice. Mauro-Flude considered what is gained by the software being free, noting that the possibility of tweaking the software would rapidly vanish. This possibility has not been considered in the existing literature on FLOSS and art. Indeed, there is limited literature on the intersection of FLOSS and performance art. This dissertation is a record of what I have found in exploring a field which is rife with equality of opportunity. Almost every computer user has free software installed on their computer in some form or another. Users such as myself and the practitioners I interviewed, stand with firm belief in the potential of FLOSS.

This dissertation has explored several examples of creative practices that engage with FLOSS. The cases cover three practices which characterise the field of FLOSS-based performance art. These include code work, live coding, and alternative practices. The cases illustrate that the act of distributing source code is a regular motif of performance practices reliant upon FLOSS. I have considered how the politics of FLOSS play out in artworks which are engaged with or involve the making of free software. I have found that sharing openly and collaborating together are cornerstones of FLOSS-based creative practices. Methodologically I have immersed myself in FLOSS in order to produce this text. Doing so has advanced my understanding of the GNU/Linux operating system.

My research has been guided by the question of how practitioners might increase knowledge at the intersection of FLOSS and performance art. I have interviewed three active practitioners in the process of producing this dissertation. The research highlighted to me the importance of conducting interviews with others to develop my understanding of the field. My dissertation research spurred me to conduct three further interviews with developers of FLOSS, with a particular emphasis on live coding environments. In this dissertation, each case study is accompanied by at least one interview. I have endeavoured to draw out the implications of each interview in the context of each case study. In that regard, qualitative data was gathered for analytic purposes. A major finding of my research was that, across the interviews, participants argued FLOSS engendered specific freedoms. Sometimes these freedoms overlapped with core freedoms espoused by institutions such as the FSF. But more often these freedoms were specific to the individual. This observation, along with writing about feminist computing practices, is broadly absent in the existing literature on performance art and FLOSS.

Bibliography

4TH SPACE Concordia University. 2021. “Supercontinent Live Coding Ensemble.”

Anonymous. 1990. “Black Perl.”

Arns, Inke. 2005. “Code as Performative Speech Act.” *Artnodes* 0 (4). doi:10.7238/a.v0i4.727.

Bergstrom, Ilias, and R. Beau Lotto. 2015. “Code Bending: A New Creative Coding Practice.” *Leonardo* 48 (1). [Leonardo, The MIT Press]: 25–13.

Berry, David M. 2008. *Copy, Rip, Burn the Politics of Copyleft and Open Source*. London: Pluto Press.

Blackwell, Alan F., Emma Cocker, Geoff Cox, Alex McLean, and Thor Magnusson. 2022a. *Live Coding: A User's Manual*. doi:10.7551/mitpress/13770.001.0001.

———. 2022b. “Notes.” In *Live Coding: A User's Manual*. doi:10.7551/mitpress/13770.001.0001.

———. 2022c. “Partial Histories.” In *Live Coding: A User's Manual*. doi:10.7551/mitpress/13770.001.0001.

Blas, Zach. 2011. “transCoder.”

Burland, Karen, and Alex McLean. 2016. “Understanding Live Coding Events.” *International Journal of Performance Arts and Digital Media* 12 (2): 139–51. doi:10.1080/14794713.2016.1227596.

Cox, Geoff, and Alex McLean. 2013. *Speaking Code: Coding as Aesthetic and Political Expression*. Software Studies. Cambridge, Mass: The MIT Press.

deLahunta, Scott. 2017. “Dance Becoming Data: Part One Software for Dancers.” *Computational Culture*, no. 6 (November).

EHO. 2013. “Live Performers Meeting. Rome.”

Felleisen, Matthias, Robert Bruce Findler, Matthew Flatt, and Shriram Krishnamurthi. 2014. *How to Design Programs, Second Edition*. Cambridge (Mass.): MIT Press.

Griffiths, Dave. 2010. “Fluxus Documentation.”

———. 2020. “Fluxus.” Pawfal.

Harwood, Graham. 2011. “Class Library.”

Jaccheri, Letizia. 2011. “Open Software and Art: A Tutorial.” In *Entertainment Computing ICEC 2011*, 468–71. Springer, Berlin, Heidelberg. doi:10.1007/978-3-642-24500-8_68.

Jackson, Gerald Stephen. 2017. “Transcoding Sexuality: Computational Performativity and Queer Code Practices.” *Qed: A Journal in Glbtq Worldmaking* 4 (2). Michigan State University Press: 1–25. doi:10.14321/qed.4.2.0001.

Katz, Andrew. 2012. “Towards a Functional Licence for Open Hardware.” *Journal of Open Law, Technology & Society* 4 (1): 41–62.

Kelty, Christopher M. 2008. *Two Bits: The Cultural Significance of Free Software*. Experimental Futures. Durham: Duke University Press.

Kittler, Friedrich. 1995. “There Is No Software.” *Ctheory*, October, 10/18/1995–10/18/1995.

Ledesma, Eduardo. 2015. “The Poetics and Politics of Computer Code in Latin America: Codework, Code Art, and Live Coding.” *Revista de Estudios Hispánicos* 49 (1). Washington University in St. Louis: 91–120. doi:10.1353/rvs.2015.0016.

Levin, Julie Russo. n.d. “Slash Goggles.”

Mackenzie, Adrian. 2005. “The Performativity of Code: Software and Cultures of Circulation.” *Theory, Culture & Society* 22 (1): 71–92. doi:10.1177/0263276405048436.

Magnusson, Thor. 2022. “Threnoscope.”

Mansoux, Aymeric. 2017. “Sandbox Culture: A Study of the Application of Free and Open Source Software Licensing Ideas to Art and Cultural Production.” Goldsmiths, University of London. doi:10.25602/GOLD.00022606.

Mansoux, Aymeric, and Marloes de Valk, eds. 2008. *Floss + Art*. Poitiers, France: GOTO 10, in association with OpenMute.

Marino, Mark C. 2009. “Disrupting Heteronormative Codes: When Cylons in Slash Goggles Ogle AnnaKournikova,” December.

Mauro-Flude, Nancy. 2014. “Error_In_Time.”

McCallum, Louis, and Davy Smith. 2011. “Show Us Your Screens.”

McLean, Alex. 2008. “Live Coding for Free.” In *Floss + Art*, edited by Aymeric Mansoux and Marloes de Valk, 224–31. Poitiers, France: GOTO 10, in association with OpenMute.

Nyokee. 2021. “Live Coding in SuperCollider: A Tutorial with Eli Fieldsteel.”

Oliveros, Pauline. 1984. *Software for People: Collected Writings 1963 - 80*. 1. ed. Baltimore, Md: Smith Publ.

Open Source Initiative. 2006. “Why ‘Free' Software Is Too Ambiguous:Advocacy.” https://web.archive.org/web/20060408160813/http://www

.opensource.org/advocacy/free-notfree.php. Accessed 24 Feb 2023.

Parikka, Jussi. 2010. “Ethologies of Software Art: What Can a Digital Body of Code Do?” In *Deleuze and Contemporary Art*, edited by Stephen Zepke and Simon O'Sullivan. Deleuze Connections. Edinburgh: Edinburgh University Press.

Preciado, Paul B. 2019. “Introduction.” In *An Apartment on Uranus*. London: Fitzcarraldo Editions.

radioqualia. 2001. “Free Radio Linux.”

Raley, Rita. 2012. “Interferences: [Net.Writing] and the Practice of Codework.” *Electronic Book Review*.

Raymond, Eric. 1998. “The Cathedral and the Bazaar.” http://www.catb.org/esr/writings/cathedral-bazaar/cathedral-bazaar/. Accessed 24 Feb 2023.

Rubio R., Juan David. 2014. “Ritualized Performance in the Networked Era: Alternative Models for New Artistic Media.” *Leonardo Music Journal* 24. The MIT Press: 21–23.

Soon, Winnie. 2018. “Vocable Code.” *Mai: Feminism & Visual Culture*.

Soon, Winnie, and Geoff Cox. 2018. *Vocable Code (13082018): a Lecture-Performance in Six Parts*. S.n.: DobbeltDagger.

Sorensen, Andrew Carl. 2018. “Extempore: The Design, Implementation and Application of a Cyber-Physical Programming Language.” The Australian National University. doi:10.25911/5d67b75c3aaf0.

Stake, Robert E. 2005. “Qualitative Case Studies.” In *The SAGE Handbook of Qualitative Research*, edited by Norman K. Denzin and Yvonna S. Lincoln, 3rd ed, 443–66. Thousand Oaks: Sage Publications.

Stallman, Richard M., and Joshua Gay. 2002. *Free Software, Free Society*. Boston (Mass.): Free software foundation.

Valk, Marloes de. 2009. “Tools to Fight Boredom: FLOSS and GNU/Linux for Artists Working in the Field of Generative Music and Software Art.” *Contemporary Music Review* 28 (1). Routledge: 89–101. doi:10.1080/07494460802664056.

Valk, Marloes de, and Aymeric Mansoux. 2017. “Hello Process!” `https://bleu255.com/~marloes/projects/Hello_process!/`. Accessed 24 Feb 2023.

Valk, Marloes de, Aymeric Mansoux, and Tom Schouten. 2007. “Synth.F.”

Valk, Marloes de, Valentina Vuksic, Aymeric Mansoux, and Chun Lee. 2008. “The Silkthreads Mainframe AKA Spiderweb.” `https://archive.bleu255.com/pikurimu/2008/02/01/performance-during-moving-forest-transmediale-berlin-de/`. Accessed 24 Feb 2023.

White, Molly. 2013. “Why I'm Not Laughing at C Plus Equality.” *Molly White*.

Yin, Robert K. 2018. *Case Study Research and Applications: Design and Methods*. Sixth edition. Los Angeles: SAGE.