The Integration of Creative Programming in Experimental Music and Digital Art

by

Matthew Pratt

submitted in partial fulfilment of
the requirements for the award of the degree of
Master of Music in Composition

Faculty of Humanities
University of Cape Town

Date of Submission: 23rd January 2023

Co-supervisors: Prof Hendrik Hofmeyr and Mr Theo Herbst

University of Cape Town

The copyright of this thesis vests in the author. No quotation from it or information derived from it is to be published without full acknowledgement of the source. The thesis is to be used for private study or non-commercial research purposes only.

Published by the University of Cape Town (UCT) in terms of the non-exclusive license granted to UCT by the author.

University of Cape Town

Declaration

I, Matthew Pratt, hereby declare that these musical and audiovisual pieces are my original work (except where acknowledgements indicate otherwise) and that none, nor any part of them has been, is being or is to be submitted for another degree in this or any other university.

I authorize the University to reproduce for the purpose of research either the whole or any portion of the contents of the notated scores alone. This authorization expressly excludes the distribution, derivation and reproduction of any of my original creative work (including musical scores, audio recordings, live performances, visual media, audiovisual media, digital media and source code) for non-commercial research purposes without full credit and my formal consent. Distribution of *Vastitas* in excess of 10% of its full duration is strictly prohibited in any context without full credit and my formal consent.

Signed by candidate

Signature

3rd February 2023

Date

Abstract

This explication presents an overview of the background, composition processes and influences involved in the creation of experimental music and digital art using creative programming. By establishing the context of my composition style, the paper will outline the methods, conceptualization and creative language used in the creation of a work of experimental music. It will also touch on the use of creative programming in digital art.

Contents

Figures

Listings

Acknowledgements

I find myself very fortunate to have had the support of those around me whilst completing this degree – without them, this would not have been possible. Therefore, I would like to extend my deepest thanks to the following individuals and organizations (in no particular order):

- Mr Theo Herbst, my co-supervisor, mentor and colleague. His influence over the past three years in my growth as an artist, musician and professional has been second to none.
- Professor Hendrik Hofmeyr, my other co-supervisor and mentor. His guidance, support and contribution to my journey as a composer since my undergraduate years has had a profound impact.
- My parents, for their ongoing support of my studies and endeavours.
- My grandparents for their unshakeable love, encouragement and faith in me and my ventures.
- My aunt Beverley, for her generosity, warmth and solace.
- My brother, "just because".
- Tim, for his utter selflessness and lifelong comradeship.
- Tamara, a true friend, confidant and musical inspiration.
- My dear dog Spike, a loyal companion and study-buddy.
- Dr Paul Modler and Mr Lorenz Schwarz, for giving me the opportunity to work with the Karlsruhe University of Arts and Design.
- UCT Postgraduate Funding Office and respective donors, for their financial assistance, without which I would not have been able to complete my studies.

Abbreviations

AI	artificial intelligence
BE@UCT	Bowed Electrons @ UCT Festival and Symposium
IDE	integrated development environment
LFO	low-frequency oscillator
MIDI	Musical Instrument Digital Interface
NWDR	Nordwestdeutscher Rundfunk
OOP	object-oriented programming
OSC	Open Sound Control
Pbindef	Event Pattern reference definition
RTF	Radiodiffusion-télévision française
UI	user interface

Chapter 1

Introduction

1.1 Background

1.1.1 Scope

In this paper, I unpack the core ideas and methods involved in conceptualizing and realizing a creative work of experimental music that integrates creative programming into the composition and performance processes. I discuss the work's background by defining its artistic context, including my personal musical background and an overview of experimental music with its various meanings. There is a brief introduction to digital and generative art theory, following which I discuss creative programming

applications in the piece. Finally, I deal with creative programming in digital visual art and display three of my own sketches.

1.1.2 Objective

The aim of this explication is to complement the realization of a piece of experimental music, at the same time explaining my compositional language through artistic practices. This dual process is intended to provide an appropriate theoretical and conceptual context by outlining my adoption of a new musical palette that includes the use of creative programming.

Given that this paper is not a traditional master's thesis, the primary objective is not to answer a formal research question by scientific methodologies. Instead, the artistic explication of my creative language and its context within experimental music effectively constitutes the aim of this investigation.[1]

Furthermore, the focus rests primarily on creative programming in experimental music – as opposed to digital art – because of the cur-

[1]This paper is located between being a pure research paper and a description of my creative practices, due to the artistic nature of the source material and the complementary theoretical/conceptual contextualization components. Hence, chapter 2 provides such academic substantiation, and chapter 3 focuses primarily on artistic applications. See section 1.2 for a description of appropriate methodologies.

rent context being that of a submission in partial fulfilment of the requirements for a Master of Music in (musical) Composition.

(In this document, the term *explication* refers broadly to "the action or process of stating or describing in detail."[2,3] The use of the term does not imply the associated formal methodologies in science, philosophy and literary criticism.)

1.1.3 Structure

The body of the paper is constructed in two main parts, as follows:

1. Artistic framework
   (a) definition of creative programming and its use cases
   (b) my historic and current position as a composer
   (c) creative context – i.e., outlining experimental music and digital art
2. Creative programming in practice
   (a) applications in experimental music
   (b) introduction to SuperCollider, the creative work's medium
   (c) creative work analysis (with code examples)
   (d) applications in digital art (with visual and code examples)

---

[2] *Oxford English Dictionary Online*, s.v. 'explication, (*n.*)', published December 2022, https://www-oed-com.ezproxy.uct.ac.za/view/Entry/66627?redirectedFrom=explication#eid.

[3] Please note that reference URLs to definitions in the Oxford English Dictionary (OED) Online will only work for those with access (i.e., university staff and students). Please contact the University of Cape Town Libraries for any queries in this regard (libraries@uct.ac.za).

Following the conclusion, there are appendices containing relevant written and visual material that was not included in the body. A glossary of terms follows thereafter, as well as a list of selected works.

1.2 Approach

1.2.1 Methodologies

On researching suitable methodologies for this explication, I found that *research-creation* and *artistic research* offered approaches that may be of use. Both terms have a history of debate among scholars when attempting to arrive at a universally accepted definition, particularly in the case of research-creation.

1.2.2 Research-Creation

I first came across the French term *recherche-création* when reading a master's thesis by a student from the music department at the University of Montreal. Supervised by Jean Piché, the student references Canadian scholar Serge Cardinal's writings on the topic[4] when outlining a research methodology; this initially proved useful when considering a format for

[4]Serge Cardinal, 'La recherche-création: une pensée audio-visuelle?', in *La recherche-création dans l'université du XXI^{e} siècle*, 80^{e} Congrès de l'ACFAS (Montreal, May 2012), https://www.creationsonore.ca/wp-content/uploads/2014/10/communications_serge-cardinal_la-recherche-creation.pdf.

my paper.[5] However, whilst searching for English-language literature that extends Cardinal's theories, I found a summary of research-creation by fellow Québécois scholars Sophie Stévance and Serge Lacasse:

> Research-creation is understood as (1) an *approach* applied to (3) [*sic*] an individual or multiple-agent (2) [*sic*] *project* combining (4) research *methods* and creative *practices* within a dynamic frame of (5) causal *interaction* (that is, each having a direct influence on the other), and leading to both (6) scholarly and artefactual *productions* (be they artistic or otherwise).[6]

The authors unpack each of these six points in detail with respect to its significance in defining research-creation. Stévance and Lacasse argue that in order to be considered true research-creation, an investigation must conform to all the listed criteria. After scrutinizing these assertions, I found that my intentions for this paper fulfilled most but not all of the requirements.

This explication satisfies the first four points, as it:

1. takes a methodological approach – i.e., it explicates my compositional language;
2. is accompanied by a project;
3. is carried out by an individual;
4. combines both research and creation.

However, it does not conform to the last two points, as it does not:

---

[5]Jean Philippe Pierre-Louis, '"Puntito": deux œuvres audiovisuelles portées par l'expérience de l'individualité' (master's thesis, Université de Montréal, 2018), 7–10, https://papyrus.bib.umontreal.ca/xmlui/handle/1866/22871.

[6]Sophie Stévance and Serge Lacasse, *Research Creation in Music and the Arts: Towards a Collaborative Interdiscipline* (London: Routledge, 2018), 123.

5. involve an explicit causal interaction between the two elements – the creative product is a realization inspired by existing knowledge and experiences, whilst the written component primarily serves to document and contextualize the nature of this inspiration;
6. result in a new scholarly production – rather, it explicates the context of a new artistic artefact.

The authors' unpacking of the fifth point was most striking to me. They stress the importance of a close interaction between research and creation, and insist that in research-creation neither the written nor the creative component could exist without the other.[7] Considering this, and after examining the text further, I realized that an alternative strategy would need to be considered; thus, I turned to artistic research.

1.2.3 Artistic Research

During my literature review, I read the work of German art school lecturer Friedrich von Borries, who proposes a definition of artistic research that uses a common thread found in several scholars' writings: "artistic research is research done by artists as or as part of their artistic practice."[8] Whilst this description is by no means exhaustive, it does provide a clear centre of gravity for other proposed ideas. Compared to

[7]Stévance and Lacasse, *Research Creation in Music and the Arts: Towards a Collaborative Interdiscipline*, 124–25.

[8]Friedrich von Borries, 'Artistic Research – Why and Wherefore?', *Journal of Science Communication* 14, no. 1 (2015): 1, C06, https://doi.org/10.22323/2.14010306.

the earlier exposition of research-creation, Borries's simple illustration of artistic research seemed like a more accessible point of departure, possibly because I find he takes an analytic rather than semantic approach.

Based on my readings of Angelika Boeck's work, artistic research seemed to be the most indicative of the direction in which I could approach my writing. Her outline constricts neither the creative aspect nor the obligatory presence of a research component. Boeck shares the preference for the idea of *art practice-based research* with a few colleagues (such as Borries and Graeme Sullivan)[9] and describes it as "research founded on a particular art practice."[10]

In elucidating this statement, she writes:

> . . . I consider it to be research founded on . . . the artist's activity in pursuit of a concrete question or a set of not yet clearly defined questions by employing artistic means and methods, and . . . the presentation of the process and/or outcome in the form of an artwork.[11]

This definition appealed to me in many ways and came as a welcome alternative. I find that Boeck defines the connection between art and research in a more organic and practical manner when compared with Stévance and Lacasse's research-creation.

---

[9]Graeme Sullivan, 'Art Practice as Research', in *Art Practice as Research: Inquiry in Visual Arts*, 2nd ed. (Thousand Oaks, CA: Sage, 2010), 95–120.

[10]Angelika Boeck, 'What is Artistic Research?', *w/k – Between Science & Art*: 2, published 25 February 2021, https://doi.org/10.55597/e6798.

[11]Boeck.

With regards to artistic research's methodology and theory, Boeck suggests that it is not unusual for these two aspects to be identified in hindsight through "reverse engineering";[12] that is, for the research to be contextualized within the work of other artists and scholars in a reflexive manner, after the artistic product is complete (or near completion). Artistic research also solves the issue of trying to arrive at the answer to a research question in written form before completing a work that attempts to do the same practically.

Boeck's text was instrumental throughout the formation of this explication, as I experienced difficulties when it came to being able to definitively delineate a conceptual framework and research methodology prior to, and even during, the creation process. Investigating the work of other artists/researchers whilst composing helped to clarify an artistic framework within which I was able to operate. Thus, the creative process had a large impact on shaping my research, and through researching I was able to enrich my artistic activities.

[12]Boeck, 'What is Artistic Research?'

1.3 Processes

1.3.1 Research Design

In this paper I aim to document creative programming, experimental music and digital art, and document their practical applications in my creative output (specifically, the electroacoustic work *Morphei Somnium*). The documentary procedure began with reading, listening, and gathering and organizing information on the topics mentioned above – of which creative programming and experimental music form the aesthetic foundation of the work in question. The piece was composed concurrent to this process and retrospectively provided subject matter for analysis in the third chapter; these parallel activities fell in line with Boeck's artistic research practice.

1.3.2 Methods

To explicate my artistic practice, I will contextualize my work by establishing the area in which the composition took place, including an introduction to creative programming and my musical background. This contextualization (chapter 2) will be extended to a discussion on the definition of experimental music, thereby introducing the methods used

and an analysis of *Morphei Somnium* in chapter 3. Similarly, chapter 2 will provide information on digital art, followed up (in chapter 3) by a discussion on the use of creative programming in digital visual art, including example sketches made in the Processing creative coding environment.

Chapter 2

Artistic Framework

2.1 Creative Programming/Coding

Creative programming (also referred to as *creative coding*) is the act of writing a computer program that, when executed, yields some artwork, be it visual, sonic/musical or multimedia in nature. Such works may be static (e.g., fixed media or installations), interactive (i.e., involving active external forces) or performed live. In *algorithmic music*, the process of writing the code and its execution is often considered to be as important as – sometimes more important than – the resultant art object.

Creative programming is a common mode of composition and performance in *experimental music*,[13] and programming environments such as Max and SuperCollider have been developed specifically for these purposes. These special software systems leverage computers' processing power and use high-level programming languages that translate creative ideas into instructions to be carried out by a machine. As a result, composers, researchers and performers are able to generate, transform and trigger both synthesized and pre-recorded sounds. In doing so, they are able to mimic traditional techniques in *musique concrète*, tape music and early electronic music (e.g., programming a delay line with gradual amplitude attenuation, thereby reproducing the original *tape echo*, among other potential time-based effects).

Digital art – a genre in which digital technology is utilized for creative purposes – is another common use case for creative coding. Processing[14] is an open-source creative coding environment based on the Java programming language in which visual sketches can be composed and has been used in a wide range of artistic contexts. One of Processing's offshoots is p5.js, which is a JavaScript implementation that allows for

[13]The multiplicity of elements in this genre are unpacked later; however, creative coding is present in most instances.

[14]https://processing.org/.

sketches to be coded and run in the web browser. P5 is frequently used in *live coding* performances and is often accompanied by music coded in a language such as Tidal Cycles, together forming a multimedia performance use case for creative coding.

2.2 Musical Background

French composer Olivier Messiaen's treatise *The Technique of My Musical Language*[15] was for a long period the cornerstone on which I based my Western art music composition practices. Messiaen's stylistic development served as a model for my own, as he was a key figure in extending classical harmony, traversing serialism and embracing new music technologies. Since my early days of composing, the language of French music has had a particular impact on my musical aesthetic, and music by Messiaen's students, spectralists Tristan Murail and Gérard Grisey proved to be a major source of inspiration. Their occupation with the sonic properties of instruments fascinated me and spurred on my exploration into more than the relatively one-dimensional pitch- and rhythm-based music that I became accustomed to. Admittedly, however,

[15]Olivier Messiaen, *The Technique of My Musical Language*, trans. from the French by John Satterfield (Paris: Alphonse-Leduc Éditions Musicales, 1944).

gaining insight from these composers came after largely skipping past the music of the 1950s and 1960s, a time when electronic and American experimental music became ubiquitous; nonetheless, I returned to these styles later in my composition journey.

Spectral music's influence on my acoustic style is evident in my 2020 pieces *Black and Violet* (after Wassily Kandinsky's eponymous 1923 painting) for solo viola and *Résonances* for orchestra. The former was inspired by two compositions from 1976: Murail's *C'est un jardin secret, [. . .] une source scellée* and Grisey's *Prologue* to the cycle *Les Espaces acoustiques* (1974–85). Both works are prime specimens of spectral music and feature inventive tone colours and textures, the latter featuring an optional electronic part.

During the composition of *Résonances*, Grisey's *Partiels* (1975) – the third piece in *Espaces* – was prominent due to the composer's derivation of the work from a spectral analysis of a trombone sounding E_2 (82.4 Hz), doubled by the contrabass section.[16] In a similar way, *Résonances* is characterized by its harmonic spectral and timbral variations. Murail's, Grisey's and my own pieces all entail focusing on instruments' (and in-

[16]Chris Arrell, 'The Music of Sound: An Analysis of "Partiels" by Gérard Grisey', in *Spectral World Musics: Proceedings of the Istanbul Spectral Music Conference*, ed. Robert Reigle and Paul Whitehead (Istanbul: Pan Yayıncılık, 2008), 319, https://issuu.com/chrisarrell/docs/arrellpartielsanalysis.

strument combinations') spectromorphological profile, which necessitates the awareness of factors such as instruments' overtone structure and their physical technical capabilities.[17]

Such sensitivity was something that I had come to learn, and discovering the possibilities in this new sound world led to my musical language evolving. Gravitating towards a more timbre-oriented style, I sought an aesthetic in which sound objects, composition methods and gestures are manipulated in a transformational idiom. The inclusion of new materials and methodologies – namely, electronic and recorded sounds and the use of computer programming – helped to extend my musical palette as I began to delve into experimental music.

2.3 Defining Experimental Music

2.3.1 Experimental vs. Avant-Garde

Preliminary readings into experimental music took me to *musique expérimentale*, which was defined by eminent French composer Pierre Schaeffer (founder of the Groupe de recherches musicales). Part of Schaeffer's definition includes his coined *musique concrète* – music that takes recorded

[17]Grove Music Online, s.v. 'Spectral music', by Julian Anderson, published in print 20 January 2001, https://doi.org/10.1093/gmo/9781561592630.article.50982.

sounds as source material in composition – which guided me towards musique expérimentale's modern association, *electroacoustic music*. Initially, I found Schaeffer's description of musique expérimentale (discussed later) to be sufficient; although later research steered me towards British composer Michael Nyman's writings. Reading the opening chapter of Nyman's book *Experimental Music: Cage and Beyond*, I discovered that I had opened up the proverbial "can of worms" for myself, given the myriad of other (disputed) definitions and related terms that have been cited in the past seventy-odd years.

Nyman's distinction between the *experimental* and the *avant-garde* was the first indicator of this conundrum. Whilst I relate modern experimental music to electroacoustic music, Nyman associates composers such as Pierre Boulez, Iannis Xenakis and Karlheinz Stockhausen with avant-garde music, despite all of them having composed what I understood to be electroacoustic music.[18,19] Confusing indeed, yet the grounds for my alignment is the obvious linguistic relationship between experimental music and musique expérimentale.[20]

---

[18]Michael Nyman, 'Towards (a Definition of) Experimental Music', in *Experimental Music: Cage and Beyond*, 2nd ed. (Cambridge: Cambridge University Press, 1999), 1.

[19]Nyman also lists Sylvano Bussotti and Harrison Birtwistle, neither of whom composed electroacoustic music.

[20]I will retain the latter's French spelling throughout to distinguish the two.

By contrast, Nyman places American composer John Cage at the centre of his argument and polarizes Cage and his experimental colleagues' approach to composition against the music of those "post-Renaissance avant-garde" composers mentioned above. Further, the fact that *Grove Music Online* (a major bibliographical resource in music scholarship) considers both Cage[21] and Stockhausen[22] to be leading figures in the post-war avant-garde adds more to this semantic discord, as it appears to directly contradict Nyman's original statement.

All things considered, however, Nyman's case for experimental music at least (if not his claimed divorce from the avant-garde) remains valid to date, with the genre's entry in the same reference repository providing a deep discussion on the topic.[23]

2.3.2 Composing Experimental Music

In *Experimental Music*, Nyman takes Cage's arguably most distinguished piece *4'33"* (1952) as subject matter when forming his explication.

---

[21]Grove Music Online, s.v. 'Cage, John', by James Pritchett, Laura Kuhn and Charles Hiroshi Garrett, published 10 July 2012, https://doi.org/10.1093/gmo/9781561592630.article.A2223954.

[22]Grove Music Online, s.v. 'Stockhausen, Karlheinz', by Richard Toop, last updated 22 October 2008, https://doi.org/10.1093/gmo/9781561592630.article.26808.

[23]Grove Music Online, s.v. 'Experimental music', by Cecilia Sun, published 10 July 2012, https://doi.org/10.1093/gmo/9781561592630.article.A2224296.

Nyman builds on the composer's separation of the activities of *composing*, *performing* and *listening* to music and discusses each at length.[24]

Writing about Cage's approach to composing experimental music, Nyman outlines experimental composers' esoteric viewpoint when it comes to notation, the composition process, the "uniqueness of the moment," musical identity and time itself. With regards to the former, Nyman highlights the change that experimental music brings to the notated score's prior established task. He claims that musical notation in experimental music no longer uses conventional symbols to represent sounds imagined by the composer for the performer to reproduce. Instead, Nyman states that notation is utilized to depict the expression of a concept, context or set of instructions governed by a collection of (as yet undefined) rules.[25]

A central idea that resonates throughout and beyond conceptual art from Cage's era is the *process*, which made its way through to creative coding and digital art. In procedural art, the composer's role becomes that of instruction- or rule-writer [the so-called "programmer"]. They provide an environment or means of calculation for the piece to be

[24]Nyman, 'Towards (a Definition of) Experimental Music', 3–26.

[25]Nyman, 3–4.

performed, a list of requirements (if any) [the "data"] and details about the process [the "algorithm"]. What follows is up to the performer and may range from the structured to the entirely haphazard.

Nyman names five process types: (1) *chance determination processes*, (2) *people processes*, (3) *contextual processes*, (4) *repetition processes* and (5) *electronic processes*.[26] When describing chance determination processes, Nyman references many of Cage's works including his 1951 piece *Music of Changes* for solo piano. This composition is a groundbreaking example of indeterminate music and implements hexagrams from the *I Ching* (ancient Chinese *Book of Changes*) to determine sounds, durations, dynamics, tempos and densities. Cage makes use of other articulating devices as well; for instance, paper imperfections (in his *Music for Piano* series, 1952–56) and star maps, seen in his *Atlas Eclipticalis* (1961–62).

In terms of *people processes*, the performers are asked to follow a set of instructions or suggestions at their leisure. Nyman cites composer Morton Feldman as originator of this technique, seen in his *Piece for Four Pianos* (1957). To explain the idea behind people processes, Nyman quotes British composer Michael Parsons, who discusses how the simultaneity

[26]Nyman, 'Towards (a Definition of) Experimental Music', 4–9.

and execution of the same material by multiple individuals results in "'unity' becoming 'multiplicity'."[27]

Contextual processes involve the musical discourse being governed by unpredictable conditions and variables that arise during performance. American composer Christian Wolff's work *Burdocks* (1970–71) and Paragraph 7 from Englishman Cornelius Cardew's *The Great Learning* (1971) are two prime demonstrations.

Reiteration to generate movement characterizes the *repetition process*. Minimalist Terry Riley's original piece *In C* from 1964 is a famous example that overlaps with people process ideologies. In the composition's performance directions, Riley writes:

> All performers play from the same page of 53 melodic patterns played in sequence.
>
> Any number of any kind of instruments can play. ...
>
> ... Patterns are to be played consecutively with each performer having the freedom to determine how many times he or she will repeat each pattern before moving on to the next. There is no fixed rule as to the number of repetitions a pattern may have ...
>
> ... Each pattern can be played in unison or canonically in any alignment with itself or with its neighboring patterns. One of the joys of *IN C* is the interaction of the players in polyrhythmic combinations that spontaneously arise between patterns. Some quite fantastic shapes will arise and disintegrate as the group moves through the piece when it is properly played.[28]

---

[27]Nyman, 'Towards (a Definition of) Experimental Music', 6.

[28]Terry Riley, *In C* (Tucson, AZ: Celestial Harmonies, 1989).

Finally, Nyman references computer music pioneer David Behrman's *Runthrough* (1967–68) for the *electronic process*. In this primordial illustration of live interactive electronic music, all that is required from the performers is the ability to turn dials, flip switches and operate handheld torches – professional performance experience is not mandatory.[29,30] Conceptually, this style crosses over into digital art and is visible in many modern multimedia installations and other interdisciplinary works that may include creative coding in their realization.

2.3.3 Towards *Musique expérimentale*

As discovered in my initial inquiry into experimental music, Pierre Schaeffer is the first to attempt to crystallize the meaning of *musique expérimentale*. The composer assimilates his (self-proclaimed) avant-garde contemporaries' concepts and proposes a description of the sum of their multiple coexisting and overlapping activities in a special issue of *La Revue musicale*. In his publication, Schaeffer combines musique concrète with tape music, German *elektronische Musik* and so-called

[29]Nyman, 'Towards (a Definition of) Experimental Music', 6–9.

[30]David Behrman, Crys Cole and Cleek Schrey, *Runthrough*, videography by Yiyang Cao, audio recorded by Bob Bellerue, ed. James Emrick, recorded live 21 July 2018, https://issueprojectroom.org/video/david-behrman-runthrough.

exotic music (referring mainly to the music of US-based composers Cage and Edgard Varèse).[31]

Scholar Carlos Palombini provides an exegesis of Schaeffer's issue in which he translates the composer's thoughts. I have endeavoured to present these remarks from a more positive perspective below.[32]

Schaeffer suggests the following:

1. that electronically synthesized sounds are valid compositional artefacts, useful in and of themselves, and should not only mimic conventional acoustic sonorities;
2. that the use of prepared and "exotic" instruments, and the exploration of their unique timbres should be supported beyond a traditional Classical approach;
3. that tape manipulation techniques (e.g., looping, time-stretching, splicing, layering and retrogression) and artificial effects (e.g., filtering and reverberation) should be recognized as valid compositional devices;
4. that sampled sounds and complex sound objects created from sounds or noises (musical or not), recorded and transformed using these techniques should be appreciated;
5. that music's spatial and gestural characteristics are important, and should be leveraged in composition in both live and recorded settings;
6. that complex and modern sonorities that extend beyond acoustic note-based music should be accepted, even if impossible to notate;

---

[31]Pierre Schaeffer, ed., 'Vers une musique expérimentale', special issue, *La Revue musicale* (Paris), no. 236 (1957): 11–13.

[32]These are my own interpretations of statements that Schaeffer makes, where he negatively criticizes his contemporary "traditional" composers' ideas. In the original journal, Schaeffer discusses what he considers to be misconceptions; I give a personal abstraction of Schaeffer's remarks based on his implied counterarguments.

7. that there should be a new conceptualization of music, and that during the composition process the composer should be encouraged to experiment with sounds and consider the psychophysiological implications;
8. that a purely theoretical approach to composition is less effective than one informed by experience when attempting to evoke new sounds from an instrument;
9. that a work should not be considered to be fully realized; the listener should be engaged in the work's (re)creation, and not solely be a passive receptor.[33]

Systematically extending Schaeffer's definition of *musique expérimentale* provides a clearer understanding of what this genre entails.

2.3.4 Towards Electroacoustic Music

To arrive at a detailed description of *electroacoustic music* – the resting point for my combined interpretation of *musique concrète*, electronic music, tape music and other experimental music forms – I have attempted to concisely break down Schaeffer's first five suggestions (discussed above) by providing examples of applications.[34]

Electronic music – compositions made with electronically synthesized sounds, known classically as *elektronische Musik* – has its primary origins

[33]Carlos Palombini, 'Pierre Schaeffer, 1953: Towards an Experimental Music', *Music & Letters* 74, no. 4 (1993): 551–52, https://www.jstor.org/stable/737576.

[34]Prefacing the last four remarks on the list, Schaeffer states that he adds these for inclusivity's sake – i.e., his primary concern is to define musique expérimentale with respect to methods of producing sound. These last points are predominantly criticisms of certain compositional practices at the time, and he does not necessarily wish to reinforce them at the risk of inciting controversy. Seeing as the examples relating to the first five points are sufficient to determine a comprehensive definition of electroacoustic music, I shall not delve deeper for the sake of brevity. See Palombini, 552.

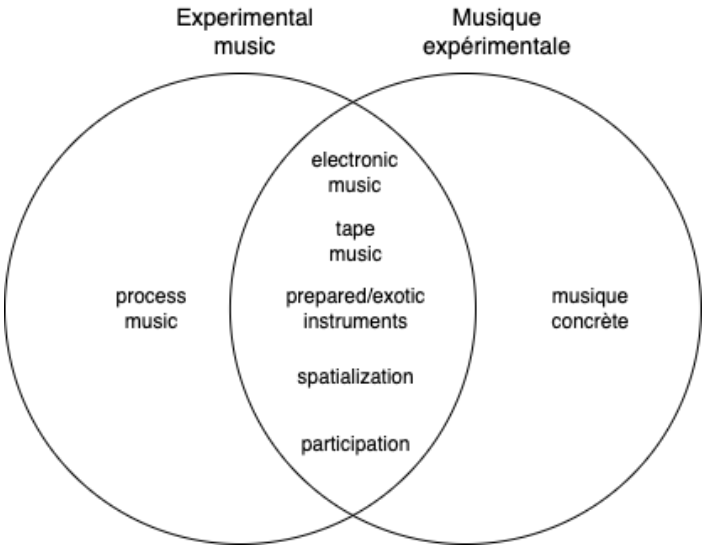


Figure 1. Experimental music vs. *musique expérimentale.*

at the Northwest German Broadcasting (NWDR) studios in Cologne. The first performance of an artistic electronic musical work was held at a music festival in the same city in 1953, and pioneer Karlheinz Stockhausen received the first commission of such a work the following year.[35]

Stockhausen worked with three major elements in sound synthesis: sine wave generators, white noise generators and electric filters.[36] His experiments revealed the complex nature of sound – its balance between pitch, timbre, intensity and duration. Such parameters morph over time

[35]Toshiro Mayuzumi, 'The Principles of Electronic Music', trans. Cathy L. Cox, *Contemporary Music Review* 37, nos. 1–2 (2018): 103, https://doi.org/10.1080/07494467.2018.1453344.

[36]Robin Maconie, *Stockhausen: On Music* (London: Marion Boyars, 1989), 90.

in a continuous, fluid manner (or remain static) – a concept that shaped Stockhausen's approach to composition.[37]

Edgard Varèse, who coined the term *organized sound*, is another key figure in early electroacoustic music.[38] Varèse aimed to add electronic media to the common composition toolkit in order to supplement existing acoustic instruments, which he did not necessarily wish to replace; he claimed that electronics should serve an additive, not destructive purpose.[39] The composer is well known for his desire for the creation of and experimentation with new instruments; namely, the use of sirens and the lion's roar in his renowned *Ionisation* for thirteen percussionists (1929–31) and the theremin in his *Ecuatorial* for large ensemble (1932–34).

Similarly, John Cage is associated with exotic instruments. His *Credo in Us* from 1942 is scored for four players with various objects, including tin cans, an electric buzzer, a radio and a phonograph.[40] In addition, Cage is famed for composing for prepared piano and asks for screws,

[37]Andrew Knight-Hill, 'Electroacoustic Music: An Art of Sound', in *Foundations in Sound Design for Linear Media: A Multidisciplinary Approach*, ed. Michael Filimowicz (New York: Routledge, 2019), 311, https://doi.org/10.4324/9781315106335.

[38]Richard Franko Goldman, 'review of *Varèse: Ionisation; Density 21.5; Intégrales; Octandre; Hyperprism; Poème Electronique* by Robert Craft & Varèse', *The Musical Quarterly* 47, no. 1 (1961): 133, https://www.jstor.org/stable/740554.

[39]Edgard Varèse and Chou Wen-chung, 'The Liberation of Sound', *Perspectives of New Music* 5, no. 1 (1966): 15, https://doi.org/10.2307/832385.

[40]John Cage, *Credo in Us* (New York: C. F. Peters, 1962).

bolts and weather stripping to be placed onto the piano's strings in his piece *Bacchanale*, completed in 1940.[41]

Observing Schaeffer's compositional approaches helps to expand on his third and fourth points regarding tape manipulation and musique concrète (which relate to the use of recorded sounds that are processed in various ways to create complex sound objects). Schaeffer employed the techniques highlighted on the list (looping, time-stretching, splicing, layering and retrogression) to create a type of "pure sound," where sound is liberated from any association with its source. He discovered that it was possible to drastically alter a sound's identity by modifying or removing parts of its dynamic envelope. The composer also noticed that continuously looping a sound can transform it into a new abstract sound object.[42]

Schaeffer's *Cinq études de bruits* (1948) are a distinguished collection of compositions in electroacoustic music's history, as they represent the birth of musique concrète and feature manipulated recordings of trains, toy tops, percussion instrument, the piano, saucepans, speech and more. Following these, Schaeffer worked with compatriot Pierre Henry

---

[41]John Cage, *Bacchanale* (New York: C. F. Peters, 1960).

[42]Knight-Hill, 'Electroacoustic Music: An Art of Sound', 308.

on the first major work of musique concrète, *Symphonie pour un homme seul* (1950) at the French radio network RTF, and the duo founded the Groupe de recherche de musique concrète (GRMC) the year thereafter.[43]

In Stockhausen's lecture 'The Four Criteria of Electronic Music', he presented his theory of *Unified Time Structuring*,[44] where he observed a change in musical quality when altering the playback speed of a recorded sound. Stockhausen's experiments led him to discover that rhythms and forms have the tendency to become timbres when sped up, and timbres can become rhythmic oscillations when slowed down. By applying this knowledge, Stockhausen determined that there exists a relationship between timbre, rhythm and form; this enormously changed the way we conceptualize music and composition.[45]

Stockhausen also believed that space is an essential component in musical composition and appreciation. Going back centuries, space as a musical device can be witnessed in Italian composer Giovanni Gabrieli's antiphonal sacred music, where choirs and instrument groups are placed at different points in St Mark's Basilica to achieve an immersive spatial

---

[43]Nicole V. Gagné, *Historical Dictionary of Modern and Contemporary Classical Music*, 2nd ed. (Lanham, MD: Rowman & Littlefield, 2019), 310.

[44]Karlheinz Stockhausen, *Stockhausen – Four Criteria of Electronic Music*, playlist (1972), selection of YouTube videos, published 11 April 2013, https://www.youtube.com/playlist?list=PLRBdTyZ76lvAFOtZvocPjpRVTL6htJzoP.

[45]Maconie, *Stockhausen: On Music*, 91–96.

effect. Dissatisfied with the fact that, at the time, most recorded classical music was monophonic, Stockhausen developed new multichannel configurations.[46] Such innovations can be heard in his revolutionary four-track electronic works *Gesang der Jünglinge* (1955–56) and *Kontakte* (1958–60) – both realized at the NWDR studios.

Likewise, Varèse recognized that physical space was an important part of the musical experience and presented his *Poème électronique* as part of a multimedia installation at the Philips Pavilion at the 1958 Brussels World's Fair. *Poème électronique* was Varèse's only purely electroacoustic composition and is the earliest documented example of electroacoustic music integrated with images in an audiovisual setting.[47]

To combine all of these concepts, composer and editor of *Organised Sound*, Leigh Landy, provides a comprehensive breakdown of common terms cited in the world of electroacoustic music and attempts to end the "terminology debate." He defines *organized sound*, *sonic art*, *sound art*, *radiophonic/radio art*, *electronic music*, *computer music*, *electronica* and at last, *electroacoustic music* in the introduction to his

[46]Knight-Hill, 'Electroacoustic Music: An Art of Sound', 313.

[47]Vincenzo Lombardo et al., 'A Virtual-Reality Reconstruction of "Poème Électronique" Based on Philological Research', *Computer Music Journal* 33, no. 2 (2009): 24, https://muse.jhu.edu/article/266409.

book *Understanding the Art of Sound Production*.[48] Landy further compresses the elements of musique expérimentale and specifies additional technicalities in his own of four proposed definitions:

> . . . any music in which electricity has had some involvement in sound registration and/or production other than that of simple microphone recording or amplification.[49]

Landy's summation is short, but I find the third listed description by Simon Emmerson and Denis Smalley to be the most insightful and complete:

> Music in which electronic technology, now primarily computer-based, is used to access, generate, explore and configure sound materials, and in which loudspeakers are the prime medium of transmission. There are two main genres. Acousmatic music is intended for loudspeaker listening and exists only in recorded form (tape, compact disc, computer storage). In live electronic music the technology is used to generate, transform or trigger sounds (or a combination of these) in the act of performance; this may include generating sound with voices and traditional instruments, electroacoustic instruments, or other devices and controls linked to computer-based systems. Both genres depend on loudspeaker transmission, and an electroacoustic work can combine acousmatic and live elements.[50]

---

[48]Leigh Landy, *Understanding the Art of Sound Organization* (Cambridge, MA: MIT Press, 2007), 9–19.

[49]Leigh Landy, 'Reviewing the Musicology of Electroacoustic Music: A Plea for Greater Triangulation', *Organised Sound* 4, no. 1 (1999): 61, https://doi.org/10.1017/S1355771899001077.

[50]Grove Music Online, s.v. 'Electro-acoustic music', by Simon Emmerson and Denis Smalley, published in print 20 January 2001, https://doi.org/10.1093/gmo/9781561592630.article.08695.

2.4 Visual Context

2.4.1 Digital Art

Digital art's history is notoriously convoluted and diverse, with much cross-pollination between disciplines. Scholar Christiane Paul has written extensively on the genre and its meaning, and frames digital art as an "art form that uses digital technology".[51] Paul is quick to criticize other terms that have been used to describe it since its inception in the late twentieth century. She particularly disputes the term *new media art* – art that is "digital born," created, stored and distributed via digital technologies.[52] This allegedly problematic phrase is regularly used interchangeably with *digital art*, but Paul claims that it is only a subset of the genre.[53]

In her volume, Paul goes to great lengths to specify what qualifies as digital art and what does not. She provides two categories: (1) art that uses digital technology as a tool to produce traditional art forms (e.g., photography, painting and sculpture) and (2) *digital-born art* that yields a less material art object – often software based – and leverages

[51]Christiane Paul, 'New Media', in *Encyclopedia of Aesthetics*, 2nd ed., ed. Michael Kelly (Oxford University Press, 2014), https://doi.org/10.1093/acref/9780199747108.001.0001.

[52]Paul.

[53]Christiane Paul, 'Introduction: From Digital to Post-digital – Evolutions of an Art Form', in *A Companion to Digital Art* (Hoboken, NJ: John Wiley & Sons, 2016), 1.

technology's own digital traits (for example, its generative capabilities). Art forms that are digital born can be described as computational, collaborative, participatory, performative, process-oriented, time-based or dynamic, among other descriptions.[54]

Such forms may be interdisciplinary and contain fixed (i.e., static) (audio)visual media, have algorithmic or generative properties or a live demonstration aspect, and may include audience participation. Overall, digital art encompasses a large portion of postmodern and contemporary art.

2.4.2 Generative Art

Generative art is not exclusively a subtype of digital art – or computer art, for that matter – as its concepts and procedures greatly precede and extend beyond a digital technology framework; still, it remains a generally accessible and mainstream genre. Philip Galanter has written extensively on generative art and, like Christiane Paul, provides a lucid definition:

> Generative art refers to any art practice in which the artist cedes control to a system with functional autonomy that contributes to, or results in, a completed work of art. Systems may include natural language instructions, biological or chemical processes, computer programs, machines,

[54]Paul, 'Introduction: From Digital to Post-digital – Evolutions of an Art Form', 2.

> self-organizing materials, mathematical operations, and other procedural inventions.[55]

Operative here is the notion of *functional autonomy*. Galanter states that rule-based art (or *algorithmic art*) does not necessarily constitute generative art, despite there sometimes being a misconstrued equation of the two. He believes that to qualify at a technical level, the creator must explicitly hand over control to a functionally autonomous system. That is to say, if the system relies on its creator's intervention for any duration during the execution, then it is not – per Galanter's definition – autonomous.

Being a procedural art form related to conceptual art, generative art reflects experimental music, which is apparent when referring back to Nyman's summary of composition processes discussed earlier. On inspection, there is a recognizable link when considering the idea of an external system that independently follows a set of rules or instructions written by an artist or composer [the "author"]. The executor or performer's behaviour once the execution or performance has begun – by their autonomous nature – is ultimately out of the author's control. Thus, in generative art, the process's rendering is more important as a

[55]Philip Galanter, 'What is Complexism? Generative Art and the Cultures of Science and the Humanities', in *Proceedings of the 11th Generative Art Conference* [GA2008] (Milan, December 2008), 154, https://generativeart.com/on/cic/papersga2008/13.pdf.

concept than the resultant art object's aesthetic. Similarly, in experimental music, the performer's interpretation of the concepts and rules that the composer gives is extremely important when evaluating the entire work.

Galanter believes that *effective complexity* is central to generative art's identity. This idea describes how "high degrees of order and disorder both create simple systems, and complex systems exhibit a mixture of both order and disorder."[56]

Highly ordered generative art includes art that uses symmetry, tiling and patterns based on predetermined rules; for instance, those found in Islamic art and architecture, or conceptual art founder Sol LeWitt's *Wall Drawings* (produced throughout the artist's career). LeWitt sets a series of preset, calculated instructions or diagrams for others (i.e., independent, self-acting systems) to follow in many of these works. Each new rendition yields a unique result based on the combined individual responses, with an ordered level of complexity. This harks back to Nyman's people processes, demonstrated in Feldman's *Piece for Four Pianos*, where four players read from the same score and each performs at their own speed.

---

[56]Philip Galanter, 'Generative Art Theory', in *A Companion to Digital Art*, ed. Christiane Paul (Hoboken, NJ: John Wiley & Sons, 2016), 157, https://doi.org/10.1002/9781118475249.ch5.

Highly disordered generative art involves elements of randomization and chance operations. As discussed earlier, experimental composer John Cage is well known for the latter (recall his *Music of Changes*). Additionally, American writer and visual artist William Burroughs popularized the aleatoric *cut-up technique* in his literary works, such as the *Nova Trilogy* (1961–64).

In the realm of computer graphics, Frieder Nake employed pseudo-random number generators in his 1965 *Polygon Drawings*;[57] meanwhile, creative coding environments such as Max and SuperCollider allow randomization to be integrated into the music-making process.[58]

Lastly, Galanter defines complex systems by referencing ideas such as self-organizing systems, non-linearity, synergy and chaos theory. He mentions fractals – self-similar objects that can be used to generate abstract patterns, often found in nature – as an example of complex systems in generative art. The scholar proceeds to list audio and visual implementations of complex systems that include Lindemeyer systems (L-systems) – which are utilized in 2-D and 3-D renderings – chaotic

[57]Galanter, 'Generative Art Theory', 158–61.

[58]Galanter, 148.

systems (used in video feedback loops) and AI art (art that is created using artificial intelligence).[59]

Generative art has a number of crossovers with creative coding in both experimental music and digital art; these are discussed in the next chapter.

2.4.3 Note on Artificial Intelligence (AI) in Art and Music

Whilst I recognize the recent explosion of AI in the creative space, I have consciously chosen not to include explicit research regarding such technology in this explication. This is in the interest of not extending the length of the paper further, as I would need to introduce and discuss concepts from philosophy (including classical aesthetics, ethics, epistemology and metaphysics) and computational creativity, which extend beyond the scope of my intended investigation. Furthermore, at this point, I have not implemented AI in any of my creative practices; thus, I would be unable to genuinely demonstrate AI art or music at an individual practical level.

I can, however, redirect the interested reader to the entry on 'AARON' in the *Encyclopedia of Artificial Intelligence: The Past, Present, and*

[59]Galanter, 'Generative Art Theory', 162–64.

Future of AI. AARON is software that was developed by British artist Harold Cohen to create paintings and is an early application of AI in art.[60] For literature on AI in music, I would suggest the article 'A Functional Taxonomy of Music Generation Systems' as a starting point.[61]

[60]Mihály Héder, 'AARON', in *Encyclopedia of Artificial Intelligence: The Past, Present, and Future of AI*, ed. Philip L. Frana and Michael J. Klein (Santa Barbara, CA: ABC-CLIO, 2021), 1–2.

[61]Dorien Herremans, Chuan Ching-hua and Elaine Chew, 'A Functional Taxonomy of Music Generation Systems', *ACM Computing Surveys* 50, no. 5 (2017): 69, https://doi.org/10.1145/3108242.

Chapter 3

Creative Programming in Practice

3.1 Experimental Music

3.1.1 Applications in Electroacoustic Music

Creative coding's real-world applications become apparent when one considers Emmerson and Smalley's definition of *electroacoustic music* (see chapter 2).[62] The scholars highlight the fact that most of today's electroacoustic music is computer based, which functionally correlates with creative coding, as code written in a creative coding language is generally executed on a computer or similar device. Electroacoustic music was originally composed using analog synthesizers, recording devices,

[62] Grove Music Online.

tape machines and the like – computers can replace just about all of these. Creative coding environments such as Max and SuperCollider provide a mechanism to store existing sounds (in *buffers*), and create (software synthesizers and generators) and manipulate sound objects, leveraging the environments' built-in *objects* and *messages*.[63]

Continuing with creative coding's utility in electroacoustic music, acousmatic music composed using a creative coding language can be played back through loudspeakers by accessing the computer's data storage, and audio driver. Many creative coding environments use their own built-in synthesis engine; although, some stand-alone languages leverage the engine from another source.[64]

In live settings, these synthesis engines are given instructions to generate, trigger and transform sounds natively or using an external interface that communicates with the engine via some network protocol. Analog instruments (including the voice) and other digital or hybrid instruments may likewise be integrated into performances. This can all be achieved within a single programming environment, exemplifying creative

[63]Max is a visual programming environment that uses patching, where objects and messages are interactive UI elements, whereas in SuperCollider, the concepts are text based.

[64]For example, Tidal Cycles communicates with SuperCollider's audio server.

coding's flexibility and potentiality in the (re)creation of electroacoustic music.

3.1.2 Process Music

Widening the net, process music exhibits creative coding's generative qualities. Code written in a creative coding language acts as the score for a work – i.e., a predetermined set of instructions (or an algorithm) that is executed by an independent entity. Such code may define a process or pattern wherein the sound itself or one of its qualities is not determined until execution, similar to Cage's indeterminate practices. Moreover, it is not uncommon to see code that gives directions for something to happen based on a previous event's completion; for example, a pulse that triggers a gate only after a note with an arbitrary duration has stopped playing.

Making use of random number generators and triggers – fundamental elements of any creative coding language – in performance or composition reflects Galanter's highly disordered generative art, where the computer makes unpredictable, split-second decisions to determine a sound's frequency, intensity and duration. Generative music that is composed in a creative coding language is often set to run indefinitely and may include

randomized triggers, pattern sequencers and LFOs. The individual modules interact with and modulate each other as if the patch were "playing itself," akin to an analog modular synthesizer.

3.2 SuperCollider

3.2.1 Hello World

Using computers and programming as a means of artistic expression has become an integral part of my compositional approach – electronic or otherwise. After being introduced to creative coding in 2020, my creative toolkit has been significantly augmented. Electronically synthesized and generated sounds, pre-recorded material and live signal processing have all been added to my musical palette's instrumental and vocal colour groups, presenting a broadened sonic world of creative potential.

Software engineer James McCartney developed 'SuperCollider: A New Real Time Synthesis Language' and shared its creation in a paper at the 1996 International Computer Music Conference in Hong Kong.[65] The software was subsequently released in 2002 as free and open source under the GNU General Public Licence, meaning that anyone can download

[65] James McCartney, 'SuperCollider: A New Real Time Synthesis Language', in *Proceedings of the International Computer Music Conference* [ICMC'96] (Hong Kong, August 1996), 257–58, https://quod.lib.umich.edu/i/icmc/bbp2372.1996.078/1.

the code, edit it and suggest changes to potentially be made to the codebase.

SuperCollider's complete programming environment consists of three elements: a real-time audio server (scsynth), a high-level interpreted language (sclang) and a text editor with built-in documentation. Within the SuperCollider IDE, it is possible to write source code in sclang (which acts as the client) to send messages to the server (the synthesis engine) via the Open Sound Control (OSC) network protocol. The server then processes these commands and organizes the nodes, buffers and signal routing to generate and transform sounds.

The SuperCollider language[66] is syntactically straightforward and extensively documented with detailed explanations and useful examples; thus, learning and navigating the environment is largely accessible to new programmers. Useful in both composition and performance, SuperCollider's source, filter, control and panning unit generators (UGens) allow for numerous, flexible sound design mechanisms, including sound generation, signal processing and event structuring.

[66]Henceforth referring to sclang.

3.2.2 SynthDefs and Object-Oriented Programming

SuperCollider is an object-oriented programming language. This programming paradigm uses *objects* (which store data) and *methods*,[67] which can be executed (or "called") on or with that data. Everything in SuperCollider is an object, and objects are organized into *classes*. The `UGen` class is an abstraction of a unit generator and the `Synth`[68] class represents a collection of UGens that work together to generate an output.[69] UGens are the building blocks of any SuperCollider program.

A SynthDef (client-side synthesis definition) is required in order to instantiate a new synth object on the server. Synth objects can create or process sound, or produce a control signal that drives other synths. SynthDefs can be thought of as templates that identify specific synth objects and contain information about various UGens in the form of a Function.[70]

---

[67]Methods are also referred to as *messages* in SuperCollider.

[68]Note the use of the uppercase *s*, which will be used when referring to the Synth class itself. The lowercase *s* has a generic meaning that may refer to synth objects or later, instruments in the Laptop 2 part of the score for *Morphei Somnium*.

[69]James McCartney, 'Rethinking the Computer Music Language: SuperCollider', *Computer Music Journal* 26, no. 4 (2002): 62, https://doi.org/10.1162/014892602320991383.

[70]Note also the uppercase *f* in Function, which, in the context of SuperCollider, distinguishes itself from the common *function* in functional (i.e., non-OOP) languages.

Functions in SuperCollider are objects that contain reusable code and can be used to generate sound when an audio-creating method like `play()` is called on them.[71] A SynthDef's Function may contain *parameters*, which are values that are specified at the beginning of the Function and then read by the UGens later in the body. Parameters in a Function are often initialized with starting values (arguments), which may be substituted when instantiating a new synth from the SynthDef.

Listing 1 shows a SynthDef called `micIn`, which is the blueprint for a synth that takes in a signal from the first audio bus in SuperCollider. Assuming that a microphone is connected to the current audio device's first input, the microphone's signal is read into the SynthDef's Function (the code surrounded by curly braces) through a `SoundIn` UGen on line 10 (per the Function's argument `in = 0` on line 5).[72] `micIn`'s Function has eight more arguments that are read later in the body.

Code defining a ring modulation effect is seen from lines 13 to 18. An interpolating sine wave oscillator (with no phase offset) modulates the input signal's amplitude, and the result is stored in the variable `ringMod`. The `ar()` method called on the `SinOsc` UGen creates a new

[71] As a habit from programming in other languages, I add parentheses to all method names (regardless of whether or not they receive arguments) to distinguish them from data variables. This notation is not compulsory in SuperCollider; in fact, it is rare.

[72] Most common programming languages start counting at zero, and not one.

```
// MORPHEI SOMNIUM: LAPTOP 1

SynthDef(\micIn, {
    // Function parameters with initialized arguments
    | in = 0, rmFreq = 700, rmBlend = 0, xpos = 0, ypos = 0,
    amp = 1, pitch_shift_out = ~bus[\empty],
    delay_out = ~bus[\empty], out = 0 |

    // microphone input signal (carrier)
    var sig = SoundIn.ar(in);

    // ring modulator
    var ringMod = sig * SinOsc.ar([
        rmFreq,
        rmFreq * 0.9,
        rmFreq * 0.7,
        rmFreq * 0.2
        ]);

    // ring modulation/direct signal mix
    sig = XFade2.ar(sig, ringMod, rmBlend * 2 - 1);

    // quadraphonic panning
    sig = Pan4.ar(sig, xpos, ypos, amp);

    // outputs
    Out.ar(pitch_shift_out, sig);
    Out.ar(delay_out, sig);
    Out.ar(out, sig);
}).add();
```

Listing 1. Microphone input SynthDef.

`SinOsc` object and has four parameters: frequency (in Hertz), phase (in radians), a multiplier value and an added value (both as floating-point numbers).

In this UGen, just the frequency argument is given – as an array (surrounded by square brackets). Inputting multiple values for the

frequency parameter creates multiple oscillators, where the first one cycles at 700 Hz. The remaining three oscillators' frequencies are reduced by multiplying the original 700 Hz by values less than one. As a result, if these oscillators are added to the dry signal, they will modulate the carrier (microphone) signal at four discrete frequencies simultaneously.

On line 21, an `XFade2` UGen blends (cross-fades) the direct and modulated signals, with no ring modulation by default. After that, the mixed signal is panned across four channels using the `Pan4` UGen – left to right (`xpos`) and back to front (`ypos`). The default setting is (0, 0), which places the signal in the centre of both axes, since the scale is -1 (left/back) to 1 (right/front).

Finally, the signal is sent to three separate output buses using the `Out` UGen. The first two outputs go to an empty bus, but when changed can be routed to dedicated pitch-shifting and delay effects buses by instantiating a custom synth from the `micIn` SynthDef. The `add()` method on line 30 adds the SynthDef to the server, so that it can be used to create a new synth.

3.3 *Morphei Somnium* Analysis: Part I

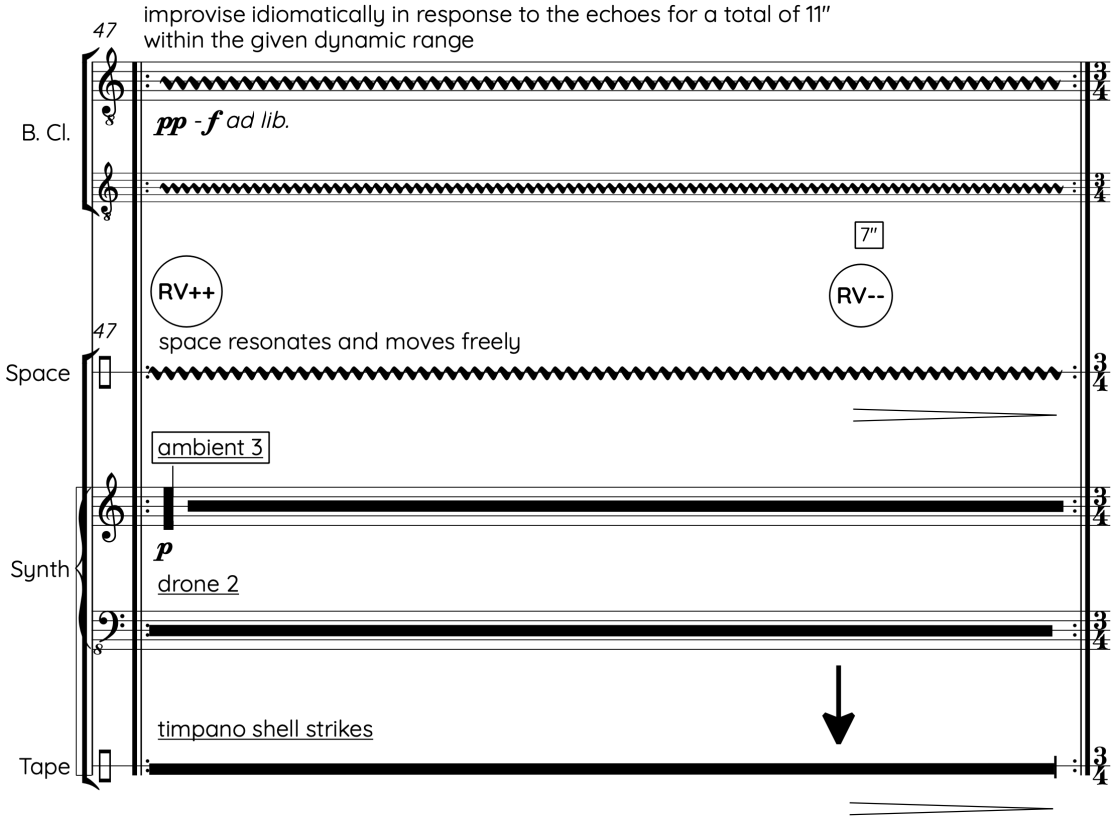
3.3.1 Structure and Bass Clarinet

Morphei Somnium ('Morpheus's Dream') for bass clarinet and two laptops is my first full-length electroacoustic composition and was composed in 2021 for a competition at the University of Pretoria School of the Arts. There are three elements:

- Bass Clarinet: primary acoustic source, featuring mainly extended techniques
- Laptop 1 (Space): controls effects applied to the bass clarinet
- Laptop 2 (Synth and Tape): generates, triggers and manipulates electronically synthesized and pre-recorded material

The bass clarinettist leads the production for the most part, with the two laptop performers following cues from the score. At the same time, there are sections in the piece where the instrumentalist must be aware of the activity in the laptop parts, particularly in laptop 1 when delay effects are applied. Broadly speaking, the work's rhythmic feel is relatively free and none of the parts follows a click track; instead, the parts interact with and against each other in fashioning the sonic landscape.

Figure 2. Free improvisatory section (tutti), bar 47.

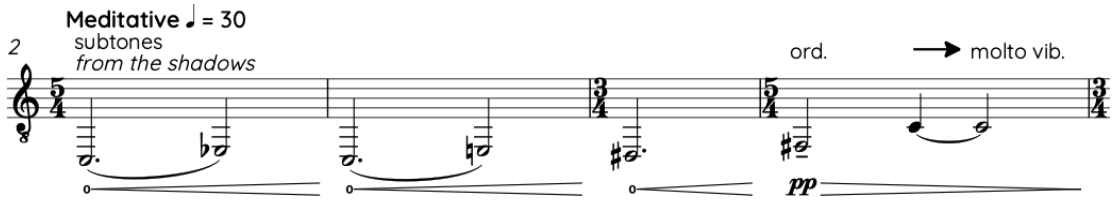


Both laptops run their own instance of SuperCollider, leveraging its synthesis engine to generate and process sounds. The performers either alter the source code directly in the IDE or interact with the server through a custom visual TouchOSC interface that was designed especially for the piece (which requires no programming experience and is the suggested method).[73] This interface acts as a controller for the audio server by sending and receiving OSC messages.

[73]TouchOSC can run on computers (desktop or laptop) or wireless devices, such as mobile phones and tablets. For this piece, I have programmed a computer and a tablet interface; however, more can be quite easily created using free software (see https://hexler.net/touchosc).

The acoustic textures and gestures that the bass clarinet provides can be split into two groups: standard and extended. The former describes common techniques that any semi-professional clarinettist would be familiar with; namely, playing with varying vibrato intensities and at extremely soft dynamics (subtones and *crescendo dal/diminuendo a niente*).[74]

Figure 3. Standard bass clarinet techniques, bars 2–5.

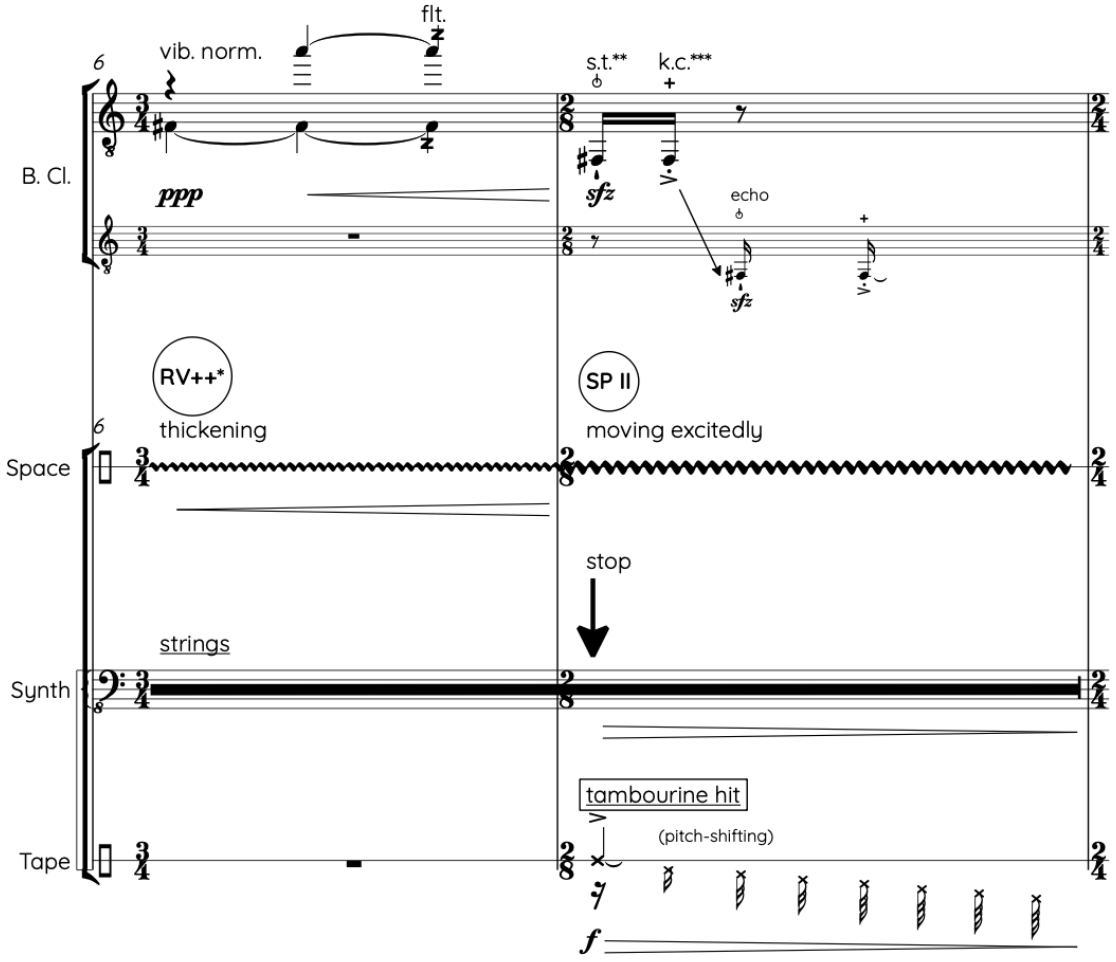


Extended techniques, the second group, make up the majority of the bass clarinet part. These techniques include flutter tongue, harmonic glissandi, slap tongue, key clicks, timbral trills (*bisbigliando*), air noises and multiphonics.

To elevate these techniques, the first laptop performer's primary purpose is to apply ring modulation, pitch-shifting and time-based effects to the bass clarinet in real time. Notated cues in the score are labelled according to predetermined values or *spaces* (indicated with the letters *sp* and a roman numeral or 0), which have been temporarily hard-coded in SuperCollider. The score gives guidelines for these spaces'

[74]The score is always transposed.

Figure 4. Extended techniques with effects; strings; tape, bars 6–7.



parameters (see figure 4).[75] Consulting these, the performer listens to the bass clarinet and interacts musically, by adjusting the settings at will.

3.3.2 Spaces: Meaning

Morphei Somnium's conceptualization began with the idea of a dream – Morpheus being the ancient Greek god of sleep and dreams, and *somnium* being the Latin word for "dream." The Greek word *morfí* means "shape"

[75]In the score, "RV++" indicates that the amount of reverb should be increased by an arbitrary amount.

or "form", and the English translation, *morph* – when used as a verb – means "to change (a person or thing) into something different, esp. th[r]ough a series of rapid transitional phases."[76] This lends itself to the image of a dreamlike landscape that morphs over time, and so 'Morpheus's Dream' can be thought of as "the dream of one who changes (something)." In the composition's case, that *something* is the landscape or space itself.

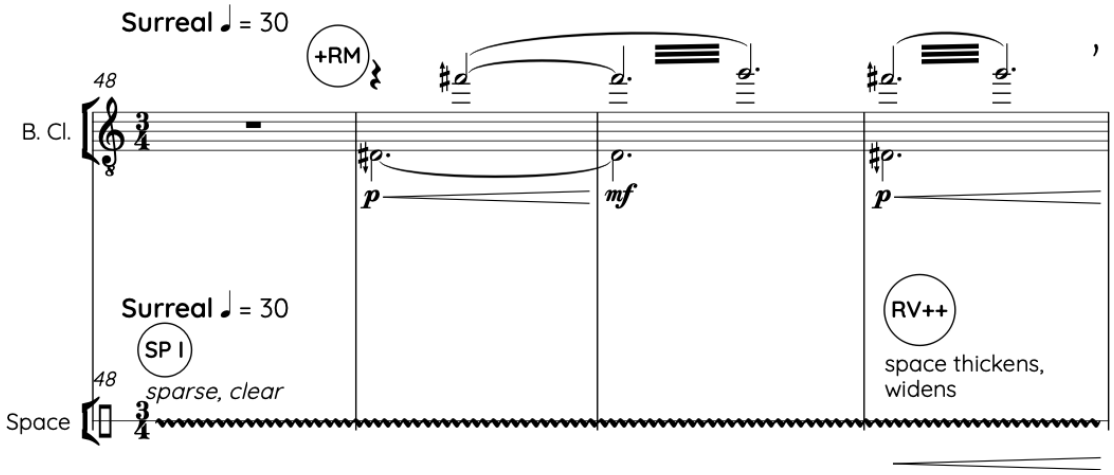
At a tangible level, these spaces are conjured with mostly time-based effects, such as delay and reverb. Other effects include ring modulation (seen in listing 1's SynthDef) and pitch-shifting (see appendix C, pp. 90–91). The code for each space sets initial arguments for the effects parameters, which the first laptop performer then adjusts throughout the piece.

The effects' intention is to evoke unique spaces that surround the bass clarinet. Subtle delay and reverb render a depth and width to the space; jarring echoes, when used with harsh timbres, bring the attention closer and scatter any sense of centrality. Denser spaces and ring modulation,

[76] *Oxford English Dictionary Online*, s.v. 'morph, (*v.*)', published December 2022, https://www.oed.com/view/Entry/246240?rskey=Yei1A6&result=6&isAdvanced=false.

when fused with textural extended techniques, radically transform the bass clarinet's natural sonic identity.

Figure 5. Multiphonics with spatialization and effects, bars 48–51.



3.3.3 OSCdefs

An OSCdef is code that references the functionality of an OSCFunc (its superclass), which registers one or more Functions in response to an OSC message.[77] In the code for the two laptop parts, OSCdefs define Functions that contain conditional logic and Routines to be carried out when a certain OSC message is received.

For example, when the performer selects a space in the first laptop's TouchOSC user interface (UI), SuperCollider sets the parameters for some or all of the effects to new values, depending on what is specified

[77] *SuperCollider 3.12.2 Documentation*, s.v. 'OSCdef', https://doc.sccode.org/Classes/OSCdef.html.

in the OSCdef's Function. The alterations to the code are made and the result is published back to the device's screen.

This happens as follows:

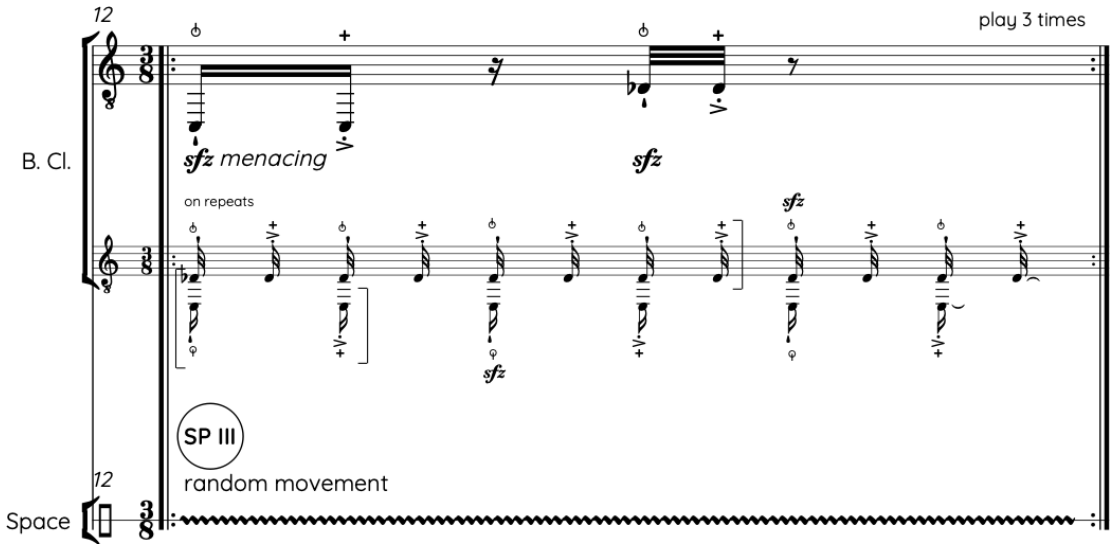
1. The laptop performer adjusts a value in the device UI, which sends a message to the SuperCollider server via OSC.
2. SuperCollider receives this OSC message, which usually contains the address and any relevant value information.
3. The code is then altered based on this information.
4. A customized message may be printed to the post window in the IDE to confirm any changes.
5. SuperCollider sends an OSC message that contains the relevant data back to the device.
6. The device UI is updated with these new values.

Despite appearing convoluted, this all happens practically instantaneously, allowing for the seamless integration between the laptop performers and the bass clarinettist. This is because SuperCollider's architecture also uses OSC to communicate between the language and the server (as the two components are separate). Therefore, with the appropriate code set up, using OSC to access the server with an external device is virtually as direct as changing the code by hand.

Figure 6 shows the first implementation of space III in the score and its OSCdef. This section has the bass clarinet playing in its lower register, using slap tongue and key clicks to create a percussive texture

on which a synth applies a short delay (about a quarter of a second) to chaotically echo the sounds.

Figure 6. Bass clarinet with echoes, bar 12.



All of the OSCdefs in both laptop parts use the variable `touchOSC`, which is a network address (`NetAddr`) that is accessible throughout the code file. The OSCdefs need this address to identify the OSC message sender (the control device) by its IP address and port number. The code that defines `touchOSC` reads as follows:

```
~touchOSC = NetAddr.new("192.xxx.x.xxx", 9000);
```

In listing 2, on line 3, the `new()` method is called on an `OSCdef` UGen to create a new instance of the class on the server. As with SynthDefs, an OSCdef's first parameter is a descriptive key – in this case, `sp3` (space III). The second parameter is an unnamed Function that contains the algorithm that will run when something happens at a certain OSC

address (the third parameter). This particular OSCdef lists only one parameter, `msg`, which is found on line 4.

```
1  // MORPHEI SOMNIUM: LAPTOP 1
2
3  OSCdef.new(\sp3, {
4      | msg |
5
6      if ( (~micIn.isPlaying), {
7          // msg[1] == 1 means that the button is ON
8          if ( (msg[1] == 1), {
9              // refresh UI
10             ~touchOSC.sendMsg('/spaces/0', 0);
11             ~touchOSC.sendMsg('/spaces/1', 0);
12             ~touchOSC.sendMsg('/spaces/2', 0);
13             ~touchOSC.sendMsg('/spaces/4', 0);
14
15             if ( (~micIn.isPlaying), {
16                 ~micIn.set(
17                     \xpos, 0,
18                     \ypos, 0
19                 );
20             });
21
22             ~delaySynth.set(
23                 \delayTime, 0.25,
24                 \decay, 3,
25                 \mix, 0.65
26             );
27
28             // update UI to reflect new settings
29             ~touchOSC.sendMsg('delayTime', 0.25);
30             // adjust scale to fit within the button's range
31             ~touchOSC.sendMsg('decay', 3.linlin(0, 10, 0, 1));
32             ~touchOSC.sendMsg('delayMix', 0.7);
33             ~touchOSC.sendMsg('/spaces/3', 1);
34
35             // print selection to post window
36             "selected: space iii".postln();
37         }, {
38             // if the button is OFF, set default space
39         });
40     }, {
41         // if there is no active micIn Synth
```

```
42          // reset UI and print error message
43          "no input: turn on mic".postln();
44      });
45  },
46  // OSC path (where to send/receive messages)
47  "/spaces/3"
48  );
```

Listing 2. Space III OSCdef (some code has been left out for simplicity's sake).

The code on line 6 states the primary condition that must be met to allow for most of the code in the Function to be run if the performer selects space III. This conditional statement dictates that in order to process the microphone's signal, there must be an active `micIn` synth on the server. To be explicit, this means that: (1) there is a microphone that is switched on and plugged into an audio interface with its level appropriately adjusted, (2) the SuperCollider server is booted and reading from the correct input of this interface, (3) the `micIn` SynthDef with the code specified in listing 1 has been added to the server using the `add()` method, and (4) the following code has been executed:[78]

```
Synth.new(\micIn).play();
```

If all of the above has successfully occurred, then the code within the first of two nested conditionals (starting on line 8) will run.

[78]Method parentheses are not strictly required if there are no arguments being passed in.

This condition is quite plain: the “III” button must be selected in the control device’s UI. This means that the device must be sending the value 1 from that address, and not 0 (the only possible values for the space buttons in the TouchOSC UI). If the correct space is indeed selected, then the code from lines 10 to 36 will run.

To start, the OSCdef deselects the other space buttons in the UI on lines 10–13 (i.e., the objects at those addresses are sent the 0 message). This is purely to provide visual feedback and has no bearing on the code that concerns audio processing just yet. Each space button in the TouchOSC UI has an address corresponding to its space number; i.e., “/spaces/0” is the “0” button, “/spaces/1” is the space “I” button, and so on. (Note that the OSCdef does not send space III’s button a 0, as this is the button that has been selected.)

Whilst the code on lines 15–20 seems redundant (the innermost conditional), it serves as a safety mechanism to prevent unwanted external code from working against other code in the Function. These lines set the pan position of an active `micIn` synth to (0, 0), if one exists on the server at that point.

The OSCdef's main audio functionality is on lines 22–26, which is to set new arguments for the delay synth that is running on the server. This is done in order to process the bass clarinet, which is connected via the `micIn` synth. The three adjustable settings on `delaySynth` are: (1) the delay time (the interval between feedback responses), (2) the decay time (i.e., how long it takes for the delayed signal to decrease by 60 dB), and (3) the delay mix, which is the blend between the dry and processed signal. The delay effect is notated literally in the score (see figure 6).

An argument of 0.25 seconds is set as the new value for `delaySynth`'s `delayTime` parameter. At 30 BPM (beats per minute), the piece's current tempo, this time interval is equivalent to about a quaver. The OSCdef sets a decay time of three seconds, and the mix value of 0.65 on line 25 means that the delayed (wet) signal constitutes 65% of the sound and the dry signal only 35%. The result is a dense and tumultuous atmosphere, seeing that the echoes are panned randomly across four channels using the `SplayAz` UGen.

Lines 29–33 show similar code to lines 10–13 – their aim is to update the UI with the new settings. The OSCdef resets the three delay parameters first and then sends a 1 to the space III button to show that

the performer has selected it. The string “selected: space iii” on line 36 is printed to the post window to affirm the selection.

Note that on line 31, the decay value of three seconds is linearly scaled so that it fits within the range of zero to one, which is the range in which the TouchOSC button operates. Therefore, as the decay range is zero to ten seconds, three seconds would be equivalent to 0.3 units (which TouchOSC accepts).

To save space and prevent a lengthier code example, on lines 37–39, the functional code has been replaced with a comment explaining what would happen if the button is not selected (i.e., the default space is selected). In the outermost conditional (checking if there is an active synth), some code has again been left out; however, its functionality is explained (as a comment on line 38) should there be no active `micIn` synth. The penultimate line of code is the space III button’s OSC path (address), from which messages are received and to which information is sent regarding changes made automatically in the code when the performer selects a new space.

3.4 *Morphei Somnium* Analysis: Part II

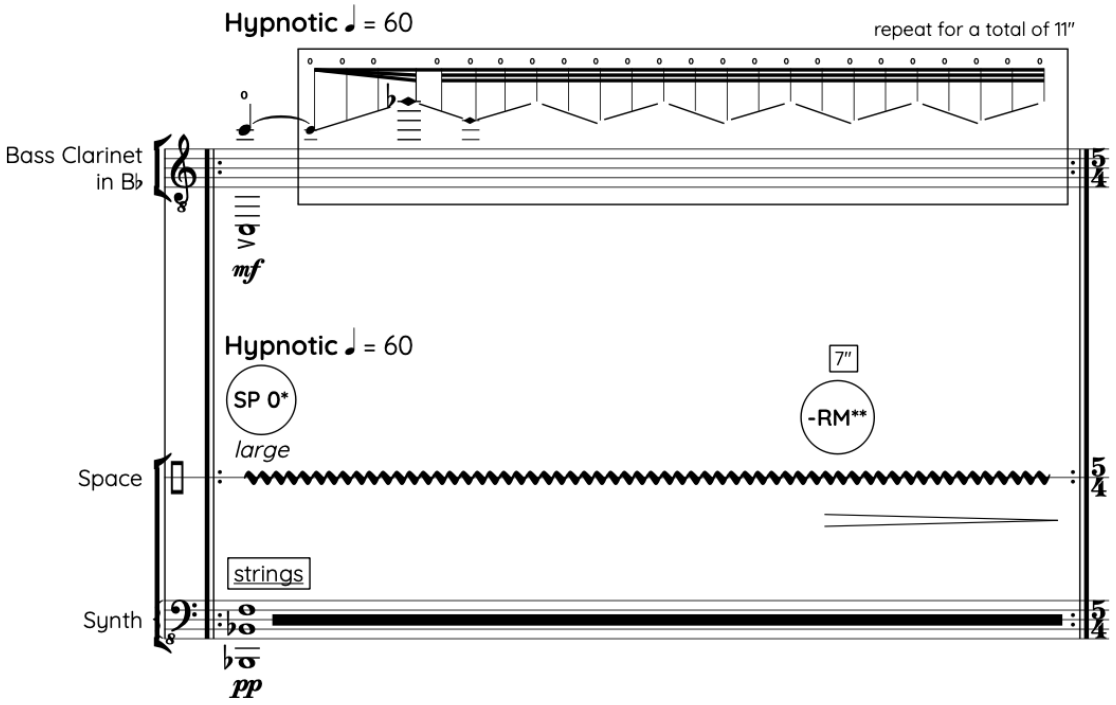
3.4.1 Synths: Strings

In addition to the bass clarinet as a sound source, *Morphei Somnium* uses electronic and pre-recorded sounds that are generated and triggered in SuperCollider. These take the form of "synths" and "tape", as specified in the second laptop part. The former's main purpose is to create an atmospheric background of sustained textures – namely, "drones" and "ambients". These supporting resonances utilize oscillators and noise generators with filtering and frequency modulation (FM) for the most part, providing a range of colours that at times contrasts the action in the other parts.

The first synth, "strings", recreates an orchestral double bass section. The synth "sits" underneath the bass clarinet, which is oscillating between harmonics four (a fifteenth above the fundamental) and seven (a fifteenth plus a minor seventh above) whilst sustaining a written low C (sounding a major ninth lower). The strings sound the lower three harmonics of this resonance with a $B\flat_0$ (ca. 29.14 Hz)[79] fundamental.

[79]Where A_4 = 440 Hz.

Figure 7. Strings synth with bass clarinet harmonic glissandi, bar 1.



To generate this texture, the `varsaw` SynthDef defines four variable sawtooth (triangle) wave oscillators (seen in listing 3). These four oscillators are all slightly out of tune with each other, as their arbitrary starting frequency (here, 30 Hz – roughly B_0) is multiplied by random values between 0.98 and 1.02 (29.4–30.6 Hz). Code within the `do()` loop is executed ten times, meaning that on each iteration, four more oscillators are added to the synth, each one expanded to four channels, with randomized phase offsets.

Note lines 10–14, where the `kr()` method is called on the `Env` UGen to generate an envelope. This envelope is applied to the synth on line 32 and has an attack time of one second, a sustain of zero seconds (i.e., the sound

```
// MORPHEI SOMNIUM: LAPTOP 2

SynthDef(\varsaw, {
    | freq = 30, atk = 1, sus = 0, rel = 3, amp = 1, out = 0 |

    var sig, temp;
    var sum = 0;

    // define an envelope
    var env = Env(
        [0, 1, 1, 0],
        [atk, sus, rel],
        [1, 0, -1]
        ).kr(2);

    // repeat the following code ten times, adding recursively
    10.do({
        // create four variable sawtooth wave oscillators
        temp = VarSaw.ar([
            freq * { Rand(0.98, 1.02) },
            freq * { Rand(0.98, 1.02) },
            freq * { Rand(0.98, 1.02) },
            freq * { Rand(0.98, 1.02) }
        ] !4,
        Rand(0.0, 1.0) !4
        ) * 0.2;

        sum = sum + temp;
        });

    // apply envelope and adjust output level
    sig = sum * env * (amp * 0.5);
    // pan randomly across four channels
    sig = SplayAz.ar(4, sum);

    Out.ar(out, sig);
}).add();
```

Listing 3. Varsaw SynthDef.

reaches its peak amplitude immediately after the attack is complete) and a release time of three seconds; although, these parameters can be

changed when a new synth is instantiated. Once again, the `SplayAz` UGen pans the synth arbitrarily across four channels.

Listing 4 shows a *Pbindef* – code that defines a reference to a *Pbind* (an object in the `Pattern` class), which is an Event that the server executes with specified arguments. Much like SynthDefs and OSCdefs have a name and Function, Pbindefs are given an identifier and an `instrument` key, whose matching value is an instance of some SynthDef. In this case, the Pbindef's identifier is `strings` and its `instrument` is an instance of the above `varsaw` SynthDef.

```
// MORPHEI SOMNIUM: LAPTOP 2

Pbindef(\strings,
    \instrument, \varsaw,
    \dur, Pexprand(0.5, 1, inf),
    \freq, 22.midicps(),
    \harmonic, Pexprand(1, 3, inf),
    \atk, Pexprand(0.25, 0.5, inf),
    \sus, Pexprand(0.6, 0.8, inf),
    \rel, Pexprand(1, 3, inf),
    \amp, 1,
    \delay_out, ~bus[\empty],
    \out, ~bus[\reverb]
);
```

Listing 4. Strings Pbindef.

This Pbindef has a duration of some random value between 0.5 and 1 beats – calculated with an exponential distribution (i.e., favouring lower

floating-point numbers)[80] – and loops infinitely using the `Pexprand` UGen. The code on line 6 shows the fundamental frequency – MIDI note 22 ($B\flat_0$ from before) – which is converted into cycles per second (i.e., Hz) with the `cps()` method. On line 7, another `Pexprand` UGen chooses a random harmonic from the first three – the fundamental itself, and the octave and twelfth above.

As suspected, the `strings` synth's attack, sustain and release parameters have been customized, once again utilizing randomization UGens to generate new values on each cycle. The amplitude is left at one and the synth is not sent to the delay bus but is sent to the reverb bus.

The remaining electronically synthesized instruments in the synth part – excluding "ambient 3" below – include two ambient textures (one using a sine wave and the other using a sawtooth wave oscillator), two drones (both using sine wave oscillators) and a "horns" synth, which uses another sawtooth wave oscillator. These synths all use similar frequency modulation (seen above), amplitude modulation (see app. C, p. 93) and filtering techniques (see below) in their audio synthesis.

[80] SuperCollider's TempoClock counts one beat as one second by default (i.e., 60 BPM); thus, this Pbindef's duration varies between one and two seconds.

At this stage, it is obvious that I favour randomization over fixed values when programming synths and the like in SuperCollider, which speaks to my adoption of experimental and generative music processes mentioned in the previous chapter.

3.4.2 Synths: Ambient 3

SuperCollider makes it possible to synthesize audio from other sources as well, which can be seen in the `noise` SynthDef in listing 5. Here, the primary source is pink noise.

After fairly standard arguments on lines 4–5 (with the exception of `rq`, the quality factor), the SynthDef generates an envelope that is stored in the `env` identifier. This envelope uses the `atk`, `sus` and `rel` arguments as times for the envelope generator.

The SynthDef defines a `PinkNoise` UGen on line 15 with a multiplier value of one, which has been expanded to create eight noise generators. A band-pass filter (`BPF` UGen) with a base centre frequency of 500 Hz (per the first initialized input argument on line 4) filters these generators on lines 18–23. In a type of nested modulation, a sine wave oscillator (whose own frequency is some random value between 1 and 5 Hz) modulates

```
// MORPHEI SOMNIUM: LAPTOP 2

SynthDef(\noise, {
    | freq = 500, atk = 1, sus = 0, rel = 3, rq = 0.01,
    amp = 1, out = 0 |

    // define an envelope
    var env = Env(
        [0, 1, 1, 0],
        [atk, sus, rel],
        [-1, 0, 1]
        ).kr(2);

    // create eight pink noise generators
    var sig = PinkNoise.ar(1 !8);

    // apply a band-pass filter to the noise
    sig = BPF.ar(
        sig,
        freq + SinOsc.ar(SinOsc.ar(0.5) * exprand(1, 5)),
        rq,
        1/rq.sqrt()
        );

    // apply envelope and amplitude
    sig = sig * env * amp;
    // pan randomly across four channels
    sig = SplayAz.ar(4, sig);

    Out.ar(out, sig);
}).add();
```

Listing 5. Noise SynthDef.

the filter's centre frequency and is itself modulated by another `SinOsc` at 0.5 Hz.

The filter uses the square root of the Q (quality) factor to adjust the level (the argument following `rq`).[81] Per custom, the SynthDef applies the envelope, and the synth is panned randomly and outputted to the main bus.

The Pbindef "ambient 3" uses an instance of the `noise` SynthDef as its instrument and can be seen in listing 6. Once again, a `Pexprand` UGen randomizes the Pbindef's duration by selecting a random value between 0.5 and 2, calculated with an exponential distribution on every new cycle (looping endlessly).

```
1  // MORPHEI SOMNIUM: LAPTOP 2
2
3  Pbindef(\ambient3,
4      \instrument, \noise,
5      \dur, Pexprand(0.5, 2, inf),
6      \freq, Pexprand(100, 200, inf)
7      * Pexprand([1, 0.85], [30, 20], inf),
8      \atk, Pexprand(2, 4, inf),
9      \rel, Pwhite(3.0, 5.0, inf),
10     \rq, Pexprand(0.001, 0.005, inf),
11     \amp, Pexprand(0.8, 1, inf),
12     \out, ~bus[\reverb]
13 );
```

Listing 6. Ambient 3 Pbindef.

The base centre frequency argument on lines 6 and 7 implements two more of these UGens, this time with one modulating the other. The

[81] SuperCollider actually uses the reciprocal of the Q factor ("rq"), so *1 / rq.sqrt()* is simply the square root of the Q factor.

first `Pexprand` randomly selects values between 100 and 200 (Hz), and the second one multiplies each value by an array of two possible values. This creates two `noise` synths, where one has a base centre frequency of between 100 and 300 Hz and the other has a frequency of between 170 and 400 Hz.

The Pbindef also uses random attack and release times, with the `rel` parameter featuring a `Pwhite` UGen, which generates random numbers with a uniform distribution (instead of exponential).

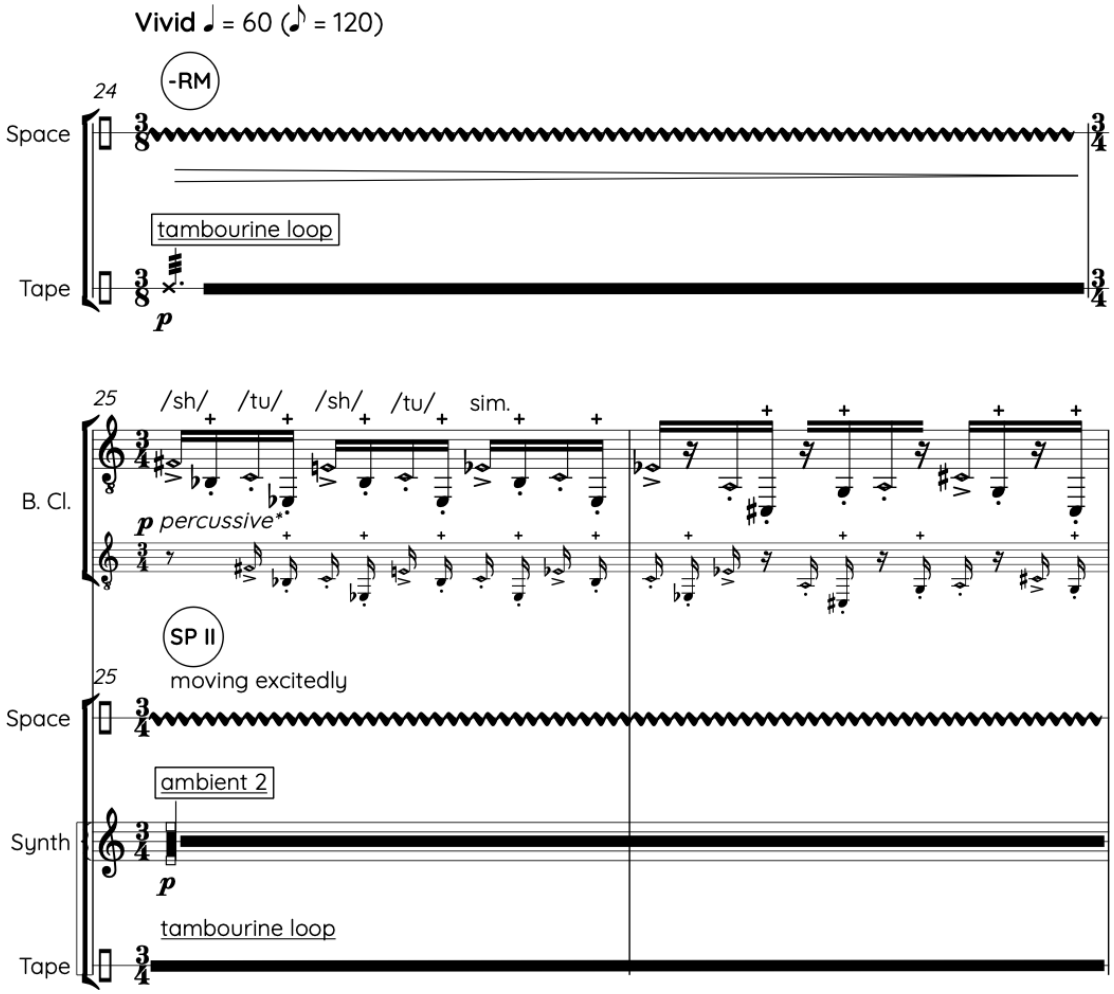
On line 10, the Q factor has a random value between 0.001 and 0.005, which is extremely narrow, and yields the intended wispy timbre for this synth. Lastly, the synth's amplitude is randomized and the synth is sent out to the reverb bus.

3.4.3 Tape: Tambourines

Morphei Somnium's tape part consists of a number of samples of tambourines being played in distinct ways (e.g., hit, shook and looped) and timpano (pl. timpani) shell strikes. The tape part's primary task is to provide novel gestures that contrast the textures of the spatialized bass clarinet and synths. Some of the techniques that are exercised in the samples' manipulation include time-stretching, transposition and layer-

ing, as well as filtering, reverb and delay. These gestures typically occur at landscape transfiguration as links to usher in a new space; however, there is also a "tambourine loop" Pattern that runs from bars 24 to 35. (see app. C, p. 95). Effects on this tambourine jangle transform it from a series of gestures into a dense texture that supports a rhythmic figure in the bass clarinet.

Figure 8. Tambourine loop; ambient 2 synth and bass clarinet rhythmic figure, bars 24–26.



The second laptop's TouchOSC interface – designed in a similar way to the first laptop's interface – is used to trigger processed samples in

the piece (see app. D, pp. 103–4). Laptop 2's UI is split into three main sections: the top row contains the latch toggle buttons to trigger the preprogrammed Patterns, the second row has a series of momentary toggle buttons to trigger one-shots, and the bottom row has rotary knobs for effects including delay, pitch-shifting and reverb (which the performer adjusts *ad lib.*). There is also an emergency stop button to cease all running Patterns and Routines, should the need arise.

The tape part's first contribution to the piece comes in bar 7 (see fig. 4) and is the clang of a tambourine being struck. This gesture acts against a brief figure in the bass clarinet's low register, which a synth processes with reverb and delay.

Listing 7 defines a Routine called `tamboHit`, which is a one-shot that consists of twelve synths, each instances of a SynthDef called `playBuf`. This SynthDef is used to trigger and process all the pre-recorded samples in *Morphei Somnium* (see app. C, p. 94).

A dictionary that references all the audio files and their locations in storage was created at the beginning of the file. This dictionary reads each audio file into its own *Buffer* object, which is a representation of a buffer containing the file on the server.

```
1  // MORPHEI SOMNIUM LAPTOP 1
2
3  ~tamboHit = Routine({
4      12.do({
5          Synth.new(\playBuf, [
6              \buf, b[\tambourines][0],
7              \atk, 0.05,
8              \sus, 1,
9              \rel, exprand(3, 5),
10             \rate, { exprand(0.89, 1.12) },
11             \startPos, 0,
12             \bpfmix, 0.8,
13             \rq, 0.1,
14             \xpos, 0,
15             \ypos, 0,
16             \amp, 1/4,
17             \pitch_shift_out, ~bus[\pitchShift],
18             \delay_out, ~bus[\delay],
19             \out, ~bus[\reverb]
20         ],
21         ~mainGrp,
22         );
23     });
24 });
```

Listing 7. Tambourine hit Routine.

`b[\tambourines][0]` points to the required timbre, which is the first audio file in the "tambourines" subdirectory. The synth in `tamboHit`'s Routine provides arguments for most of the parameters defined in the `playBuf` SynthDef (see lines 6–19).

The buffer number (`buf`) comes first and corresponds to the raw audio file on the server. Next, the synth gives the envelope arguments: an attack time of 0.05 seconds, sustain time of 1 second and a release

time of some random value between three and five seconds (once again, chosen with an exponential distribution). The `rate` parameter on line 10 is the buffer's playback rate – i.e., its speed – where a value of one is the original. This parameter is randomized on each of the `do()` loop's iterations, thereby setting independent playback rates for all twelve synths. These values range from 89% to 112% of the original file speed. The product is a sound cloud of tambourine hits, each individual timbre with a varied pitch (due to the time-stretching) and timbral profile.

The addition of more pitch-shifting (controlled in real time by the second laptop performer) transforms this complex texture further, and to add yet more to the obscurity, a `BPF` UGen applies a band-pass filter with a Q factor of 0.1 to the overall resonance. `tamboHit`'s synth does not have a customized centre frequency argument; thus, it takes `playBuf`'s default value of 400 Hz. However, the dry/wet mix (`bpfMix`) favours the filtered signal, with it making up 80% of the combined sound (see line 12).

On lines 14 and 15, the synth's spatial position arguments make no alterations, and on line 16, each synth's amplitude is scaled down to compensate for the excessive total level of twelve synths sounding

simultaneously. Thereafter, the synths are sent to all three effects buses and added to the main group (which contains all the instrument synths).

A brief look at an OSCdef from the second laptop part shows something similar to those found in the first laptop part.

```
1  // MORPHEI SOMNIUM: LAPTOP 2
2
3  OSCdef.new(\tamboHit, {
4      | msg, time, addr, recvPort |
5
6      // if the cue button is selected
7      if ( (msg[1] == 1), {
8          // play Routine
9          ~tamboHit.play();
10
11         // print one-shot name to post window
12         "tambo hit".postln();
13
14         // set pitch-shift parameter value
15         ~pitchShiftSynth.set(\mix, 0.7);
16
17         // update device UI
18         ~touchOSC.sendMsg('/pitchShiftMix', 0.7);
19     });
20     // reset Routine
21     ~tamboHit.reset();
22 },
23 // OSC path (where to send/receive messages)
24 "/tape/tamboHit"
25 );
```

Listing 8. Tambourine hit OSCdef.

In listing 8, a new OSCdef called `tamboHit` (responsible for the similarly named Routine's execution) is created on line 3. Following the same parameters as before, line 7 checks that the performer has selected

the “tambo hit” button in the UI. If this is the case, then the server executes the Routine and prints a message with the one-shot’s name to the post window.

On line 15, the OSCdef sets a new argument for the pitch-shifting synth (which receives input from all synths that send to its bus). The OSCdef then updates the value on the respective rotary knob in the UI. Thereafter, on line 21, the Routine is reset for future use, and on line 24 is the address to or from which OSC messages are sent or received.

3.4.4 Tape: Timpano Shell Strikes

Besides the numerous tambourine samples, the chime of a timpano being struck with a mallet provides an additional contrasting textural layer. Five individual samples were cut out of a high-quality fifty-second royalty-free audio clip and serve as the raw material for the `timpShell` Pbindef. As with all the timbres in the Laptop 2 part, the code processes the original timpano tones in a variety of ways.

To start off, listing 9 shows the Pbindef in question, with an identifier and instrument (an instance of the `playBuf` SynthDef). On each new cycle, the source buffer is chosen at random from the five timpano strikes in the “timpani” subdirectory (see line 5). This is done by calling the

```
// MORPHEI SOMNIUM: LAPTOP 2

Pbindef(\timpShell,
    \instrument, \playBuf,
    \buf, b[\timpani][(0..4).choose()],
    \dur, Pexprand(2, 5, inf),
    \atk, Pexprand(1, 3, inf),
    \sus, 1,
    \rel, Pexprand(6, 8, inf),
    \rate, Pwhite(0.8, 1.2, inf),
    \startPos, 0,
    \freq, 30 * Pexprand(1, 20, inf),
    \rq, Pexprand(0.05, 0.2, inf),
    \bpfmix, 0.9,
    \xpos, Pwhite(-1, 1, inf),
    \ypos, Pwhite(-1, 1, inf),
    \amp, 5,
    \delay_out, ~bus[\delay],
    \out, ~bus[\reverb]
);
```

Listing 9. Timpano shell Pbindef.

`choose()` method on an array containing all integers from zero to four, which corresponds to the audio files' indices in memory, where 0 is the first file and 4 is the fifth file.

A `Pexprand` UGen randomizes the Pbindef's duration to between two and five seconds, and the attack time is between one and three seconds. `timpShell` takes one second to reach its peak intensity (the sustain value) and its release time is between six and eight seconds. A `Pwhite` UGen randomizes the buffer's playback rate (with a linear distribution)

to be 80%–120% of its original speed. On line 11, the synth plays back the audio file from the beginning in each instance.

A band-pass filter (which changes on every cycle) with a centre frequency of 30 Hz multiplied by a random value between one and twenty is applied to the buffer. This filter's Q (quality) factor is randomized to between 0.05 (very narrow) and 0.2 (reasonably narrow), and the mix between dry and wet signal is 90% in the wet signal's favour.

Importantly, the timpano shell strikes are randomly panned across both axes; thus, on each cycle, a strike will ring at an unpredicted location in physical space both from left to right and from front to back (relative to the listener). This magnifies *Morphei Somnium*'s multidimensionality and occurs at a section in the piece where the landscape is markedly hazy. The amplitude is increased – as the tones are quite quiet – and to conclude, the Pbindef outputs the buffer to the delay and reverb buses.

On the whole, SuperCollider served as a powerful tool in composing *Morphei Somnium*, and I only leveraged a few of its countless UGens. In this piece alone, the environment allowed for the integration of algorithmic composition, randomness, spatialization, interactive performance and electroacoustic music methodologies with merely a few months'

training in the language. The end result is a piece of mixed music that was recorded in 2021 and aired at BE@UCT 2022.

3.5 Digital (Visual) Art

3.5.1 Algorithmic and Computer Art

Creative programming in digital art straddles both of Christiane Paul's categorizations (mentioned earlier). Technically a predecessor to digital art, an early implementation of Paul's first category is *algorithmic art*. Algorithmic art (also known as rule-based art) existed prominently during the 1960s with the Fluxus group and conceptual art, and eventually migrated to the digital realm, with computer artists such as Charles Csuri, Vera Molnár and Frieder Nake at the fore.[82]

British computer artist Ernest Edmonds defines (modern) algorithmic art as "[art that] is produced with the aid of a computer by programming it to follow some procedure that generates the art object."[83]

Self-proclaimed "Algorist"[84] Roman Verostko is a digital artist who has spent many years developing what he calls *art form generators* (i.e.,

[82]Paul, 'Introduction: From Digital to Post-digital – Evolutions of an Art Form', 5.

[83]Ernest Edmonds, 'Algorithmic Art Machines', *Arts* 7, no. 1 (2018): 3, https://doi.org/10.3390/arts7010003.

[84]See https://www.algorists.org/algorist.html.

software) that drive pen plotters connected via computer. These devices are able to draw lines with ink on paper, thereby translating the artist's algorithmic intentions into works of visual art. Verostko finds wonder in the ability for an artist to employ such instructions to develop "[art] form-growing" concepts in their creative process; he admires the material generative opportunity that computer programming brings when coupled with mechanics.[85]

One of computer art's most established and respected works is Vera Molnár's *Dialog Between Emotion and Method* from 1986, which is a fine example of geometric abstraction that uses repetition and randomness. A number of modern-day creative programmers (myself included) have referenced this work whilst discovering and experimenting with generative and computer art. Figure 9 is an export of four of my own randomized iterations of a sketch that is inspired by Molnár's work (recreated in the Processing creative coding environment).[86]

Traditionally speaking, with algorithmic art, the digital aspect only exists with respect to the process and technology implemented – the art object is typically concrete; for example, an image produced using

[85]Roman Verostko, 'Algorithmic Fine Art: Composing a Visual Arts Score', in *Explorations in Art and Technology*, 2nd ed., ed. Linda Candy, Ernest Edmonds and Fabrizio Poltronieri (London: Springer, 2018), 78, https://doi.org/10.1007/978-1-4471-7367-0_7.

[86]Sadly, the code for this sketch has been lost.

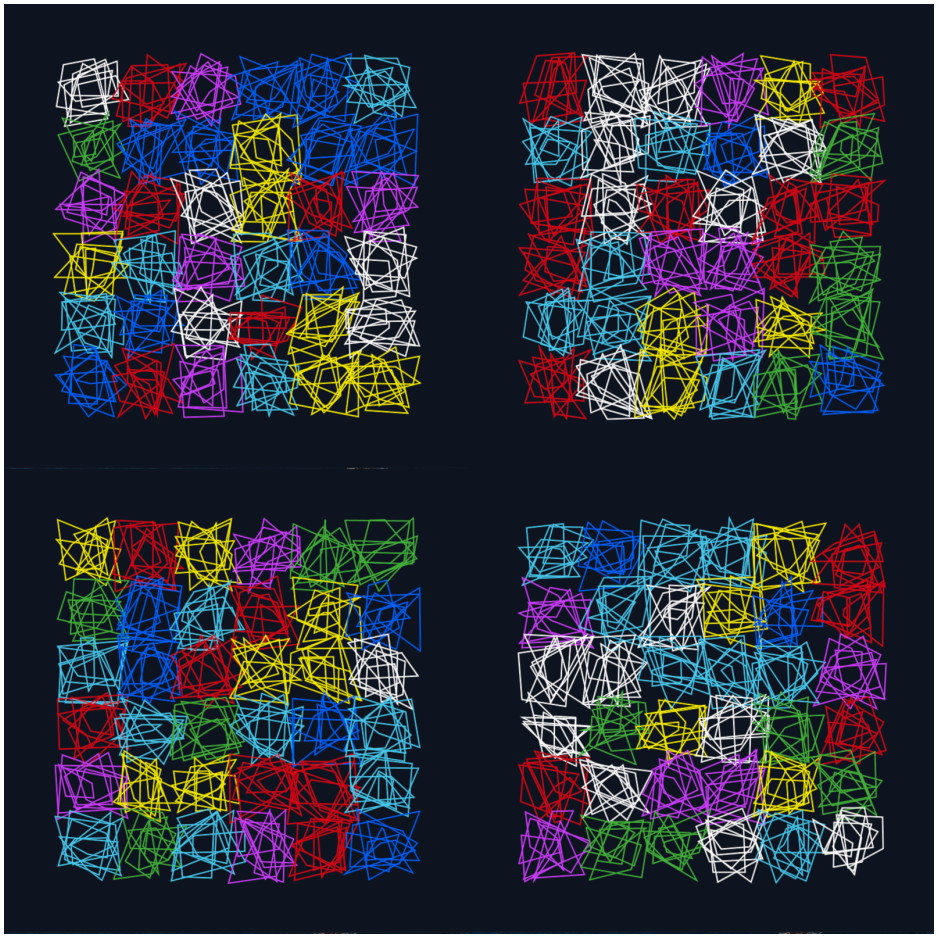


Figure 9. *Vera Molnár 001, 003–005.* Tribute to a computer art pioneer.

a pen plotter or inkjet printer. Furthermore, the emphasis on the procedural element in algorithmic art draws the attention away from the resultant artwork. Whilst it gives algorithmic art its generative quality, in many ways this is contrary to digital art concepts, where the presence of digital technology as a medium is supposedly the focus. All the same, as an art form that integrates computer programming into the creative process, algorithmic art fits the common description of digital art, but not always digital-born art.

3.5.2 Demoscene

Another application of creative coding in digital (multimedia) art is found in the *demoscene*, an art movement that peaked in the 1990s. Scholar Doreen Hartmann classifies this movement as:

> . . . a vivid digital subculture of approximately up to 10,000 people [who] work internationally and collaboratively on different kinds of non-commercial digital artefacts, . . . essentially so called (computer) demos, which are audio-visual real-time animations.

Meanwhile, *demosceners* are people who generate digital images, animations and sounds with code, combining creative processes with complex computer programming methods in an individual way.[87]

This more modern case of creative coding in digital art tends more towards Paul's digital-born field, as the art object is software based – that is, computer-generated, where the result is a digital file residing on a computer or similar device, and not, for example, a live-action film captured with a (digital) camera. In the latter case, the technology used may be digital, however, the art object is aesthetically analog.

What is perhaps different in the demoscene is that, according to Hartmann, demos (demonstrations) are intentionally non-interactive,

[87]Doreen Hartmann, 'Computer Demos and the Demoscene: Artistic Subcultural Innovation in Real-Time', in *Proceedings of the 16th International Symposium of Electronic Art* [ISEA2010 Ruhr], ed. Judith Funke et al. (Berlin: Revolver, 2010), 124, https://www.isea-archives.org/docs/2010/proceedings/ISEA2010_proceedings.pdf.

which apparently serves to preserve the synchronization between the visual and sonic elements.[88] This is an alternative practice to some forms of digital art, where the work's identity lies in the fact that it is interactive. Nonetheless, in my opinion, the demoscene presents a solid argument for the integration of creative programming in digital-born art.

3.5.3 Contemporary Creative Programming

Since the early 2000s, several creative coding languages and environments have been developed and made available to any interested party. Many of these environments are free and open source and are actively maintained by programmers in the community to ensure that the code stays secure and robust.

Some popular creative coding environments used for generating visuals include:

- openFrameworks: creative coding toolkit built on top of WebGL
- Cinder: C++ library for graphics, audio and video (a common alternative to openFrameworks)
- Processing: popular programming environment built for use in computer art, new media art and visual design
- p5.js: JavaScript implementation of Processing that runs in the web browser, aimed at education and experimentation

[88] Hartmann, 'Computer Demos and the Demoscene: Artistic Subcultural Innovation in Real-Time', 124.

- Max/MSP/Jitter: visual programming language for making music and multimedia art (proprietary)
- Pure Data (Pd): an open-source alternative to Max, developed by Miller Puckette

These and more environments are used for everything from video games to digital art installations and interactive performances, as well as in the commercial industry (e.g., advertising and product prototyping).

3.5.4 Processing

Processing is a popular creative coding environment and is many creative programmers' introduction to the medium. The language uses Java – the quintessential compiled object-oriented programming language – with a few modifications and additions geared at tailoring Processing's functionality. Some of the features of the environment include its ability to receive input, work with images and typography, generate and transform 2-D and 3-D shapes and render animations, which makes it a complete toolkit for creating generative and computer art.

Listing 10 shows some sample code that uses basic 2-D shapes to create a geometric artwork using Processing.

Every sketch in Processing requires two functions: `setup()` and `draw()`. The first function is used to create a canvas on which the sketch

```
1  void setup() {
2    size(1080, 1080);
3  }
4
5  void draw() {
6    background(0);
7    stroke(#f1f1f1);
8    strokeWeight(15);
9
10   square(108, 108, 864);
11   fill(0);
12
13   triangle(540, 216, 216, 864, 864, 864);
14
15   circle(540, 360, 80);
16   circle(320, 800, 80);
17   circle(760, 800, 80);
18
19   save("eskom.png");
20 }
```

Listing 10. Basic Processing sketch using 2-D shapes.

is drawn, and the second contains the code for the sketch itself. On line 2 (in the `setup()` function), the size of the canvas is determined (1080x1080 pixels)[89] and on line 6 (in the `draw()` function), the background colour is set to black. The `stroke` and `strokeWeight` fields refer to the colour (white) and thickness (15 pixels) of the lines that draw the shape, and on lines 10–17 is code for drawing the shapes themselves. Lastly, the `save()` function exports and saves the sketch as a PNG in the same directory as the sketch. The result is seen in figure 10 below.

[89] Figures 10–12 have all been scaled down.

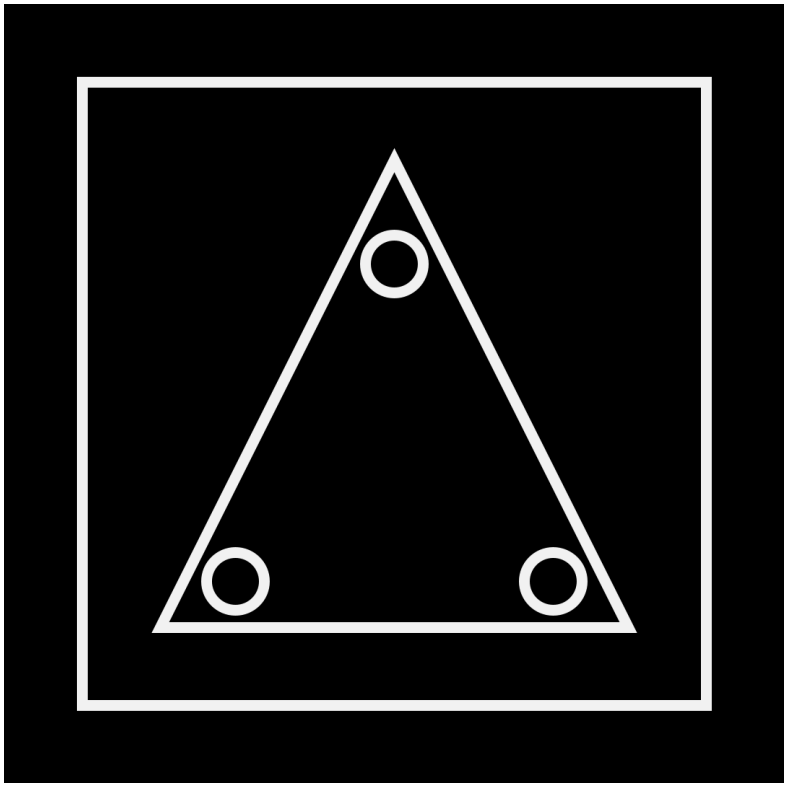


Figure 10. *Eskom* (2021). Digital art (computer-generated visual media).

An example using colour, noise (random value generators), iteration (repetition) and an external controller (a key being pressed on the computer keyboard) can be seen in figure 11. (See app. C, pp. 96–97 for the code)

As a final demonstration, figure 12 is a rendering of a series of six still frames from an animation of coloured boxes which appear to be rotating in three dimensions. This is, in fact, an illusion, as the sketch does not use Processing's 3-D renderer. The sketch uses iteration and translation to create the false sense of movement, which is then captured as a series

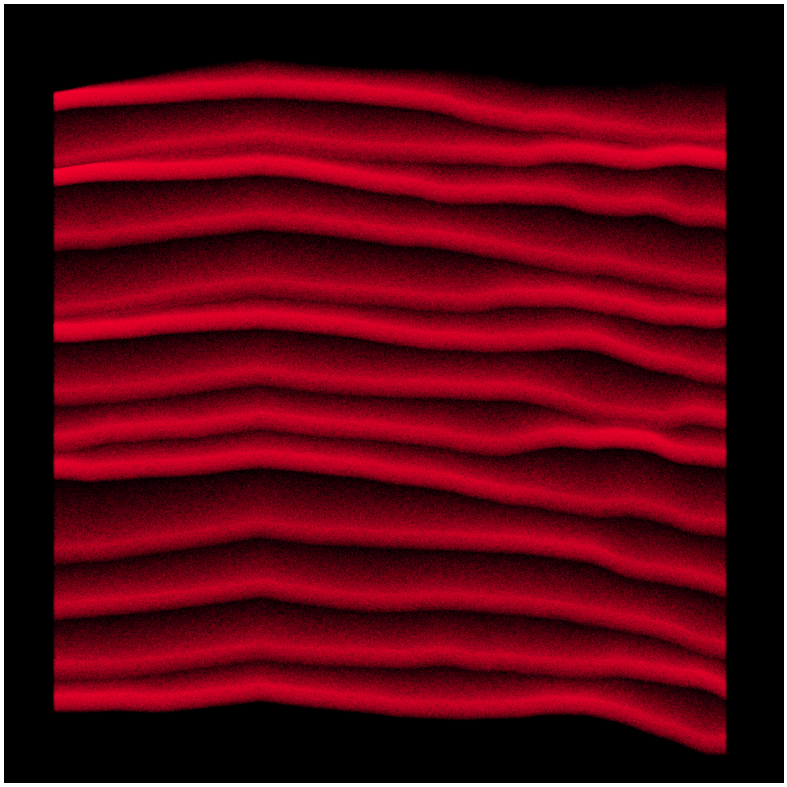


Figure 11. *Fabric 3* (2022). Digital art (computer-generated visual media).

of still images at 60 FPS (frames per second), after which the individual files were concatenated into a MOV file.

Multimedia (i.e., audiovisual) art is also a practical application of creative coding; however, whilst possible, it is not always as feasible as purely visual art in text-based environments such as Processing (it is much more effective in Max/MSP/Jitter). That said, integrating multiple creative coding environments into the creative process allows artists to make the most of each environments' strengths by combining source material; for example, the combination of an acousmatic electroacoustic work composed in SuperCollider with an animated 3-D

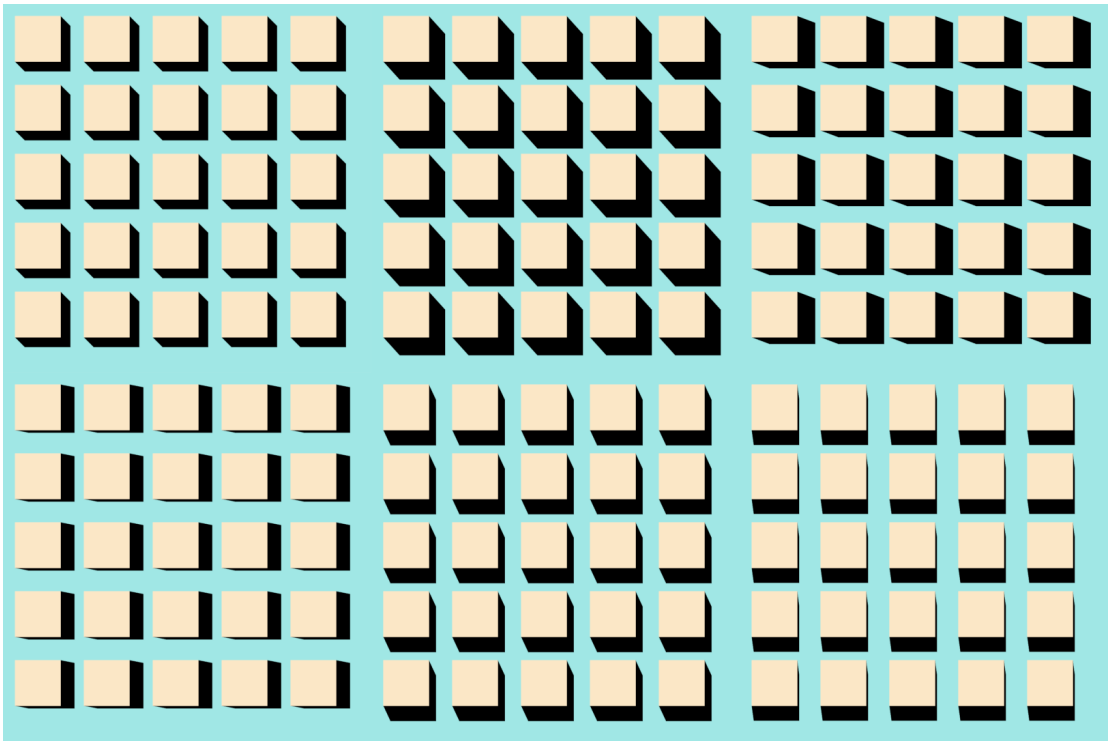


Figure 12. *Colour blocks and shadows* (2022). Digital art (computer-generated visual media).

Processing sketch. There are many opportunities for digital artists to integrate creative coding into their visual creative process using the environments mentioned above, and I have only touched on a few.

Creative coding's flexibility and extensibility in both the sonic and visual realms make it a robust creative tool. Environments such as SuperCollider and Processing can be used as artistic media for anything from class projects to large-scale installations and interactive performances. Learning how to code is a skill that every contemporary artist and composer should develop if they wish to keep up with the fast-moving pace of technology in the creative space.

Conclusion

This explication has taken an artistic research approach by investigating creative programming, experimental music and digital art both academically and practically. Such practice-based research was accomplished by describing my musical background and providing a theoretical and conceptual contextualization for and analysis of my work *Morphei Somnium*, which served as the practical component.

In using creative and academic practices to explicate my compositional language, what I have presented in this paper has achieved my objective, as I have provided a parallel insight into my creative process and the theoretical conceptualization thereof. Through this explication, I have demonstrated how creative programming can be integrated particularly in the composition of experimental music, but also in digital art, with practical examples. Thus, I conclude that creative programming as a

medium in general is an effective tool for modern and contemporary artistic expression.

With the variety of programming environments at the artist's disposal, they are presented with a powerful set of tools that can be integrated into their creative practice. This can be achieved with minimal obstruction given the availability of these free and open-source environments and the gentle learning curve that comes with most of them. Creative coding empowers both artists and programmers to get involved in experimental music and digital art, and forms a fruitful marriage between art and technology.

APPENDICES

Appendix A

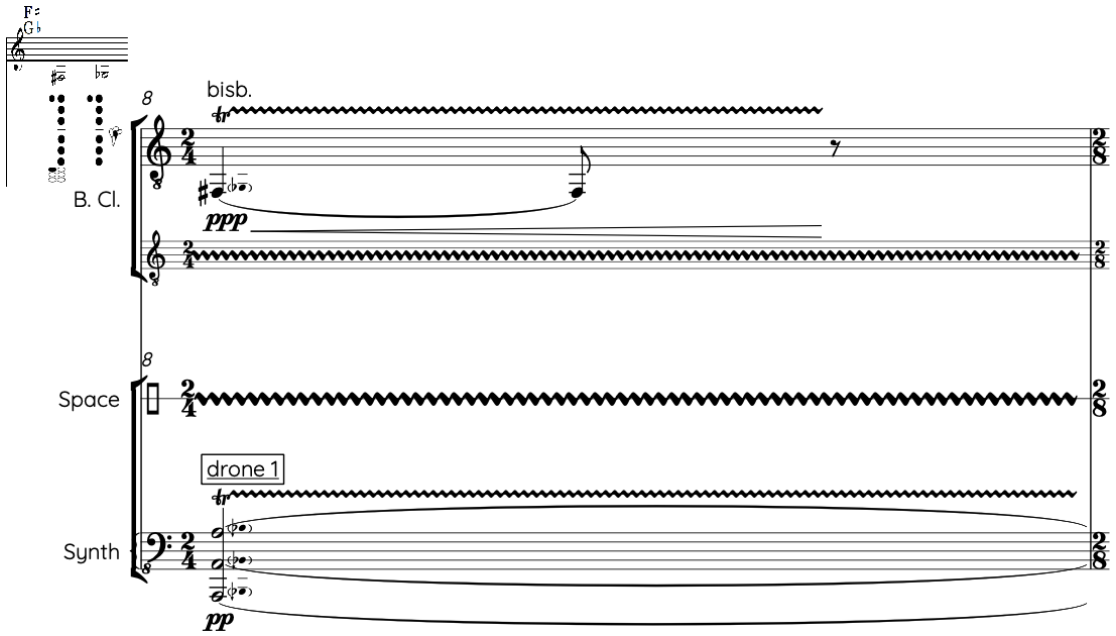
Programme Notes

morphei somnium (2021) is a sonic sketch depicting the domain of Morpheus, the primordial ancient Greek god of dreams, who takes many forms. The work mixes the dark tone and jagged lines of the bass clarinet with hypnotic drone textures, shimmering percussion, live signal processing and spatial immersion to reflect the mysterious and surreal nature of this sleepless fantasy.

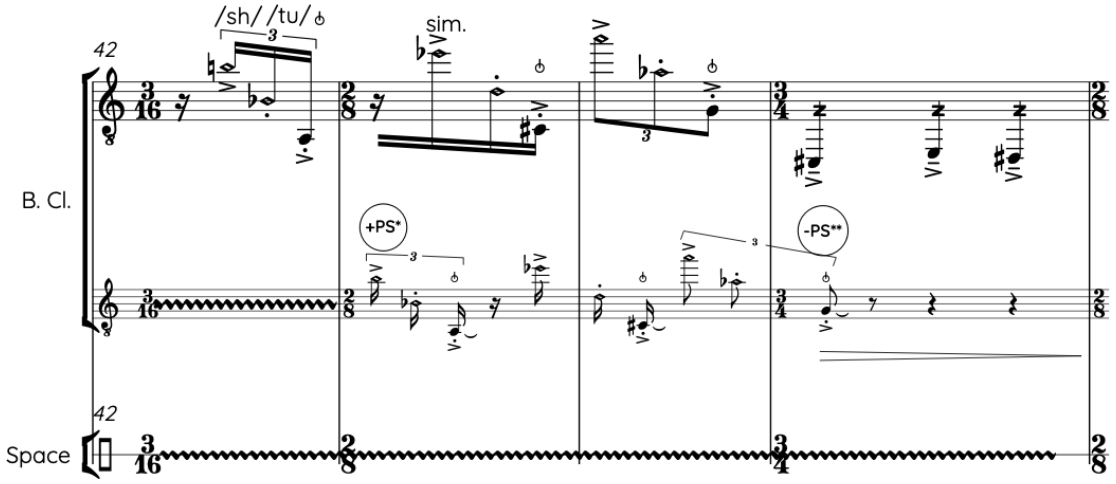
Appendix B

Score Snippets

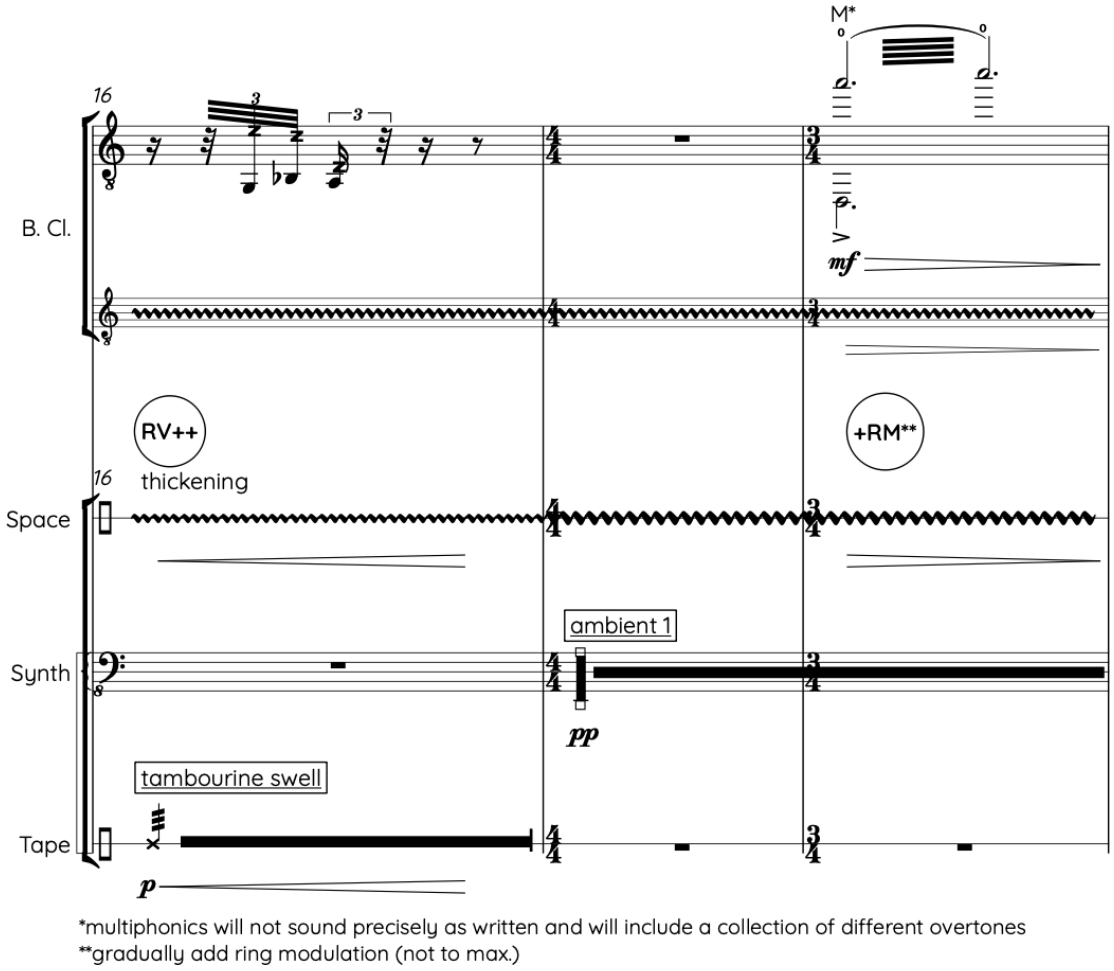
Morphei Somnium. Bass clarinet timbral trill and drone 1 synth, bar 8.



Morphei Somnium. Bass clarinet with pitch-shifting, bars 42–45.



Morphei Somnium. Change in landscape (tutti), bars 16–18.



Appendix C

Code Examples

```
// MORPHEI SOMNIUM: LAPTOP 1

OSCdef.new(\sp0, {
    // information for each OSC message response
    | msg |

    // confirm that there is an active micIn Synth
    // on the server before executing the code
    if ( (~micIn.isPlaying), {
        // when SP0 is selected in the UI, execute this code
        if ( (msg[1] == 1), {
            ~set0 = Routine({
                // update device UI to deselect other SP presets
                ~touchOSC.sendMsg('/spaces/1', 0);
                ~touchOSC.sendMsg('/spaces/2', 0);
                ~touchOSC.sendMsg('/spaces/3', 0);
                ~touchOSC.sendMsg('/spaces/4', 0);

                if ( (~micIn.isPlaying), {
                // reset microphone effects parameters
                    ~micIn.set(
                        \rmBlend, 0.2,
                        \xpos, 0,
                        \ypos, 0,
                        \delay_out, ~bus[\delay],
                        \out, ~bus[\reverb]);
                });

                // set effects arguments for SP0
                ~delaySynth.set(
                    \delayTime, 1,
                    \decay, 5,
                    \mix, 0.2);
```

```
                ~pitchShiftSynth.set(\mix, 0);
                ~reverbSynth.set(\mix, 0.4);

                // update device UI to display changes
                ~touchOSC.sendMsg('delayTime', 1);
                ~touchOSC.sendMsg(
                    'decay',
                    5.linlin(0, 10, 0, 1));
                ~touchOSC.sendMsg('delayMix', 0.2);
                ~touchOSC.sendMsg('pitchShiftMix', 0);
                ~touchOSC.sendMsg('ringModBlend', 0.2);
                ~touchOSC.sendMsg('reverbMix', 0.4);

                // select SP0 and print selection to post window
                ~touchOSC.sendMsg('/spaces/0', 1);
                "selected: space 0".postln();
            }).play();
        }, {
            // if the SP0 button is not selected
            // set default effects parameters
            ~setDefault.play();
            // print to post window
            "default space".postln();
        });
    }, {
        // if there is no active micIn Synth
        // deselect all other space buttons
        ~touchOSC.sendMsg('/spaces/1', 0);
        ~touchOSC.sendMsg('/spaces/2', 0);
        ~touchOSC.sendMsg('/spaces/3', 0);
        ~touchOSC.sendMsg('/spaces/4', 0);

        // print error message to post window
        "no input: turn on mic".postln();
    });
    // reset Routines for future use
    ~set0.reset();
    ~setDefault.reset();
},
// OSC path (where to send/receive messages)
"/spaces/0"
);`
```

Morphei Somnium. Default space OSCdef.

```
// MORPHEI SOMNIUM: LAPTOP 1

SynthDef(\delay, {
    | in = 10, atk = 0.1, sus = 1, rel = 5, gate = 1,
    delayTime = 0.5, maxDelay = 1, decay = 5,
    mix = 0.35, lpf = 5000, amp = 1, out = 0 |

    var delay;
    // read in signal from bus 1, four channels
    var sig = In.ar(in, 4);

    // define an envelope
    var env = EnvGen.kr(
        Env.asr(atk, sus, rel, [1, -1]),
        gate,
        doneAction: 0
    );

    // set up 16 delay lines, each with four channels
    16.do({
        delay = CombL.ar(
            sig,
            maxDelay,
            delayTime !4,
            decay
        );

        // low-pass filter at 5 kHz
        delay = LPF.ar(delay, lpf);
    });

    // blend dry and delayed signals
    sig = XFade2.ar(sig, delay, mix * 2 - 1);

    // apply envelope and amplitude
    sig = sig * env * amp;
    // pan randomly across four channels
    sig = SplayAz.ar(4, sig);

    Out.ar(out, sig);
}).add();
```

Morphei Somnium. Delay SynthDef.

```
// MORPHEI SOMNIUM: LAPTOP 2

SynthDef(\sine2, {
    | freq = 40, atk = 4, sus = 0, rel = 8, rq = 1,
    amp = 1, out = 0 |

    // define an envelope
    var env = Env(
        [0, 1, 1, 0],
        [atk, sus, rel],
        [1, 0, -1]
    ).kr(2);

    // create eight sine wave oscillators with
    // random frequencies between 40 Hz and 8 kHz
    var sig = SinOsc.ar({
        ExpRand(freq, freq * 20) } !8
    ) * 0.2;

    // apply a band-pass filter to limit
    // the oscillators to within audible range
    sig = BPF.ar(
        sig,
        freq.clip(20, 20000),
        rq,
        1/rq.sqrt()
    );

    // modulate the amplitude of the oscillators
    amp = SinOsc.kr({ ExpRand(0.01, 12) } !8
    ).range(0, 1);

    // apply envelope and amplitude
    sig = sig * env * amp;
    // pan randomly across four channels
    sig = SplayAz.ar(4, sig);

    Out.ar(out, sig);
}).add();
```

Morphei Somnium. Sine wave oscillator SynthDef (amplitude modulation).

```
// MORPHEI SOMNIUM: LAPTOP 2

SynthDef(\playBuf, {
    | buf = 0, atk = 0, sus = 0, rel = 3, rate = 1,
    freq = 400, startPos = 0, rq = 1, bpfMix = 0,
    xpos = 0, ypos = 0, amp = 1,
    pitch_shift_out = ~bus[\empty], delay_out = ~bus[\empty],
    out = 0 |

    // define an envelope
    var env = Env(
        [0, 1, 1, 0],
        [atk, sus, rel],
        [1, 0, -1]
        ).kr(2);

    // define a playback UGen
    var sig = PlayBuf.ar(
        1,
        buf,
        BufRateScale.ir(buf) * rate,
        startPos: startPos
        );

    // blend dry and band-pass filtered signals
    sig = XFade2.ar(
        sig,
        BPF.ar(sig, freq, rq, 1/rq.sqrt()),
        bpfMix * 2 - 1
        );

    // apply envelope and amplitude
    sig = sig * env * amp;
    // quadraphonic panning
    sig = Pan4.ar(sig, xpos, ypos);

    // out to pitch-shifting, delay and main output buses
    Out.ar(pitch_shift_out, sig);
    Out.ar(delay_out, sig);
    Out.ar(out, sig);
}).add();
```

Morphei Somnium. Buffer playback SynthDef.

```
// MORPHEI SOMNIUM: LAPTOP 2

Pbindef(\tamboLoop,
    \instrument, \playBuf,
    \buf, b[\tambourines][1],
    \dur, Pexprand(0.5, 1, inf),
    \atk, 1,
    \sus, 1,
    \rel, 3,
    \rate, Pexprand(0.25, 2, inf),
    \startPos, 0,
    \freq, 30 * Pexprand(1, 20, inf),
    \rq, Pwhite(0.05, 0.1, inf),
    \bpfmix, 0.5,
    \xpos, 0,
    \ypos, 0,
    \amp, Pexprand(0.3, 0.6, inf),
    \delay_out, ~bus[\delay],
    \out, ~bus[\reverb]
);
```

Morphei Somnium. Tambourine loop Pbindef.

```
// User Reddit: https://www.reddit.com/user/LaPuissanceDuYaourt/
// Example: https://www.reddit.com/r/processing/comments/t4exp4/
// strata_code_included/

float yOffset = 0.0;
float boundSpacing = 15;
float boundFuzz = 25;
color strokeColor = #FC2E20;
int weight = 1;
int margin = 75;
int noiseInc = 50;
float noiseScale = 0.003;
float highest = margin;
float xOffset = 0.25;
ArrayList<ArrayList<Float>> noiseVals = new ArrayList<ArrayList<Float>>();

void setup() {
  size(1080, 1080, P2D);
  smooth(8);
  background(0);
}

void draw() {
  noFill();
  strokeWeight(weight);
  stroke(strokeColor);

  while (highest < height - noiseInc) {
    ArrayList<Float> toAdd = new ArrayList<Float>();
    float base = highest;

    for (int i = margin; i < width - margin; i++) {
      float widthFactor = 1.5 * abs(width/3 - i) / (width/3);
      float n = base + widthFactor * noiseInc * noise((i + xOffset)
      * noiseScale);
      // point(i, base + 50 * n);

      toAdd.add(abs(n - 1));

      if (n > highest) {
        highest = n;
      }
    }

```

```
45      noiseVals.add(toAdd);
46      xOffset += (width * 0.75);
47    }
48
49    for (int i = 1; i < noiseVals.size(); i++) {
50      ArrayList<Float> valRow = noiseVals.get(i);
51
52      for (int j = 0; j < valRow.size(); j++) {
53        float lowBound = i > 0 ? noiseVals.get(i - 1).get(j) : 75;
54        float highBound = valRow.get(j);
55        float fuzz = random(boundFuzz);
56        float y1 = lowBound
57        + random(highBound - boundSpacing + fuzz - lowBound);
58        float y2 = y1 + random(highBound - boundSpacing + fuzz - y1);
59
60        point(margin + j - 7 + random(4), y2); //, r, r);
61        // line(margin + j - 6 + random(6),
62        y1, margin + j - 6 + random(6), y2);
63      }
64    }
65  }
66
67  void keyPressed() {
68    if (key == ENTER) {
69      background(0);
70      noiseSeed((long) random(1000000000));
71    }
72
73    if (key == TAB) {
74      saveFrame("frames/strata_FC2E20-####.png");
75    }
76  }
```

Fabric 3. Processing sketch. Modified code originally by a Reddit user (see credit on line 1).

```
// Reddit User: https://www.reddit.com/u/LaPuissanceDuYaourt
// Example: https://www.reddit.com/r/processing/comments/uceyrt/
// shadow_cubes_tribute_to_rudolph_de_harak_code/
// GitHub: https://github.com/Brian-Fearn/Processing/tree/main/ShadowCubes

int margin = 35;
int perSide = 5;
PGraphics pg;

void setup() {
  size(1080, 1080, P2D);
  pg = createGraphics(1080, 1080, P2D);
  pg.smooth(8);
  frameRate(60);
}

void draw() {
  pg.beginDraw();
  pg.background(#59981A);

  int sidePlusShadow = (pg.width - margin * 2) / perSide;
  int hSide = int(sidePlusShadow * 2.0 / 3);
  int vSide = int(sidePlusShadow * 2.0 / 3);
  float h = 0.1 + 0.9 * 0.5 * (1 - sin(PI + radians(frameCount * 0.5)));
  float v = 0.1 + 0.9 * 0.5 * (1 - sin(radians(frameCount)));
  int hShadow = int(hSide / 2.5 * h);
  int vShadow = int(vSide / 2.5 * v);

  pg.noStroke();

  for (int x = margin; x <= pg.width - margin; x += sidePlusShadow)
  {
    if (x == margin) {
      if (x < pg.width - margin) {
        for (int y = margin; y < pg.height - margin; y += sidePlusShadow)
        {
          int px = x;
          int py = y;

          pg.fill(0);

          pg.push();
          pg.translate(px, py);
```

```
44
45            pg.beginShape();
46            pg.vertex(0, vSide);
47
48            pg.quadraticVertex(0, vSide, hSide, vSide);
49            pg.quadraticVertex(hSide, vSide, hSide, 0);
50            pg.quadraticVertex(hSide, 0, hSide, -vSide);
51            pg.quadraticVertex(hSide, -vSide, hSide, 0);
52
53            pg.vertex(hSide + hShadow, vShadow);
54            pg.vertex(hSide + hShadow, -vShadow);
55            pg.vertex(hSide + hShadow, vSide + vShadow);
56            pg.vertex(hShadow, vSide + vShadow);
57
58            pg.endShape(CLOSE);
59            pg.fill(#ECF87F);
60            pg.square(0, 0, hSide);
61            pg.pop();
62          }
63        } else if (x == pg.width - margin) {
64          for (int y = pg.height - margin; y > margin; y -= sidePlusShadow)
65          {
66            int px = x;
67            int py = y;
68
69            pg.fill(0);
70
71            pg.push();
72            pg.translate(px, py);
73
74            pg.beginShape();
75            pg.vertex(hSide, 0);
76
77            pg.quadraticVertex(hSide, 0, hSide, vSide);
78            pg.quadraticVertex(hSide, vSide, 0, vSide);
79            pg.quadraticVertex(0, vSide, -hSide, vSide);
80            pg.quadraticVertex(-hSide, vSide, 0, vSide);
81
82            pg.vertex(hShadow, vSide + vShadow);
83            pg.vertex(-hShadow, vSide + vShadow);
84            pg.vertex(hSide + hShadow, vSide + vShadow);
85            pg.vertex(hShadow + hShadow, vShadow);
86
87            pg.endShape(CLOSE);
88            pg.fill(#ECF87F);
```

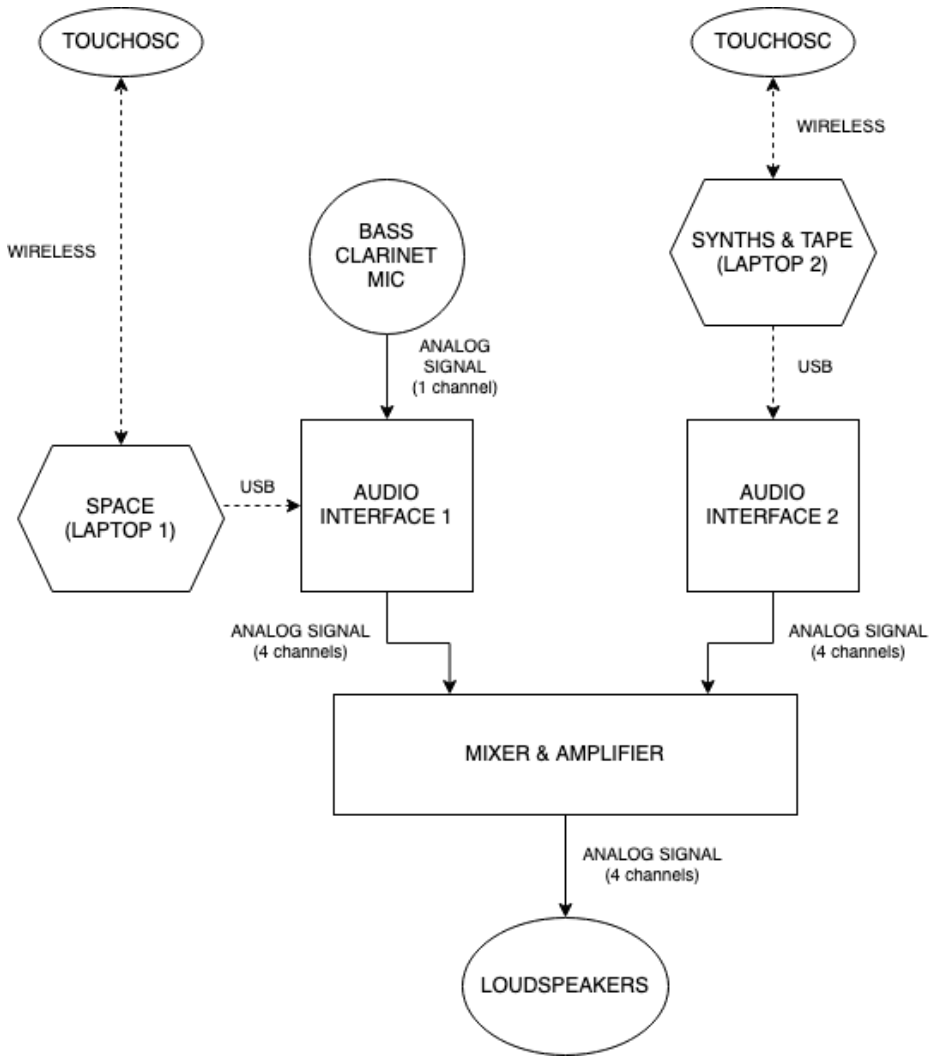
```
89            pg.square(0, 0, hSide);
90            pg.pop();
91          }
92        }
93      }
94
95      if (x < pg.width - margin) {
96        for (int y = margin; y < pg.height - margin; y += sidePlusShadow)
97        {
98          int px = x;
99          int py = y;
100
101         pg.fill(0);
102
103         pg.push();
104         pg.translate(px, py);
105
106         pg.beginShape();
107         pg.vertex(0, vSide);
108
109         pg.quadraticVertex(0, vSide, hSide, vSide);
110         pg.quadraticVertex(hSide, vSide, hSide, 0);
111         pg.quadraticVertex(hSide, 0, hSide, -vSide);
112         pg.quadraticVertex(hSide, -vSide, hSide, 0);
113
114         pg.vertex(hSide + hShadow, vShadow);
115         pg.vertex(hSide + hShadow, -vShadow);
116         pg.vertex(hSide + hShadow, vSide + vShadow);
117         pg.vertex(hShadow, vSide + vShadow);
118
119         pg.endShape(CLOSE);
120         pg.fill(#ECF87F);
121         pg.square(0, 0, hSide);
122         pg.pop();
123       }
124
125       if (x == pg.width - margin) {
126         for (int y = pg.height - margin; y > margin; y -= sidePlusShadow)
127         {
128           int px = x;
129           int py = y;
130
131           pg.fill(0);
132
133           pg.push();
```

```
134             pg.translate(px, py);
135
136             pg.beginShape();
137             pg.vertex(hSide, 0);
138
139             pg.quadraticVertex(hSide, 0, hSide, vSide);
140             pg.quadraticVertex(hSide, vSide, 0, vSide);
141             pg.quadraticVertex(0, vSide, -hSide, vSide);
142             pg.quadraticVertex(-hSide, vSide, 0, vSide);
143
144             pg.vertex(hShadow, vSide + vShadow);
145             pg.vertex(-hShadow, vSide + vShadow);
146             pg.vertex(hSide + hShadow, vSide + vShadow);
147             pg.vertex(hShadow + hShadow, vShadow);
148
149             pg.endShape(CLOSE);
150             pg.fill(#ECF87F);
151             pg.square(0, 0, hSide);
152             pg.pop();
153           }
154         }
155       }
156     }
157
158     pg.endDraw();
159
160     // saveFrame(
161       "./frames/59981A-ECF87F/shadow_cubes_59981A_ECF87F-####.png"
162       );
163
164     image(pg, 0, 0, width, height);
165
166     if (frameCount % 60 == 0) {
167       println(frameCount / 60 + " --- " + frameRate);
168     }
169   }
```

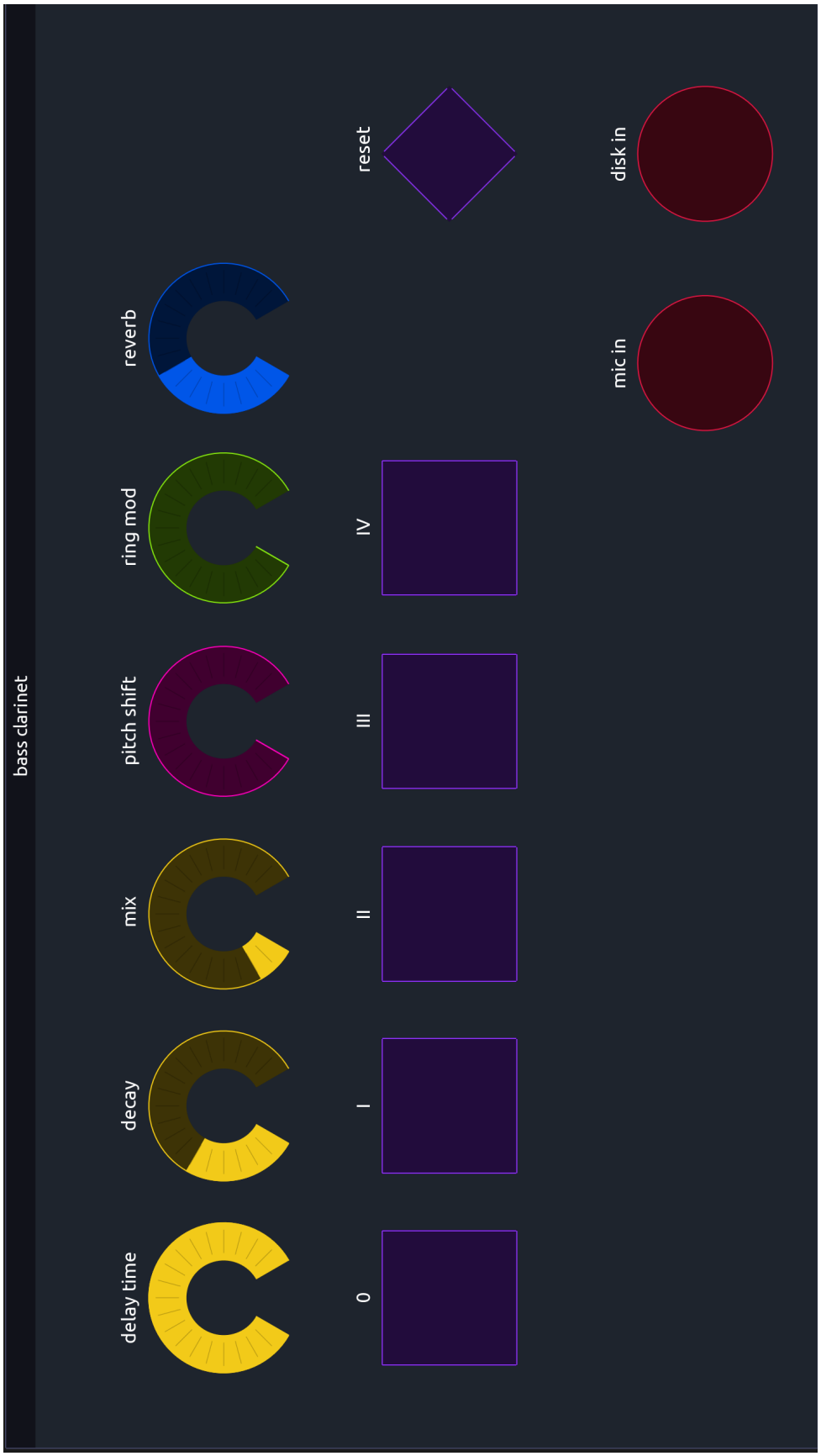
Colour blocks and shadows. Processing sketch. Modified code originally by a Reddit user (see credit on line 1).

Appendix D

Illustrations



Morphei Somnium. Performance signal flow.



Laptop 1 TouchOSC interface, default space.

Laptop 2 TouchOSC interface.

Glossary

acousmatic. Composed for projection through loudspeakers.

buffer. An object that is used to hold sampled audio, such as a sound file loaded into memory.

bus. A channel that carries audio or a control signal to connect UGens.

computer art. An artistic genre where computers are involved in creating the artwork.

Event. An Environment that represents an action.[90]

gesture. An energetic spatio-temporal trajectory.

one-shot. A musical sample or event that is sounded only once.

Routine. A Function that can return in the middle and then resume where it left off.[91]

sound object. A composite object, made up of a number of smaller sounds, often with a global ADSR shape.[92]

spectromorphology. The observation of spectral energies and shapes in space (in a musical context).

unit generator. An object that processes, controls or generates sound.

---

[90] *SuperCollider 3.12.2 Documentation*, s.v. 'Event', https://doc.sccode.org/Classes/Event.html.

[91] *SuperCollider 3.12.2 Documentation*, s.v. 'Routine', https://doc.sccode.org/Classes/Routine.html.

[92] Adrian Moore, *Sonic Art: An Introduction to Electroacoustic Music Composition* (New York: Routledge, 2016), 21.

Selected Works

Behrman, David. *Runthrough*. 1967–68. YouTube video. Published 15 July 2012. https://www.youtube.com/watch?v=pdhOMQ1huXk.

Cage, John. *4'33"*. 1952. YouTube video. Published 15 December 2010. https://www.youtube.com/watch?v=JTEFKFiXSx4.

———. *John Cage / "Credo in Us", performed by Third Coast Percussion*. 1942. YouTube video. Published 28 October 2011. https://www.youtube.com/watch?v=nL5Kym7hVFI.

———. *Music of Changes*, playlist. 1951. Selection of YouTube videos. Last updated 17 December 2022. https://www.youtube.com/playlist?list=OLAK5uy_kCIblLdqdZcieAt4jbqBGREMVYoZGFsK4.

Cardew, Cornelius. *The Great Learning*, Paragraph 7. 1971. YouTube video. Published 9 January 2019. https://www.youtube.com/watch?v=qIBG1A9B3Iw.

Feldman, Morton. *Piece for Four Pianos*. 1957. YouTube video. Published 26 November 2010. https://www.youtube.com/watch?v=cCTkzgSePiQ.

Gabrieli, Giovanni. *Giovanni Gabrieli – O Magnum Mysterium (1587)*. YouTube video. Published 16 September 2016. https://www.youtube.com/watch?v=orEwwkLnqZ4.

Grisey, Gérard. *Grisey Partiels Asko*. 1975. YouTube video. Published 22 June 2015. https://www.youtube.com/watch?v=1v7onrjN6RE.

LeWitt, Sol. 'Sol LeWitt: A Wall Drawing Retrospective'. MASS MoCA. Published 3 February 2022. https://massmoca.org/sol-lewitt/.

Schaeffer, Pierre. *Schaeffer: Five Studies of Noises* ['Cinq études de bruits'], playlist. 1948. Selection of YouTube videos. Last updated 18 December 2022. https://www.youtube.com/playlist?list=OLAK5uy_l_WO_K_ZH2KtSjt3gDIm-mnlNt0CYMQqM.

Stockhausen, Karlheinz. *Gesang der Jünglige – Karlheinz Stockhausen* ['Gesang der Jünglinge']. 1955–56. YouTube video. Published 9 December 2012. https://www.youtube.com/watch?v=UmGIiBfWI0E.

Stockhausen, Karlheinz. *Stockhausen – Kontakte*. 1958–60. YouTube video (with score). Published 2 May 2020. https://www.youtube.com/watch?v=l_UHaulsw3M.

Varèse, Edgard. *Edgard Varèse, Ionisation – Ensemble intercontemporain*. 1929–31. YouTube video. Published 19 December 2012. https://www.youtube.com/watch?v=wClwaBuFOJA.

Wolff, Christian. *Burdocks*. 1970–71. YouTube video. Published 20 March 2014. https://www.youtube.com/watch?v=LqIzAXSAYMw.

Bibliography

Arrell, Chris. 'The Music of Sound: An Analysis of "Partiels" by Gérard Grisey'. In *Spectral World Musics: Proceedings of the Istanbul Spectral Music Conference*, edited by Robert Reigle and Paul Whitehead, 318–33. Istanbul: Pan Yayıncılık, 2008. https://issuu.com/chrisarrell/docs/arrellpartielsanalysis.

Behrman, David, Crys Cole and Cleek Schrey. *Runthrough*. Videography by Yiyang Cao, audio recorded by Bob Bellerue, edited by James Emrick. Recorded live 21 July 2018. https://issueprojectroom.org/video/david-behrman-runthrough.

Boeck, Angelika. 'What is Artistic Research?' *w/k – Between Science & Art*. Published 25 February 2021. https://doi.org/10.55597/e6798.

Borries, Friedrich von. 'Artistic Research – Why and Wherefore?' *Journal of Science Communication* 14, no. 1 (2015): C06. https://doi.org/10.22323/2.14010306.

Cage, John. *Bacchanale*. New York: C. F. Peters, 1960.

———. *Credo in Us*. New York: C. F. Peters, 1962.

Cardinal, Serge. 'La recherche-création: une pensée audio-visuelle?' In *La recherche-création dans l'université du XXI^{e} siècle*. 80^{e} Congrès de l'ACFAS. Montreal, May 2012. https://www.creationsonore.ca/wp-content/uploads/2014/10/communications_serge-cardinal_la-recherche-creation.pdf.

Edmonds, Ernest. 'Algorithmic Art Machines'. *Arts* 7, no. 1 (2018): 3. https://doi.org/10.3390/arts7010003.

Gagné, Nicole V. *Historical Dictionary of Modern and Contemporary Classical Music*. 2nd ed. Lanham, MD: Rowman & Littlefield, 2019.

Galanter, Philip. 'Generative Art Theory'. In *A Companion to Digital Art*, edited by Christiane Paul, 146–80. Hoboken, NJ: John Wiley & Sons, 2016. https://doi.org/10.1002/9781118475249.ch5.

———. 'What is Complexism? Generative Art and the Cultures of Science and the Humanities'. In *Proceedings of the 11th Generative Art Conference* [GA2008], 151–67. Milan, December 2008. https://generativeart.com/on/cic/papersga2008/13.pdf.

Goldman, Richard Franko. 'Review of *Varèse: Ionisation; Density 21.5; Intégrales; Octandre; Hyperprism; Poème Electronique* by Robert Craft & Varèse'. *The Musical Quarterly* 47, no. 1 (1961): 133–34. https://www.jstor.org/stable/740554.

Hartmann, Doreen. 'Computer Demos and the Demoscene: Artistic Subcultural Innovation in Real-Time'. In *Proceedings of the 16th International Symposium of Electronic Art* [ISEA2010 Ruhr], edited by Judith Funke, Stefan Riekeles, Andreas Broeckmann and Hartware MedienKunstVerein, 124–26. Berlin: Revolver, 2010. https://www.isea-archives.org/docs/2010/proceedings/ISEA2010_proceedings.pdf.

Héder, Mihály. 'AARON'. In *Encyclopedia of Artificial Intelligence: The Past, Present, and Future of AI*, edited by Philip L. Frana and Michael J. Klein, 1–2. Santa Barbara, CA: ABC-CLIO, 2021.

Herremans, Dorien, Chuan Ching-hua and Elaine Chew. 'A Functional Taxonomy of Music Generation Systems'. *ACM Computing Surveys* 50, no. 5 (2017): 69. https://doi.org/10.1145/3108242.

Knight-Hill, Andrew. 'Electroacoustic Music: An Art of Sound'. In *Foundations in Sound Design for Linear Media: A Multidisciplinary Approach*, edited by Michael Filimowicz, 303–26. New York: Routledge, 2019. https://doi.org/10.4324/9781315106335.

Landy, Leigh. 'Reviewing the Musicology of Electroacoustic Music: A Plea for Greater Triangulation'. *Organised Sound* 4, no. 1 (1999): 61–70. https://doi.org/10.1017/S1355771899001077.

———. *Understanding the Art of Sound Organization*. Cambridge, MA: MIT Press, 2007.

Lombardo, Vincenzo, Andrea Valle, John Fitch, Kees Tazelaar, Stefan Weinzierl and Wojciech Borczyk. 'A Virtual-Reality Reconstruction of "Poème Électronique" Based on Philological Research'. *Computer Music Journal* 33, no. 2 (2009): 24–47. https://muse.jhu.edu/article/266409.

Maconie, Robin. *Stockhausen: On Music*. London: Marion Boyars, 1989.

Mayuzumi, Toshiro. 'The Principles of Electronic Music'. Translated by Cathy L. Cox. *Contemporary Music Review* 37, nos. 1–2 (2018): 100–120. https://doi.org/10.1080/07494467.2018.1453344.

McCartney, James. 'Rethinking the Computer Music Language: SuperCollider'. *Computer Music Journal* 26, no. 4 (2002): 61–68. https://doi.org/10.1162/014892602320991383.

McCartney, James. 'SuperCollider: A New Real Time Synthesis Language'. In *Proceedings of the International Computer Music Conference* [ICMC'96], 257–58. Hong Kong, August 1996. https://quod.lib.umich.edu/i/icmc/bbp2372.1996.078/1.

Messiaen, Olivier. *The Technique of My Musical Language*. Translated from the French by John Satterfield. Paris: Alphonse-Leduc Éditions Musicales, 1944.

Nyman, Michael. 'Towards (a Definition of) Experimental Music'. In *Experimental Music: Cage and Beyond*, 2nd ed., 1–30. Cambridge: Cambridge University Press, 1999.

Palombini, Carlos. 'Pierre Schaeffer, 1953: Towards an Experimental Music'. *Music & Letters* 74, no. 4 (1993): 542–57. https://www.jstor.org/stable/737576.

Paul, Christiane. 'Introduction: From Digital to Post-digital – Evolutions of an Art Form'. In *A Companion to Digital Art*, 1–19. Hoboken, NJ: John Wiley & Sons, 2016.

———. 'New Media'. In *Encyclopedia of Aesthetics*, 2nd ed., edited by Michael Kelly. Oxford University Press, 2014. https://doi.org/10.1093/acref/9780199747108.001.0001.

Pierre-Louis, Jean Philippe. '"Puntito": deux œuvres audiovisuelles portées par l'expérience de l'individualité'. Master's thesis, Université de Montréal, 2018. https://papyrus.bib.umontreal.ca/xmlui/handle/1866/22871.

Riley, Terry. *In C*. Tucson, AZ: Celestial Harmonies, 1989.

Schaeffer, Pierre, ed. 'Vers une musique expérimentale'. Special issue, *La Revue musicale* (Paris), no. 236 (1957).

Stévance, Sophie and Serge Lacasse. *Research Creation in Music and the Arts: Towards a Collaborative Interdiscipline*. London: Routledge, 2018.

Stockhausen, Karlheinz. *Stockhausen – Four Criteria of Electronic Music*, playlist. 1972. Selection of YouTube videos. Published 11 April 2013. https://www.youtube.com/playlist?list=PLRBdTyZ76lvAFOtZvocPjpRVTL6htJzoP.

Sullivan, Graeme. 'Art Practice as Research'. In *Art Practice as Research: Inquiry in Visual Arts*, 2nd ed., 95–120. Thousand Oaks, CA: Sage, 2010.

Varèse, Edgard and Chou Wen-chung. 'The Liberation of Sound'. *Perspectives of New Music* 5, no. 1 (1966): 11–19. https://doi.org/10.2307/832385.

Verostko, Roman. 'Algorithmic Fine Art: Composing a Visual Arts Score'. In *Explorations in Art and Technology*, 2nd ed., edited by Linda Candy, Ernest Edmonds and Fabrizio Poltronieri, 77–83. London: Springer, 2018. https://doi.org/10.1007/978-1-4471-7367-0_7.