Musical Mapping of Two-Dimensional Touch-Based Control Layouts

Thor Kell

Music Technology Area
Schulich School of Music
McGill University
Montreal, Canada

September 2014

A thesis submitted to McGill University in partial fulfillment of the requirements for the degree of Master of Arts in Music Technology.

2014/09/30

Abstract

This thesis presents an in-depth examination of musical mapping in two dimensions. Music has existed in two dimensions, in various forms of notation, for hundreds of years. Somewhat more recently, as the touchscreen has reached consumer levels of affordability, music has become controllable via interactive two-dimensional surfaces. This work examines how musical parameters are mapped to and controlled by these surfaces, including interactive touchscreens and static scores.

Three reviews of mapping choices in two dimensions are presented. The first provides a detailed review of mappings used in music applications that run on Apple's iOS operating system, focusing on the most popular applications. The second again reviews iOS music applications, but casts a much wider net, reviewing all music applications at a much higher level. The third review examines notation practices with a focus on the graphic scores of the 20th century. Each of these reviews enumerates the mappings used, summarizes major mapping trends, and discusses unique mappings choices.

From the data obtained by these reviews, several abstracted control layouts are defined, in terms of the location of controls, the type of controls, and the mapping that follows from the layout. These abstractions are based on common, recurring control layouts that appear in many musical applications and in many notation techniques.

Finally, this thesis discusses the engineering process of building a machine learning model to recognize these layouts so as to be able to classify an arbitrary layout of buttons accordingly. An open, web-based API to access this algorithm is also created, as is a sample application, *Pattern Recognition*, that uses this API to classify and map layouts created by end users.

Résumé

La musique a été retranscrite de façons diverses sur des media à deux dimensions depuis des siècles. Avec la popularisation récente des écrans tactiles il est dorénavant possible d'utiliser des surfaces bi-dimensionnelles pour le contrôle interactif de la musique. Ce travail de thèse livre une analyse en profondeur des stratégie de contrôle (ou mapping) mises en oeuvre pour agir sur la musique grâce à des dispositifs à écrans tactiles.

Ce travail débute par trois revues bibliographiques détaillées, chacune se concentrant sur les stratégies de mapping musical en deux dimensions. Chaque revue livre une synthèse des tendances les plus répandues ainsi qu'une analyse des stratégies plus insolites. Tout d'abord, les applications musicales les plus populaires conçues pour l'iOS d'Apple sont examinées. Ensuite, l'étude est élargie aux autres applications musicales pour iOS, en utilisant une approche plus haut niveau. Enfin, les méthodes de notation graphiques du XXe siècles sont considérées.

Grâce à ces revues, cette thèse élabore plusieurs archétypes d'interfaces bi-dimensionnelles. Ces modèles généraux prennent en compte l'arrangement des éléments de l'interface dans le plan, les types de contrôles offerts, et les mises en correspondances résultant de la structure de l'interface.

Finalement, ce mémoire s'achève par une description de la mise au point d'un système de reconnaissance automatique de la structure d'une interface de contrôle. Une interface de programmation libre et offerte par un service web a été créée à cet effet. Une application de démonstration, "Pattern Recognition", a aussi été réalisée. Elle utilise des éléments de l'interface de programmation pour pour classer les structures spatiales d'interfaces et générer des stratégies de contrôle.

Acknowledgments

To my family, for being smarter than me.

Big love to The Echo Nest family: Brian Whitman, Tristan Jehan, Kurt Jacobson, Joe Gester, and many others. Special/magic thanks to Amanda Bulger and Elissa Barrett.

Thanks, of course, to my supervisor, Marcelo Wanderley, for infinite wisdom, citation/scholarship black magic, and a never-ending smile. Special thanks to Bob Hasegawa for the same for Chapter 4.

Montréal, toujours. Especially Nadia Pona & Cassandra Miller; also Teagan Schultz, Erin Gee, Mason Koenig.

Hugs to my comrades-in-arms at Music Tech, without whom I would have never survived: Vanessa Yaremchuk, Alastair Porter, Carolina Medeiros, Avrum Hollinger, Mahtab Ghamsari-Esfahani, Ben Bacon, Julian Vogels, Marcello Giordano, Mike Winters, Aaron Krajeski, Joe Malloch, Ian Hattwick, Bertrand Scherrer, Deborah Egloff, Håckon Knutzen, Emma Frid, and everyone else. Special thanks to Darryl Cameron for making it all work.

Thanks to Boston, for fine lodgings and fine companionship and the best bar in the world: Laurel Pardew, Charlie Van Kirk, Pran Bandi, Melinda Cross, Liv Gold, Blake Brasher, Michelle Qi, Nick Joliat, Paula Te, Emilo Jasso, Rich Whalley, Colin McSwiggen, Bayard Wenzel.

Bonus thanks to UVic: George Tzanetakis, Kirk McNally, Peggy Storey, Christopher Butterfield. Extra bonus thanks to SoundCloud 2010: Hannes Tydén, Eric Wahlforss, Robert Bönhke.

Contribution of Authors

Thesis regulations require that contributions by others in the collection of materials and data, the design and construction of apparatus, the performance of experiments, the analysis of data, and the preparation of the thesis be acknowledged.

The content of Chapter 2 was originally published in the Proceedings of Sound & Music Computing Coference 2013 [1]. My supervisor, Marcelo M. Wanderley, co-authored the paper. It has been re-edited to fit the structure of this thesis. Likewise, the content of Chapter 3 was originally published in the Proceedings of Sound & Music Computing Conference 2014 [2]. Marcelo M. Wanderley was again my co-author, and it has also been re-edited for this thesis.

The screenshots of iOS applications in Chapter 2, Chapter 3, and Chapter 5 are publically available on the iTunes store, and are reproduced here under fair dealing / fair use. The images of scores in Chapter 4 are likewise reproduced here under fair dealing / fair use. Permission from the composers or current rightsholders was obtained for images that excerpt a large amount of the score: these permissions are listed inline with each image.

Vanessa Yaremchuk provided machine learning wisdom for both the review methodology in Chapter 3 and the engineering in Chapter 6. The various engineering services and APIs used in Chapter 6 are footnoted or cited accordingly, as they are mentioned.

Contents

List of Figures

List of Tables

List of Acronyms

API	Application Programming Interface
DAW	Digital Audio Workstation
DJ	Disc Jockey
HTTP	Hyper Text Transfer Protocol
iOS	iPhone Operating System
MIDI	Musical Instrument Digital Interface
MPC	Music Production Center
OSC	Open Sound Control
REST	REpresentational State Transfer
SVM	Support Vector Machine

Chapter 1

Introduction

Electronics in general and computers in particular have a long and noble history of making music [3], from the first playback of *Daisy* to the current explosion of electronic music. Computers themselves are now highly mobile: an average 'phone' has dozens of times the processing power of yesterday's desktop machines. Furthermore, the mobile computing market is growing, whereas the desktop market remains constant [4].

It is thus no surprise that tens of thousands of music applications exist for mobile phones, most concentrated on the iOS platform [5]. Nor is it a surprise that more and more musicians are using these applications to perform, produce, and practice their music. The iOS store has recorded over 60,000,000,000 downloads of over 1,000,000 apps [6]. As will be seen in Chapter 3, about 40,000 of those apps are music applications. A *very* rough first approximation would suggest that 24,000,000 of those downloads are music applications.

Touchscreen devices like the iPhone and iPad (along with Android devices such as the Nexus 7, the Microsoft Surface, and other devices) present a particular challenge when designing music applications. The potential hardware inputs are limited: a capacitive multi-touch surface, potentially with an accelerometer, a microphone and one or more gyroscopes. Yet the software that can be applied to these inputs is variable and limitless. Does the application capture individual touches, complex gestures, or something in between? Does it use the additional sensors, singularly or combined, to provide deeper information about the state of device?

To make things yet more complex, the sonic output generated by the application soft-

ware is also essentially limitless. Anything from sample playback to detailed synthesis techniques can be used to create sound, limited only by the computational power of the device. Defining the relationships between these layers is known as mapping, and it is the primary focus of this thesis.

Hunt et al define mapping as "The art of connecting these two, traditionally inseparable, components of a real-time musical system" [7], referring to the control mechanism and sound generator, respectively. They further discusses the importance of this process in terms of how the player responds to the instrument. Hunt et al have also written about mappings in terms of live performance, and with regards to expert musical interaction. Other authors who have discussed mapping include Rovan et al [8], in terms of detailed mapping of gestures; and Bowler et al [9], in terms of interpolating between N input parameters and M synthesis parameters.

Mapping is a key component in the creation of touchscreen music applications. Indeed, a distinct subset of these music applications allow users to design and map their own layouts of controls. This flexibility allows for unique control systems, which in turn lead to unique music. However, these interfaces often include dozens upon dozens of buttons, and the process of mapping each button to a musical event is often painstaking at best.

This thesis researches the mapping process across a variety of two-dimensional media, with a final goal of automating the mapping of arbitrary layouts of controls to musical parameters. It must be noted that mapping is a very difficult problem [7]: this thesis does not propose to solve it, or even provide the 'best' mapping for a given layout of controls. It will, however, strive to provide a not-unreasonable, well-grounded automatic mapping process, while also providing deep insight into how musical mappings are currently designed and represented on touch devices.

In order to achieve this, two disparate sources are examined and reviewed, in order to cover a wide range of potential mappings: music applications on iOS, and twentieth century graphic scoring techniques. From these two sources, a set of abstracted control layouts are defined, with associated mappings. Finally, this thesis discusses the creation of software that uses machine learning to detect these layouts, and return an appropriate mapping for them.

1.1 Thesis Overview

This thesis is structured in five parts. The first two chapters form a review of mapping trends in iOS music applications. Chapter 2 presents a detailed review of the most popular 1,200 iOS applications, in terms of both the metaphor displayed to the user and the exact mapping of musical parameters used. Novel applications are also examined in detail in this review. Chapter 3 expands this review to all 38,750 (at the time of writing) music applications, albeit at a much higher level. Whereas Chapter 2 defines ten categories, Chapter 3 defines a further fifty-six classes, and reviews their mappings.

Chapter 4 presents a contrasting and complementary view to the mappings of music in two dimensions, examining the graphic score as a source of interface mappings. Sixteen scores, from acknowledged classics to bleeding-edge new works, are reviewed and their mappings discussed. As will be seen, some key mappings persist across both iOS applications and graphic scores. Chapter 5 takes these most persistent and popular mappings and abstracts them into their topological forms: the piano becomes two offset rows of buttons, the musical staff becomes a column of alternating buttons, and so on. These abstractions then provide the source material for the final, example software.

Chapter 6 provides academic context and engineering details for the example software. Machine learning methods are used to recognize the abstracted control layouts defined in the previous chapter, and then provide automatic mappings for them. This chapter discusses the classification algorithm itself, the process of building a web-facing classification API to access the algorithm, and *Pattern Recognition*, the user-facing application itself. This sample software will fulfill the goal, stated above, of automating the mapping process.

1.2 Contributions

The engineering efforts of the final chapter form the most practical contribution: a functional algorithm for applying automatic mapping of musical parameters to any layout of buttons, and an open classification API allowing other developers to access it. The *Pattern Recognition* software provides a practical example of the classification algorithm and the classification API.

The two reviews of iOS trends that provided the data for this algorithm also stand as major contributions. The high-level review in Chapter 3 provides an overview of the

entire 'Music' application space, and how the various applications therein are mapped. Furthermore, the classified text data and screenshots have been made publicly available as an aid to future research around music applications, the iOS ecosystem, and text-based classification. The detailed iOS review in Chapter 2 gives more in-depth descriptions of the mappings used in the most popular iOS music applications, and lists summaries of mappings trends.

Finally, Chapter 4 ties the two-dimensional aspects of mapping back to that most two-dimensional of music media: the score, written or printed on paper. Chapter 4 considers the graphic score as input interface, and lists and summarizes the mappings used in sixteen important scores. This novel review provides a balance to the iOS focus of the rest of the thesis.

Chapter 2

Musical iOS Applications: In-Depth Review

2.1 Introduction

This chapter presents a first, in-depth review of mappings and metaphors presented in musical iOS applications (apps). These apps are split into ten categories, based on the visual metaphor presented to the end user, and the mappings for each category are described. These two factors impact the relationship between the control layer and the sound creation layer: the mapping layer defines these relationships explicitly, but the visual metaphor presented drives the selection of mapping. Hunt et al. have written about the value of mappings in mediating between these two layers [10]. Fels et al. have discussed the value of metaphor in human-machine interactions, and how it can improve a performer's understanding of the mapping and the instrument [11]. On iOS devices, as will be seen, the metaphors tend to be exceedingly obvious: pianos and guitars abound.

Some applications, however, have non-obvious mappings (timbre control based on where a piano key is touched, for example) that a metaphorical piano does not have. Furthermore, the wide range of abstract applications make the question of metaphor (or lack thereof) a key one. Wessel and Wright have presented more abstract control metaphors, with a focus on the relationship of gesture and metaphor to the acoustic results [12]. Likewise, McGlynn et al. have written about the expressive possibilities of interfaces that are not modeled on existing metaphors [13]. Their paper does not explicitly mention mapping, but mapping

choices are inherent in each interface they discuss.

This chapter provides real-world insight into how metaphors and mappings are used for music making on iOS devices. It also offers suggestions as to how to best use this data to create effective iOS music apps, in terms of both standard and non-standard mappings.

2.2 Method

From the approximately 800,000 apps on the iOS app store [14], 1,200 music apps were chosen for review. These were selected by examining the 'Top Paid','Top Free', and 'Top Grossing' subsections of the iOS music app page. Each of those subsections lists 200 apps and differs across iPhone and iPad, giving 1,200 applications, with some small overlap. Of these music apps, 337 deal with music creation in some way. These 337 apps were looked at in detail. "Music creation" is given a broad scope here: any application that allows creative interaction with music, in real time or not, is counted. This includes karaoke applications, but does not include radio applications, simple sound recorders, fingerprinting apps, or artist themed apps.

A cursory overview of the apps indicated that they could be organized into categories based on overarching metaphor - the most obvious being piano apps. Each app was assigned a metaphor, and then the total number of apps for each metaphor were added up. The goal of this classification was to delimit categories that would have broadly similar mappings. As the numbers for each app were counted, it became clear that there were ten main categories, and then a large number of varied, heterogeneous apps. Indeed, outside of the ten categories (all of which had at least thirteen apps), the metaphor with the most apps was the violin, with two.

The final list of categories was as follows: Piano, DJ, Digital Audio Workstation (DAW), Music Production Controller (MPC) [15], Guitar, Drum Kit, Synthesizer, Sequencer, Karaoke, Amplifier Simulator (Amp Sim), and Other. For each category, the metaphor and the general mappings for the metaphor were examined. A number of apps from each category were looked at in detail in order to discover novel or additional mappings. All apps in the Other category were looked at in detail. Regardless of category, each app was analyzed in terms of the direction and layout of its mappings, giving an overview of how musical parameters are mapped in general.

Note that only a subsection of the applications with standard metaphors were down-

loaded and tested; their mappings are assumed to be consistent across the category. A larger subset of these applications were examined via their websites. However, *every* app in the Other category was looked at in detail. When an application could not be downloaded and tested by hand (due to hardware limitations or a cost above $25.00 CDN), it was examined via screenshots and video. Specifically, those applications are: *Korg iKaosillator, Rockmate, Ocarina 2*, and *Live FX*.

2.3 Metaphors

Table 2.1 contains an overview of the number of applications in each category. Note that the Other category has been split into apps that represent known acoustic instruments (a trumpet, for example), and apps that have no acoustic referent. It must also be noted that apps that appeared on both the iPhone and iPad are counted twice.

Table 2.1: Number of Apps per Metaphor, iPhone and iPad

Metaphor	iPhone	iPad	Total
Piano	25	43	68
DJ	17	15	32
DAW	14	16	30
MPC	14	14	28
Guitar	12	13	25
Drum Kit	7	14	21
Synthesizer	4	16	20
Sequencer	6	13	19
Karaoke	9	9	18
Amp Sim	5	8	13
Other	21	34	55
Other (Acoustic Instruments)	4	4	8
Total	138	199	337

As can be seen, piano apps are the standout category, followed somewhat surprisingly by DJ apps. The other two acoustic instruments, Guitar and Drum Kit, are below DAWs and MPC apps. This primacy of the electronic is perhaps not surprising given that iOS is an electronic platform, but it is belied by the massive popularity of piano applications. The piano may simply be such a well-known metaphor that it transcends the limitations of the iOS platform (lack of easy volume and timbre control, etc).

Continuing down the list are Synthesizers, Sequencers, Karaoke apps, and then Amp Sims - applications that mimic guitar amplifiers and effects pedals. In the Other category, a small subsection of apps mimics other acoustic instruments, again suggesting that non-acoustic metaphors are more dominant. The rest of the Other apps present no consistent metaphor.

The following sub-sections detail each category in terms of its metaphor and mappings, and discuss some of the variations within each category.

2.3.1 Piano

Piano apps display a traditional keyboard that plays discrete pitches. Pitches are mapped from left to right, low to high, in steps of one semitone. The vast majority of apps display a keyboard, though some simply display abstract circles (*Smule Magic Piano*). Playback of multiple pitches is possible. Volume control is generally not possible, nor is timbre control, though some apps offer a 'pedal' button, for sustained notes (*Piano Infinity*), or give control over the amount of reverberation added (*Piano Complete*). Some apps provide a toggle to switch between sounds - piano, grand piano, harpsichord, cat, dog, and so on (*Real Piano HD, Piano Infinity, Cat Piano Concerto*). Exact tuning control (A440 vs. A442, for example) is also sometimes available (*Real Piano HD*), and some apps give access to a synthesizer-esq pitch bend wheel and a mod wheel for real-time volume control (*Pianist Pro*). Solutions for volume control include a 'force based' volume control (*Real Piano HD*), and a volume control based on where the user strikes each key - higher up the key is softer, near the bottom of the key is louder (*Pianist Pro*). Some programs include teaching modes where notes fall from the top of the screen to the bottom, and must be played as they hit the bottom (*Smule Magic Piano, Piano Infinity*).

Fig. 2.1: *Cat Piano Concerto*, a typical Piano app.

2.3.2 DJ

These apps provide two virtual turntables, with a virtual mixer. The volume of each turntable is controlled by a vertical fader, with louder being higher. The mix between turntables is controlled by a horizontal fader. Play, stop, and pause commands are controlled by buttons. The speed of each turntable is controlled by a pitch fader; faster is towards the user for some apps (*djay*), matching a traditional turntable, and away from the user for other apps (*DJ Rig Free*). This fader is generally in percent. 'Pitch bends', small corrections to the speed of each turntable, are controlled by buttons. The user can touch the virtual turntable to scratch or backspin, but not to change the speed of the turntable (*DJ Rig Free*).

Fig. 2.2: *djay*, a typical DJ app.

2.3.3 Digital Audio Workstation

DAW apps provide a complete solution for producing music and working with audio. They often include synthesizers, sequencers, and MPCs, as well as effect sections and mixers. Some go so far as to include auxiliary sends (*Auria*). The key distinction between a DAW app and a full-featured sequencer is that DAWs work with recorded audio: audio is recorded with a traditional red 'Record' button, and represented in clips wherein time moves from left to right, and amplitude is represented vertically (*FL Studio Mobile HD, Music Studio Lite*).

Fig. 2.3: *Auria*, a typical DAW app.

2.3.4 MPC

These apps are based on the Akai **M**usic **P**roduction **C**enter [15] line, a classic of hip-hop production. They have some number of trigger buttons in a grid - traditionally 16 buttons in a 4 x 4 grid. These buttons play a user-configurable sample when triggered. The user typically records one line, then loops it and records another line. Tempo can be tapped in (*iMPC*) or set with a slider (*BeatPad Lite*). The app may have a dedicated mixer (*iMPC*), or set volume via a slider on each pad (*Rhythm Pad*). There may be a separate FX section (*DJ Soundbox Pro*), or deep synthesis control of each drum sound (*Impaktor*). Finally, instead of the traditional 4x4 grid, some MPC apps have fewer buttons (*Rhythm Pad* has 8).

Fig. 2.4: *iMPC*, a typical MPC app.

2.3.5 Guitar

Guitar apps display 'strummable' strings, and a fretboard. Frets are selected by holding down the appropriate area, and lower notes are placed to the left, as when holding a guitar. The lowest string is likewise placed closest to the user, and the strings are mapped vertically, again as when holding a guitar. Some apps provide direct access to complex chords via buttons (*Guitar!, Real Guitar Free*). Some apps provide vibrato by shaking the device (*Smule Magic Guitar*), and others allow effects via virtual pedals, with the timbre controlled by rotary knobs (*PocketGuitar*). Most apps do not provide timbral control or volume control.

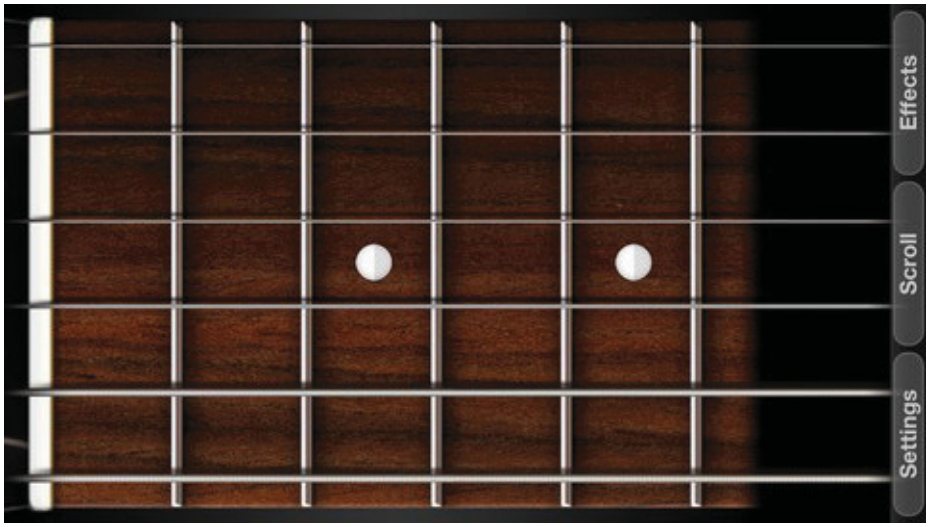


Fig. 2.5: *Pocket Guitar*, a typical Guitar app.

2.3.6 Drum Kit

These apps represent a traditional drum kit, with some number of drums. Tapping each drum plays an appropriate sample, or one of a set of appropriate samples for that drum. Rolls can sometimes be performed by sliding a finger on a drum head; a faster slide leads to faster rolls (*Ratatap Drums Free*). As with the piano apps, volume and exact timbre control are generally not available. However, some applications provide force-based volume control (*Ratatap Drums Free*), and some play differing samples based on the exact location of the tap - playing the bell vs. the edge of a cymbal, for example (*Drums!*). Finally, the user can often switch between drum kits or drum kit layouts (*Drum Kit Pro, Drums!*)

Fig. 2.6: *Ratatap Drums*, a typical Drum app.

2.3.7 Synthesizer

A synthesizer app exposes a selection of controls to a synthesis engine, and provides a piano-style keyboard for triggering the synthesized sounds. Control of the synthesis parameters is typically done with rotary knobs, but horizontal (*Alchemy*) or vertical (*Minisynth*) sliders, or XY pads (*Alchemy*) are also often used. Common parameters include:

- Wave type - sawtooth, sine, square, etc (*Magellan*)
- Filters - cutoff, type, resonance (*Alchemy*)
- Frequency modulation (*iMS20*)
- ADSR envelope control (*iMS20)*

In addition to triggering sounds with a piano keyboard, sequencers are included in some synthesizers (*Magellan, iMS20*), as are grids with volume mapped vertically (*Magellan*), and XY pads (*iMS20*). Indeed, some synthesizers can set the scale used by the keyboard or XY pad (*Animoog, iMS20*). In the case of the *Animoog*, this changes the layout of black and white keys. Finally, some synthesizers apps include extra effects, which are controlled with rotary knobs (*Magellan*) or with virtual patching environments (*iMS20, Audulus*).

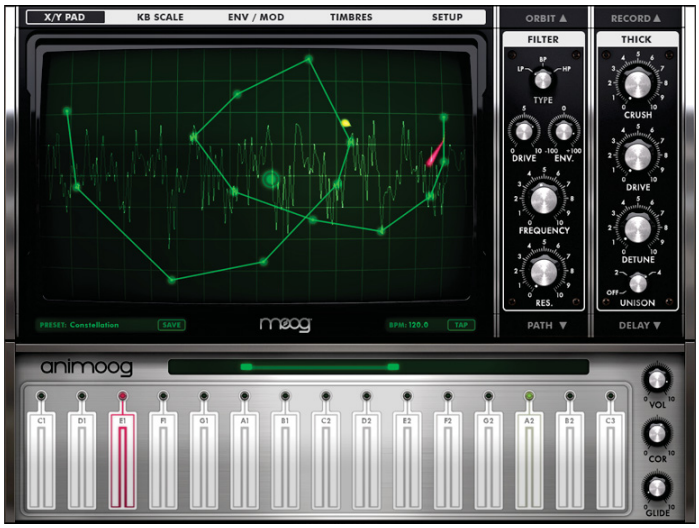


Fig. 2.7: *Animoog*, a typical Synthesizer app.

2.3.8 Sequencer

This category is inclusive of both drum machines and step sequencers. Time is divided into some number of discrete steps (16, 32, or 64), and time then advances step-by-step from left to right, according to a set tempo. One or more sounds or drums can be triggered on each step. Some sequencers model traditional drum machines (*Korg iElectribe*), and only allow access to a single track at a time, whereas others offer a grid with multiple tracks (*EasyBeats 2 Pro*). Some include DAW-style mixers with vertical sliders (*KeyZ*), some add effects sections with rotary control (*Molten Drum Machine*), and some have an MPC-style interface for adding events to the grid (*FunkBox Drum Machine*). The mapping of time also varies: some only display a single bar of time, whereas others allow a bar to be sequenced, and then allow the bar itself to be sequenced with other bars (*Genome MIDI Sequencer, DM1*). Zooming in time is occasionally provided by a rotary knob that controls the subdivision of a beat (*Molten Drum Machine*). Finally, volume per sound is sometimes controlled by the vertical position of the sound in the grid (*Looptastic Producer*).

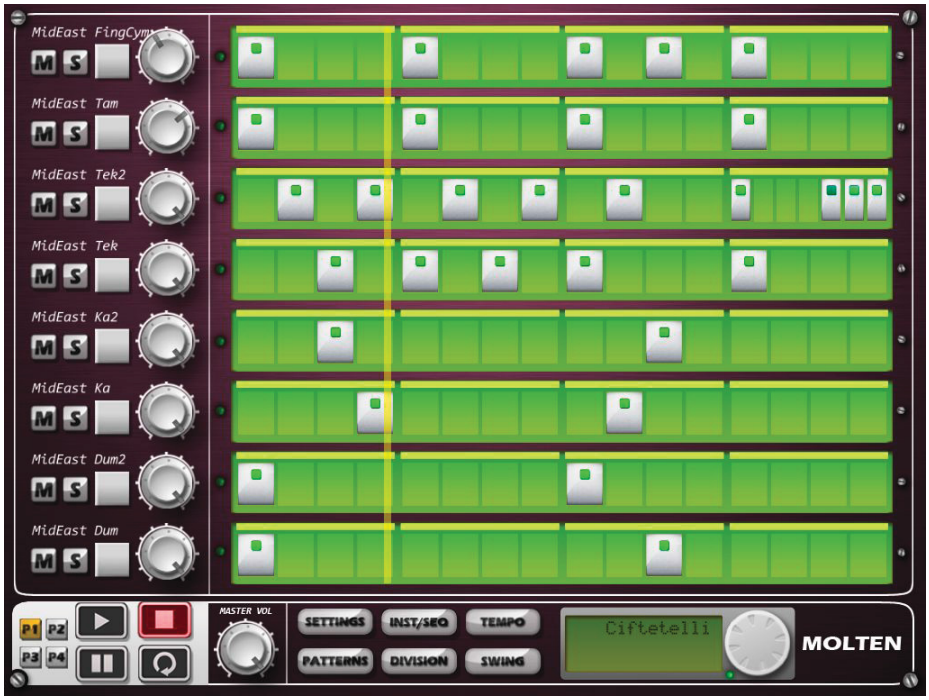


Fig. 2.8: *Molten Drum Machine*, a typical Sequencer app.

2.3.9 Karaoke

Karaoke apps allow the user to sing along to the instrumental track of a known song. At the very least, they present and somehow highlight the lyrics to be sung. Some provide visible pitch mapping, usually with pitch mapped vertically (higher notes are higher in pitch, lower notes are lower) and time moving from left to right (*StarMaker: Karaoke+*). Other options include additional reverb or echo (*Soulo Karaoke*), automating tuning effects that can be toggled on and off (*Sing! Karaoke, StarMaker: Karaoke+*), and toggles and level sliders for guide vocals (*StarMaker: Karaoke+*).

Fig. 2.9: *StarMaker: Karaoke+*, a typical Karaoke app.

2.3.10 Amp Sim

These apps provide some sort of model of a hardware FX box, usually a guitar pedal or guitar amplifier. Control of the effect is provided by rotary knobs (*AmpliTube*), horizontal faders (*AmpKit*), and on/off switches (*AmpliTube, AmpKit*). Some examples of the effects and parameters under control, from *AmpliTube*, are:

- Octave Pedal: direct level, octave level.
- Delay: Delay time, feedback, delay level.
- Phaser: speed.

Some apps additionally allow the user to position a virtual microphone in front of the virtual amplifier, providing nonlinear, two dimensional control of timbre (*Ultimate Guitar Amps and Effects*).

Fig. 2.10: *AmpliTube*, a typical Amp Sim app.

2.3.11 Other

The Other category ranges from touch-based implementations of acoustic instruments to wildly abstracted music applications. Violin, harmonica, and trumpet applications were examined, along with gravity-based sequencers, isomorphic pitch-space controllers, and granular synthesizer experiments. In general, the most atypical mappings appeared in this category. For example, *Rework* maps pitch radially out from the centre, and *ThumbJam* allows the user to add vibrato and tremolo by shaking the device. Another interesting example is the *ReacTable* app - here, the layout of control objects can be defined by the user, and their relative location controls how signal flows between them. This patching paradigm is used in many desktop applications, but in relatively few mobile applications.

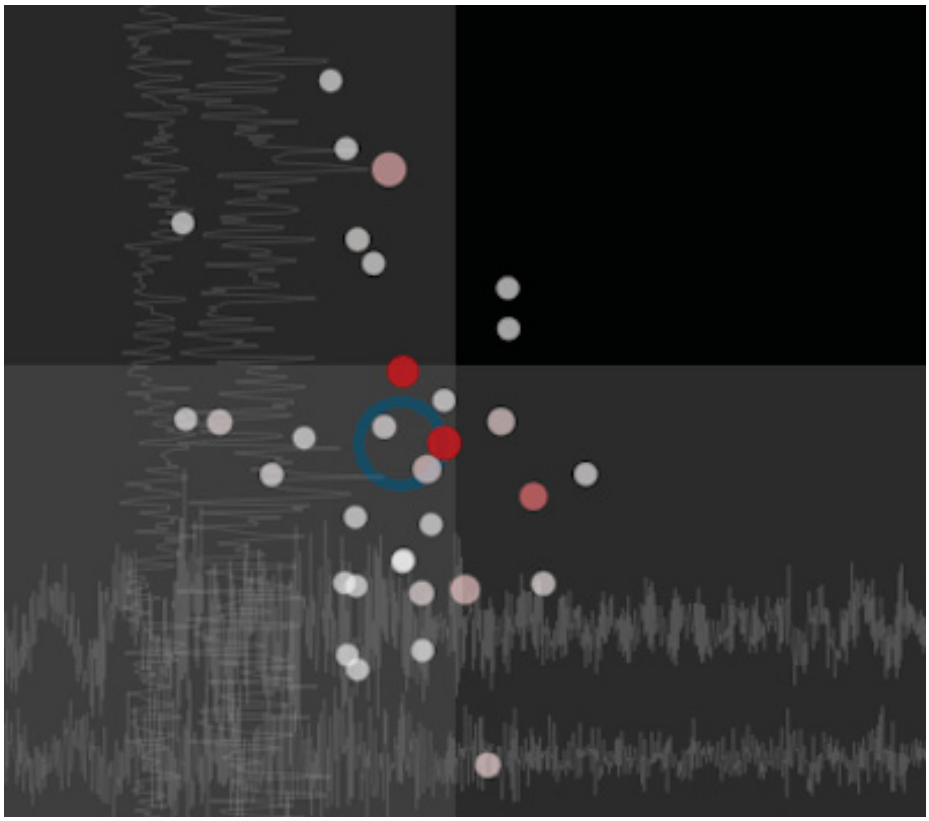


Fig. 2.11: *Borderlands*, an app from the Other category.

2.4 Mappings

2.4.1 Standard Categories

Beyond the metaphors listed above, the raw mappings behind each app were examined. For example, a standard piano application maps pitch horizontally from left to right (all directions given imply an increase), with discrete buttons. Likewise, a standard DAW application has a mixer that maps volume vertically, from bottom to top, continuously. Table 2.2 breaks down mappings in terms of pitch, trigger, time, volume, and timbre, across the ten metaphors listed above: Piano, DJ, DAW, MPC, Guitar, Drum Kit, Synthesizer, Sequencer, Karaoke, and Amp Sim.

It is important to note that some apps contain multiple mappings for a given parameter. Thus, the numbers in Table 2.2 will not add up to the total number of apps listed in Table 2.1. Secondly, despite the fact that many applications present rotary knobs or dials to control parameters (especially for timbral controls), these are not controlled in a rotary manner. They are in fact controlled as a vertical slider, and are notated here as such. Finally, some apps do not rotate when the device rotates. If the app presented a known metaphor (such as with guitar apps), the device was oriented to match the way the metaphorical instrument would be held. If the app presented no known metaphor, a best guess was taken, based on orientation of text, icons, and other visual cues.

In Tables 2.2 and 2.3, each column refers to the parameter to be mapped. Pitch, Trigger, Volume, and Timbre are self-explanatory. The Time column applies to applications like sequencers and DAWs that allow a user to queue or schedule events in time, and to tempo controls in DJ apps and sequencer apps.

Each row refers to the mapping used. Most are self-explanatory. The Known Layout mapping is less clear. It refers to controlling a parameter through some visual layout that does not fit in a simple horizontal or vertical mapping, but is nevertheless clear to the user. For example, a drum kit app would control timbre via a Known Layout - that of a drum kit. Likewise, a trumpet app that mimics the valves of a trumpet would control pitch via a Known Layout.

Table 2.2: Number of Mappings for Standard Categories

Mapping	Pitch	Trigger	Time	Volume	Timbre
Horizontal: Left-to-Right	143	0	67	32	0
Horizontal: Right-to-Left	0	0	0	32	0
Vertical: Top-to-Bottom	32	0	0	0	0
Vertical: Bottom-to-Top	73	0	0	142	114
Continuous	50	0	48	174	114
Discrete	178	0	19	174	114
Known Layout	0	0	0	0	49
Toggle	0	45	0	0	50
Touch	0	243	0	0	0
Gesture	0	43	0	0	0
Microphone	0	18	0	18	0

2.4.2 Standard Categories: Results

As can be seen from Table 2.2, the mappings for those standard categories do not cover a wide range of the possibilities. The runaway winner for pitch input, for example, is discrete pitches mapped left to right - almost certainly on a piano keyboard. It is important to note that mappings based on the keyboard are so common because users understand them instantly, without having to build up their own model for how an app maps pitch. Mapping pitch using a system of gestures would be interesting and novel, but would not be easy to use.

2.4.3 Other Category

In order to get a clearer view of potentially novel mappings, the raw mappings for each of the apps in the Other category (from Table 2.1) are listed in Table 2.3.

Most mappings listed in Table 2.3 are self-explanatory. The Touch Area mapping refers to the width-times-height area touched, in terms of the size: a tap with a pinky finger covers a smaller area than a thumb, for example. The Physics mapping refers to some model of the physical world: virtual balls bouncing with pitch matched to their speed, for example. Finally, the Location mapping refers to placing a virtual object at a certain XY location in the app: Moving a virtual loudspeaker closer to a virtual microphone, for example.

Table 2.3: Number of Mappings for Other Category

Mapping	Pitch	Trigger	Time	Volume	Timbre
Horizontal: Left-to-Right	22	0	15	4	11
Horizontal: Right-to-Left	0	0	0	0	1
Horizontal: Edge-to-Center	0	0	0	1	0
Vertical: Top-to-Bottom	2	0	1	0	1
Vertical: Bottom-to-Top	16	0	6	12	16
Rotation: Clockwise	2	0	5	0	0
Rotation: Counter-Clockwise	1	0	0	0	0
Radial: Center-to-Edge	2	0	1	1	1
Radial: Edge-to-Center	0	0	0	0	0
Diagonal: Bottom-Left-to-Top-Right	1	0	0	0	0
Continuous	9	0	18	17	28
Discrete	40	0	9	2	2
Known Layout	3	0	0	0	4
Toggle	1	7	0	0	16
Touch	0	26	0	0	0
Touch Area	0	0	0	1	0
Gesture	0	1	0	0	0
Microphone	0	9	0	3	0
Shake	1	2	0	1	0
Tilt	4	2	0	1	2
Physics	2	2	0	0	0
Location	0	4	1	0	1
Colour	3	0	0	0	2

2.4.4 Other Category: Results

As can be seen from Table 2.3, these mappings are substantially more creative than the mappings for known metaphors. Indeed, many new mappings appear, and some of them are used for only single apps. Standard horizontal and vertical mappings remain very popular, but in general, these apps are more interesting - though they may also be correspondingly more difficult for an end user to grasp.

2.5 Discussion

This categorization of applications has shown that the majority of iOS music applications are based on known metaphors, and that piano applications are by far the most popular, followed by emulation of electronic music interfaces: DJ rigs, DAWs, and MPCs. Taken as a single class, the Other category would be the second most popular category, just behind piano apps. However, as these apps vary from simple percussion apps (*iMaracas*) to sophisticated isomorphic pitch controllers (*SoundPrism*), it would be disingenuous to group them together and point to their high number as evidence of the power of novel metaphors. Further investigation of this category would be needed in order to draw more accurate conclusions.

To the contrary, this research indicates that simple or known mappings and metaphors, such as the all-powerful piano keyboard, are the most popular. Even complex synthesis applications emulate physical synthesizers, with sundry dials and faders for timbral control. In the Other category, where apps lack a common metaphor, standard horizontal or vertical mappings still appear. However, numerous apps present novel mappings and novel inputs, indicating that there is more design space to be explored outside of keyboards and drum kits. Indeed, regardless of their lack of known metaphor, apps like *Figure, Borderlands* and *Samplr* show that successful applications can be made with novel mappings.

The importance of metaphor cannot be overstated. The massive popularity of piano apps, DJ apps, and so on, can be explained by Fels et al. [11] and their discussion of how a metaphor provides the user with a "iterature" of common knowledge about the interface. This leads to transparency between the mappings and the user, which makes the mappings more effective for beginners. Wessel and Wright [12] discuss the value of metaphors in terms of organizing musical material. They also discuss the value of using abstract and creative metaphors to control parameters like pitch and timbre. As has been shown above, most iOS applications lack such a creative metaphor: only 55 apps out of 337 do not fit into known categories. It may be possible to bring new categories to life, however. The lack of success of, say, iPhone violins could be because no app has made the correct set of mappings with which to emulate a violin.

In terms of mappings, Tables 2.2 and 2.3 could be used to aid the design of new iOS applications. While it seems premature to relate these mappings directly to profitability and financial success (especially as the App Store does not provide sales numbers for each

app), the fact that the vast majority of applications map pitch from left to right indicates that an app aimed at widespread success should at least include such a mapping as an option. The same can be said for the mapping of volume vertically, and of time from left to right. Tables 2.2 and 2.3, however, could also be used to create spectacularly atypical iOS apps, simply by utilizing mappings that are under-represented. Such an app might map pitch from right to left, continuously, while controlling timbre via the microphone, and selecting rhythms via certain gestures. Or, the app might run time counter-clockwise, control pitch via the area of each touch, and map volume radially. These examples highlight the possibilities for deeply creative mapping solutions that exist on the iOS platform.

The most successful use of these tables, however, is probably in a combination of these two approaches. A scattershot, unfocused collection of novel mappings will probably result in a scattershot, unfocused app. However, an app with some traditional mappings and some novel mappings may be both more of a research success and more of a popular success. This would especially apply when using under-utilized controls such shaking and tilting, or when controlling underutilized parameters such as timbre.

Finally, it is also important to note the limitations of the iPhone and iPad hardware, and how those limitations impact mappings. Though capable of exceptional capacitive multi-touch input, iOS devices lack the ability to easily tell how hard a user is tapping them, or any way of giving the user tactile feedback on their input. In some cases, this leads to creative mappings to work around these limitations. For instance, *Smule Magic Piano* maps the timbre of each note vertically: touching higher up a key plays a darker sound. Likewise, *Ratatap Drums* uses data from the accelerometer to detect the force of a tap, and adjusts the volume accordingly.

2.6 Conclusion

This chapter has summarized the most popular categories, mappings, and metaphors for musical iOS apps. Data for this chapter was gathered in February 2013. It must be noted that the iOS App Store is an ever-changing world: the top 200 apps of February 2013 are almost certainly not the top 200 apps of July 2013 - and without question will not be the top 200 apps of 2015.

As of February 2013, however, there were a massive prevalence of piano apps, and of apps that show known metaphors to the user. There is also a subset of apps with no known

metaphor, which were, as a rule, the applications with the most creative mappings. Across all apps, the majority used simple mappings: pitch from left to right, volume from top to bottom, and so on. Even within the Other subset of apps, these simple mappings were the most popular. However, this subset also included deeply creative mappings, making use of tilting, physics models, radial lines, and more. These lists of mappings could be used to explore underutilized designed spaces on iOS and similar platforms.

Touch applications for music, on iOS and on other platforms, will only become more popular as such technology becomes more and more available. This chapter has helped expose how mappings and metaphors are currently used by the most popular apps on these devices. The next chapter expands this review to all music apps, though at a much higher level.

Chapter 3

Musical iOS Applications: High-Level Review

3.1 Introduction

This chapter details a high-level review of all iOS apps in the Music category. Each app is classified, and a summary of mappings across all such classifications is presented. The previous chapter reviewed the top 1,200 best-selling iOS apps, in terms of the interaction metaphor they presented to the user and the exact mappings that they use. Ten main categories were defined, and their mappings were delineated. This chapter continues that effort by providing basic classification for all 38,750 (as of January 28th, 2014) music apps, and a summary of mappings across all such classifications. This rather large scope was determined by the lack of access to smaller subsets of the data. The iTunes store lists the top 1,200 best-selling music apps, which was the data set used for the prior chapter. The only other data source is the iTunes website, which provided the data for all 38,750 apps.

Arner has examined a small subset of iOS apps, with a focus on their gestural interaction and uses of multitouch [16]. Approaching the problem from the other direction, Tanaka et al [17] have provided a survey-based analysis of how mobile devices of all sorts are used musically. In terms of classification, Zhu et al [18] have examined text and context-based machine learning methods for automatically classifying apps, and Chen & Liu [19] have used similar techniques to attempt to model how popular various types of applications are.

This overview will provide large-scale data on how musical mappings and metaphors are

defined on iOS. In addition to the ten categories defined in the previous chapter, this chapter defines forty new categories of musical apps and delineate their mappings. Moreover, in order to understand the iOS music ecosystem as a whole, this overview supplies broad classifications for 'music' applications that do not allow the user to create music, such as radio and artist apps. Summaries of the total number of apps for each category are provided, along with the total number of mappings across all apps, and thoughts on how to make use of this data when designing musical mappings and interfaces. A dataset for future use is also created, consisting of the title, URL, and descriptive text for each of the 38,750 apps, both with and without classification. This publicly available dataset will assist future studies of iOS applications, and of text-based classification techniques.

3.2 Method

Data was downloaded from the web-facing iTunes website[1], using a webcrawler built in Python with the Beautiful Soup[2] framework. In total, 38,750 apps were crawled. The app name, URL, and descriptive text were saved.

The analysis of this data had two goals. First, to find all apps that matched the ten categories listed in the previous chapter. These categories are: Piano, DJ, Digital Audio Workstation (DAW), MPC (A pad-based sampler/sequencer, based on the Akai MPC [15]), Guitar, Drum Kit, Synthesizer, Sequencer, Karaoke, and Amplifier Simulator (Amp Sim). 'Radio' and 'Artist' apps were added to this list, due to the large numbers seen during cursory examinations of the data. The hope was to train a classifier to recognize these twelve known categories. Once apps that matched these categories were found, the second goal would be attempted: to discovered and count new categories, ideally using K-Means or similar processes.

In order to achieve the first goal, several supervised machine learning methods were attempted, using both the TextBlob[3] and SciKit-Learn[4] [20] Python libraries. Training data was selected by examining apps that included the title of the category in their name or descriptive text, and then selecting apps that fit into the category in question. Twenty-five to fifty apps were selected for each category. Both Bayesian classification and Support

[1] https://itunes.apple.com/us/genre/ios-music/id6011?mt=8
[2] http://www.crummy.com/software/BeautifulSoup/
[3] http://textblob.readthedocs.org/
[4] http://scikit-learn.org/

Vector Machines (SVM) were trained on this data. Using only the name of each application as a feature proved ineffective, as did using the entire descriptive text. Table 3.1 shows these poor results for both Bayes and SVM.

In terms of the Bayesian classifier, this poor performance is probably due to both the very high number of features and a high level of inconsistent dependencies among the dataset [21]. SVM, on the other hand, performs poorly when the number of features is much larger than the number of training samples [20]. In this case, each class had only twenty-five to fifty samples, with 8,882 features.

A whitelist of words important to each category was thus constructed. The whitelist can be seen in Table 3.2. This reduced the number of features to 114. As Table 3.1 again shows, this whitelist improved both the Bayesian and SVM classifications, using both the app name and descriptive text to 90% accuracy, on the test dataset.

As the SVM model using both the app name and the descriptive text was producing good results on the test data, the next step was to run the trained model on the entire dataset. This was done category by category, in order to remove classified apps with each iteration. The results from this, as seen in the first column of Table 3.3, seemed reasonable at first blush. However, a manual examination of the remaining apps showed that many apps, especially Radio apps, were missed, suggesting that the models were overfitting to the test data. In hindsight, comparing the results between the columns of Table 3.3 show that some of the tested categories worked very well (Piano), while others did very, very badly (Radio).

Table 3.1: Accuray of Various Classification Methods

Method	Training Data	Whitelist	Accuracy
Bayes	App Name	False	0.55
SVM	App Description	False	0.31
Bayes	App Name	True	0.77
Bayes	App Description	True	0.88
Bayes	Both	True	0.90
SVM	App Name	True	0.57
SVM	App Description	True	0.83
SVM	Both	True	0.90

Table 3.2: Whitelisted Words per Category

Category	Whitelisted Words
Radio	Radio, Station, FM
Artist	Upcoming, Latest, Bio, Connected, Official, Exclusive, Fan, News, Band, Musician, Composer
Piano	Piano, Keyboard, Chord, Scale, Key, Note, Theme, Hand, Harpsichord, MIDI
Drum	Drum, Drumming, Kit, Drummer, Snare, Kick, Crash, Ride, Cymbal, Percussion, Percussionist, Beat, Roll, Hihat, Hi-hat, Brush, Stick, Bongo, Conga, Taiko
Guitar	Guitar, String, Strum, Strumming, Vibrato, Tremolo, Electric, Tab, Twang, Mandolin, Steel, Pedal
Karaoke	Sing, Song, Karaoke, Star, Catalog, Share, Recording, Stage
DJ	Turntable, Deck, Scratch, Mix, Mixer, Mixing, Cue, Crossfader, Sync, Beatmatch
MPC	MPC, Pad, Sample, Production, Akai
Sequencer	Sequence, Sequencer, Groovebox, Beatbox, Step, MIDI, Pattern, Tempo, BPM, Machine
DAW	Loop, Record, Recording, Audio, Band, Mixer, Aux, Produce
Synth	Analog, Analogue, Engine, Filter, Fat, Envelop, Synth, LFO, Polyphonic, Monophonic, Sine, Square, Triangle
Amp Sim	Rig, Cabinet, Mic, Stomp, Amp, Tube

Table 3.3: SVM Number of Apps per Category vs. Actual Number of Apps per Category

Category	Estimated	Actual
Radio	5288	10057
Piano	798	752
Drums	644	741
Karaoke	740	246
DAW	226	138
MPC / Sampler	220	136

These results were probably due to insufficiently trained models. Each category only had twenty-five to fifty apps to train on, and they were selected iteratively through the dataset, not at random. Radio apps, it would appear, are much more heterogeneous than the training data that was used.

In addition to attempting to classify known categories of applications, the second goal was to define new categories - ideally by clustering unclassified apps together. This was first attempted on test data, and did not give good results. Using SciKit-Learn's K-Means algorithm on the twelve categories of test data was ineffective, even when using the whitelisted name and the whitelisted description. The apps both failed to cluster in groups around their categories, and failed to give correct numbers of apps per cluster. Table 3.4 shows the number of apps per cluster, and Table 3.5 shows the categories per cluster. Figures 3.1a and 3.1b shows the results of this clustering, with its dimensionality reduced via principle component analysis (PCA). As can be seen, each cluster does not contain only a single category. It was also hoped that PCA might allow for manual segmentation of each category. However, as can be seen by the PCA of the data in Figure 3.1c, this was not possible: the categories are too intermingled to be able to draw useful segment boundaries.

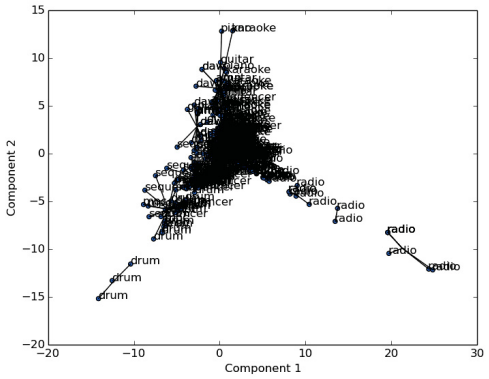
Given the difficulty clustering known data, perhaps due to K-Means' difficulty with clusters of varying shapes and densities (as seen in Figure 3.1a), clustering the entire dataset was even less viable, especially as the total number of categories was not known and the whitelist would be ineffective on these unknown categories. This left the analysis process with a somewhat effective method of classifying known categories, and an ineffective method of finding new categories.

Table 3.4: Clustering Results

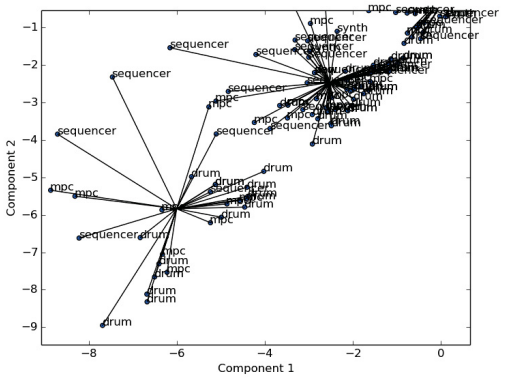
Cluster	Number of Apps	Number of Categories
1	6	1
2	94	9
3	29	3
4	191	12
5	3	1
6	14	1
7	27	6
8	57	7
9	24	2
10	63	5
11	19	6
12	2	1

Table 3.5: Clustering Breakdown

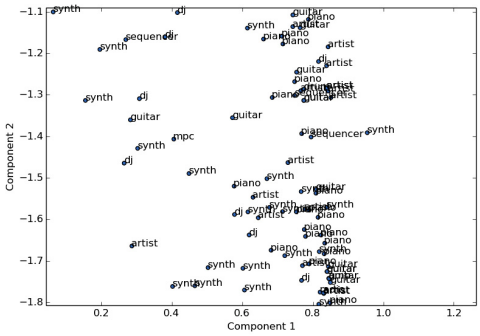
Cluster	Category Breakdown
1	Radio: 6.
2	Guitar: 29, Piano: 21, Karaoke: 14, DAW: 9, DJ: 8, Amp: 6, Synth: 3, Artist: 2, Sequencer: 2.
3	Drum: 15, MPC: 9, Sequencer: 5.
4	Artist: 67, Synth: 34, Piano: 25, DJ: 22, Guitar: 15, Sequencer: 13, Radio: 5, Amp: 3, MPC: 3, Drum: 2, DAW: 1, Karaoke: 1.
5	Drum: 3.
6	Radio: 14.
7	Karaoke 9, Amp: 8, Guitar: 5, Piano: 2, DAW: 2, Sequencer: 1.
8	Sequencer: 16, DJ 14, Synth: 10, MPC: 10, Drum: 4, DAW: 3.
9	Radio: 23, Artist: 1.
10	Drum: 26, MPC: 19, Sequencer: 13, Synth: 3, DAW: 2.
11	Amp: 9, DAW: 5, Piano: 2, DJ: 1, Guitar: 1, Karaoke 1.
12	Radio: 2.



(a) Labeled K-Means clusters.



(b) Labeled K-Means clusters, zoomed in.



(c) PCA data, zoomed in.

Fig. 3.1: K-Means & PCA results.

During this process it was discovered that, for a human, classifying each application based on the whitelisted name, the shortened app name in the URL, and the whitelisted descriptive text was simple to the point of being trivial.

For example, the below three strings strongly suggests an Artist application:

- " ", "amon-amarth-mobile-backstage", "official fan exclusive fan"

Likewise, these strings suggest a Amp Sim application:

- " ", "ampkit", "amp guitar amp electric guitar amp guitar amp mic pedal guitar recording share guitar fan"

In contrast, the below descriptive text suggests a Synth, but the name suggests a novel application.

- " ", "anckorage-spring", "audio connected keyboard midi engine midi midi midi midi"

In this case, the entire descriptive text was checked in a second step, and the application was then correctly classified as a Synth. The descriptive text is excerpted below:

"*Anckorage Spring is a physical modelling audio synthesiser based on the simulation of a set of connected mass- spring, integrating non-linearities, fluid and static friction, mechanical limits, gravity and bouncing. It is designed to be controlled by a continuous controller (like the Haken Continuum www.hakenaudio.com)...*"

Using this method it was found that 500 apps, with the use of a Python script to skip through them, could be shunted into the initial twelve categories in as little as fifteen minutes, giving a 'mere' thirty-nine hours to complete the task of manual classification. This is, of course, not to say that text-based machine learning is ineffective. Zhu et al [18] have made use of text data to successfully classify apps (though only 680 of them) of them, and Whitman et al [22] have used natural language processing and text-based machine learning on community metadata as a key component of their work in classifying music. The present paper, however, was simply looking to classify a large number applications with a high degree of accuracy, not investigate machine learning techniques. Manual classification also had the advantage of being completable in a known, though long, amount of time, whereas automatic classification presented a very open-ended problem. Furthermore, good, manually classified data could also be used as ground truth data for future investigations of text-based classifications and of the iOS app store.

It was thus decided to brute-force the problem. After the first 10,000 apps had been classified by hand, two heuristics were added to speed the process: apps that had 'radio' in the name were immediately defined as Radio applications, and apps that had whitelisted descriptive text of 'official update new connect' were immediately defined as Artist applications.

During this process, many applications could not be fit into the twelve known categories. Those were logged separately, and then examined with the full text of their names and descriptive text. Out of *those* apps, new categories (accordion apps, for example) were defined, based on the descriptive text. Totally novel apps were again logged separately. Due to time constraints, the 481 novel applications have not been examined in detail.

Once this two-tiered process was complete, each category was counted. In order to define the mappings for each category, the screenshots for each app in each category were downloaded and examined, again using a web crawler. In some few cases, (the Karinding, for example) videos of apps were examined in order to define the mappings. Only the general mappings for each category were defined. For example, if all but one Xylophone app maps pitch to the colours of the rainbow, the Xylophone category as a whole will be assigned this mapping of pitch to colour.

3.2.1 Verification

This classification process is not a perfect one. Even ignoring typos, this sort of fast human classification is prone to errors. In order to verify the quality of this method, 100 randomly selected apps were examined using their full name and full descriptive text: 94 were correctly classified, and 6 were incorrect. Then, 100 more randomly selected apps were tested, ignoring apps from the Radio, Artist, Media, and Non-English categories. Once again, 94 were classified correctly, and 6 were in error. It must also be noted that the Media, Educational, and Tool categories contain many interesting apps that are outside the scope of this thesis. More in-depth app reviews would be well served to begin with these categorizations - to say nothing of the various applications in languages other than English.

3.3 Results

3.3.1 Music-Making Applications

Applications that allow the user to produce music are, of course, the focus of this thesis. Table 3.6 shows the number of applications per category. Each of these categories also include applications with similar layouts. The 'Guitar' category, for example, also includes lute, banjos, mandolins, ukeleles, and so on. Categories that may require further explanation are listed below:

- Ball Sim - Apps that trigger sounds via a physics simulation of balls or other objects moving around.
- Chord Sequencer - Apps that allow the user to sequence symbolic representations of chords, either in guitar tablature or text / numeric format.
- Dulcimer - Western dulcimers, hammered dulcimers, and so on.
- Gamelan - Indonesian Gamelan instruments, include bells, gongs, and metallophones.
- Guqin - The guqin is an ancient Chinese zither, with angled strings.
- Hang - The hang is a modern pitched percussion instrument, similar to the steelpan.
- Kalimba - The kalimba, or thumb piano, is an African plucked percussion instrument.
- Karinding - The karinding is an Indonesian mouth harp.
- Looper - Apps that loop audio recorded by the user, rather than sequencing samples.
- Melodica - A reed-based wind instrument, with a small keyboard for selecting pitches.
- MIDI / OSC - Apps that output MIDI or OSC, to control other devices. As these apps vary wildly, their mappings are not included in the final count.
- Ocarina - The ocarina, a simple wind instrument, occurs in many cultures, but is perhaps most famous for its role in the 'Ocarina of Time' video game.
- Ondes Martenot - An early 20th century electronic instrument, featuring both ribbon and keyboard control of pitch.

- Steelpan - A pitched percussion instrument, originally from Trinidad & Tobago.
- Vuvuzela - A trombone-like South African instrument instrument, with a single pitch.
- Zither - Eastern zithers, including the guzheng, jentreng, qanun and gayageum.

Table 3.6: Number of Applications per Category, Musical Applications

Rank	Category	Number of Apps	Rank	Category	Number of Apps
1	Piano	752	26	Organ	25
2	Drum	741	27	Gamelan	24
3	Sequencer	606	28	Trumpet	23
4	Novel	481	29	Ball Sim	23
5	Guitar	385	30	Zither	17
6	Synth	277	31	Harmonica	16
7	Karaoke	246	32	Kalimba	15
8	Effect	149	33	Clarinet	13
9	DAW	138	34	Water Glasses	11
10	MPC / Sampler	136	35	Trombone	9
11	Xylophone	132	36	Dulcimer	9
12	DJ	119	37	Singing Bowl	9
13	Accordion	74	38	Cello	9
14	Band	67	39	Saxophone	8
15	Flute	67	40	Horn	7
16	MIDI / OSC	65	41	Melodica	6
17	Harp	47	42	Vuvuzela	5
18	Amp Sim	45	43	Ocarina	4
19	Bells	40	44	Washboard	4
20	Looper	36	45	Conductor	3
21	Chord Sequencer	36	46	Hang	3
22	Bagpipes	33	47	Pan Pipes	2
23	Notation	33	48	Ondes Martenot	2
24	Steelpan	33	49	Guqin	1
25	Violin	31	50	Karinding	1

As can be seen, this process discovered forty new app categories, for a total of fifty music-making categories in general. The mappings listed below represent a summary of the mappings for each category: a highly detailed discussion of each category is outside the scope of this thesis. Screenshots of examples of each category are also outside the page count for this thesis: they are available on the IDMIL website at idmil.org/projects/ios_mappings.

3.3.2 Non-Music-Making Applications

Broad categories were defined for apps that do not make music. These make up the majority of the music section of the app store. Table 3.7 shows the numbers of apps per category, and each category is defined below. This section also includes 'Junk' apps that are not music apps at all, and apps that were unclassifiable due to their descriptive text not being in English.

- Radio - Apps for a particular radio station, that assemble many radio stations, and so on.
- Media - Apps that deliver non-auditory media, allow for the playback of auditory media in a non-musical way, including soundboards, 'best songs' for a genre, and so on.
- Artist - Apps for promoting a particular artist, a group of artists, a festival, a recording studio, and so on.
- Non-English - Apps with descriptive text not in English, and thus not reviewable in this thesis.
- Educational - Apps for teaching an instrument, a theoretical concept, and so on.
- Tool - Apps for accomplishing music related tasks, including tuners, spectrum analyzers, and so on.
- Games - Apps for playing games about music or musicians
- Junk - Apps that have been mislabeled and are not music apps.

- Remote - Apps for remote control of non-musical audio systems, such as home theatre systems.
- Discovery - Apps for finding new music, new playlists, and so on.
- Christmas - Apps about Christmas.
- Print - Apps for a particular print magazine, or emulating a print magazine.
- Recorder - Apps for recording sound.
- Social - Apps for communicating about music on Twitter or other social media platforms.
- Fitness - Apps for controlling music while working out.
- Fingerprint - Apps for fingerprinting audio.

Table 3.7: Number of Applications per Category, Non-Musical Applications

Rank	Category	Number of Apps
1	Radio	10057
2	Media	7416
3	Artist	7161
4	Non-English	2806
5	Educational	2052
6	Tool	1406
7	Games	905
8	Junk	354
9	Remote	334
10	Discovery	272
11	Christmas	268
12	Print	249
13	Social	220
14	Recorder	154
15	Fitness	50
16	Fingerprinter	20

3.4 Mappings

Table 3.8 shows the total mappings, across all categories.

Table 3.8: Mappings for Musical Applications

Mapping	Pitch	Trigger	Time	Volume	Timbre
Horizontal: Left-to-Right	2142	0	1559	138	141
Horizontal: Right-to-Left	11	0	0	128	0
Horizontal: Center-to-Edge	15	0	0	0	0
Vertical: Top-to-Bottom	35	0	152	0	0
Vertical: Bottom-to-Top	1307	0	0	1483	1358
Diagonal: Bottom-Left-to-Top-Right	38	0	0	0	0
Rotational: Clockwise	0	0	9	0	0
Circular	33	0	0	0	0
Radial: Edge-to-Center	33	0	0	0	0
Grid	43	0	0	0	0
Vertical Size	105	0	0	0	0
Overall Size	33	0	0	0	0
Colour	78	0	0	0	12
Symbolic / Text	69	0	33	33	33
Continuous	190	0	612	1630	1498
Discrete	3931	0	1042	33	33
Playback	0	1104	0	0	0
Toggle	8	272	0	9	1207
Touch	0	3096	0	24	25
Gesture	0	86	0	10	2
Shake / Swing	0	20	0	20	0
Known Layout	167	0	0	0	746
Microphone Input	0	282	0	36	0
Audio Input	0	194	0	0	0
Force	0	0	0	81	0
Physics	12	23	0	23	18

Most mapping definitions used are self-explanatory. In terms of those that are less clear, a ‘Known Layout’ refers to an app that matches the visual layout of a real instrument, and maps some parameter based on this in a way that does not fit into any other category. For example, a drum application maps timbre based on a Known Layout - that of a drum kit. ‘Force’ here means methods of determining how hard the user is tapping the device,

often by polling the accelerometer or the microphone. A 'Gesture' indicates any motion more complex than a touch, typically a dragging or circular movement. When applied to the Volume parameter, this indicates that the speed of the gesture directly varies the volume of the sound. Finally, vertical mappings refer to the gesture used, not the metaphor presented: many apps present the user with rotary knobs which are actually controlled by vertical motion. This chapter has used the mapping throughout, rather than the metaphor.

3.4.1 Pitch

Pitch is dominated by keyboard-like, left-to-right or bottom-to-top mappings. Discrete pitches are likewise much more prevalent than continuous pitch. Some few apps increase pitch from top to bottom (zithers, for example), and even fewer increase pitch from right to left (trombones and pan pipes, in particular). Outside of these linear mappings, the next most popular mapping for pitch is the 'Known Layout' of wind instruments, which is usually abstracted to a set of 3-6 buttons that additively modify the pitch: pressing two buttons together gives a new pitch, rather than two pitches.

Mappings of pitch to colour are not uncommon, but a single dominant mapping of colour to pitch was not found. Likewise, mappings of pitch to size exist, but are always secondary to some other mapping (horizontal in the case of xylophones, and circular in the case of steelpans). Symbolic and text mappings are entirely based on various Western systems, including the sharps / flats of traditional staff notation, and various representations of chords (e.g. V^6, Dm7).

3.4.2 Trigger

Unsurprisingly, given that the primary interaction method on iOS devices is a touchscreen, mapping one touch to one sonic event is by far the most popular method for triggering sounds. Toggles are also popular, along with events or states that are often controlled by toggles, such as the playback of a sequencer or audio input from another device. Gestural mappings are not common, and mostly use simple movements: a circular motion to trigger a drum roll instead of a single drum hit, for instance. Making use of the device's other sensors, via a swing or a shake of the device, is not common. No applications were found that triggered sounds via a gesture made by moving the device itself - such mappings may, however, exist in one of the unexamined Novel apps.

3.4.3 Time

Time moves from left to right, and from top to bottom. Discrete time is slightly more prevalent than continuous time. Some very few apps map time rotationally, clockwise. Even in Notation apps, where time & rhythm are represented symbolically, the flow of time is from left to right.

3.4.4 Volume

Volume is dominated by vertical mappings, usually presented continuously. Some apps make use of force-based or shake / swing methods for determining volume. These, along with wind instrument apps that base volume on the input from the microphone, are the closest to 'real' acoustic instruments. An even smaller but more interesting mapping is that of touch / gesture to volume. For instance, some Gamelan apps allow for virtual bars to be muted by touching them in particular locations, and some Singing Bowl and Water Glass apps play louder sounds based on the speed of the triggering circular gesture.

3.4.5 Timbre

Like volume, timbre is mostly controlled vertically and continuously. Many apps use toggles to change between preset timbres (in piano apps, for instance), and many use Known Layouts to control the timbre of the sound played - drum kits are a prime example of this. Other timbral controls are much more rare. Surprisingly, colour is only used rarely for timbral control. However, like Volume, a very small number of apps use additional touches to control timbre. To continue the Gamelan example, some apps also allow for a muted timbre to be played if a virtual bar is touched before triggering it.

3.4.6 Summary

From Table 3.8 and the above paragraphs, it is clear that most apps use typical mappings: pitch from left to right, sounds triggered by touch, and volume / timbre controlled by vertical faders. Most of these mappings do not take advantage of sensors outside of simple touch and location. Complex gestures, microphone input, and shaking/swinging the device are used to control parameters from pitch to volume to timbre for a small number of applications, but are in general ignored. Likewise, most apps separate the control of each

parameter, mapping them to different controls and in different ways. Integral mappings are almost entirely ignored. The 481 apps in the Novel category have not been examined, however. They would almost certainly contribute to making Table 3.8 more varied.

3.5 Dataset

In order to further research around iOS music apps, the dataset and the Python scripts used to examine the dataset are publicly available The dataset (consisting of the name, URL, and descriptive text for each app) is provided, classified and unclassified, in order to allow for a wide variety of machine learning approaches and / or brute-force approaches. The complete collection of data and code can be found at idmil.org/projects/ios_mappings.

3.6 Conclusion

This chapter has provided a high-level review of all music apps on the iOS app store as of January 2014. This builds upon the prior chapter which provided an in-depth look at the most popular iOS music apps. This section has also provided the raw text data, classified and unclassified, for future research around text-based machine learning, app classification and more.

In the review itself, many new iOS instruments were discovered, representing extant acoustic instruments from bagpipes to zithers. A smaller number of new, purely electronic instrument categories, including loopers, chord sequencers, and bouncing-ball apps, were also discovered. The review also provided a high-level overview of mappings for each of these categories. This data can be used to understand how musical parameters are mapped on touchscreen devices, and thus influence how new musical applications are designed.

To be specific, the dominance of simple mappings is clear: pitch generally moving horizontally and discretely, volume and timbre moving vertically and continuously, and time moving from left to right. Although many applications make use of more complex mappings and more complex inputs, they are in a minority. This is a potentially rich area for innovation: one can easily imagine apps that use the microphone, accelerometer, and gyroscopes of iOS devices in new and interesting ways. Likewise, integral mappings for timbre and volume or non-traditional representations of pitch and time could both lead to interesting and innovative apps for making music.

Further work after such a high-level review is legion. A detailed examination of the Media, Educational, and Tool categories should be done, and would no doubt reveal sundry new ways to map musical parameters in tuning apps, how-to-play apps, and so on. Indeed, a deep dive into each of the main categories described above could provided further detail about how each category maps parameters. Likewise, the 481 Novel applications should be examined in detail, in the style of the previous chapter.

Touchscreen devices, and iOS in particular, are here to stay, and the ability of these platforms to create music at all levels of sophistication is only going to grow. Music applications, however, are not the only source of two-dimensional mappings of music. The next chapter examines music notation in general and the contemporary graphic score in particular, with an focus on how musical parameters are represented, layed out, and parameterized.

Chapter 4

Contemporary Graphic Scores: In-Depth Review

4.1 Introduction

The graphic score is an under-represented resource with regards to mapping. Although authors have collected examples of interesting graphic scores (Cage & Knowles' *Notations* [23]) and discussed notation as a practice in exacting detail (Read's *Music Notation* [24]), none of them have approached the material from an interaction design or music technology perspective.

This chapter will present a review of graphic scores viewed as mappings for musical control systems. Sixteen scores are examined, as seen in Table 4.1, ranging from 1950 to the present day, and including works by Morton Feldman, John Cage, and Anthony Braxton. Highly improvisational scores such as Earle Brown's *December '52* are also discussed. The parameters under discussion are pitch, time/rhythm, volume, timbre, and articulation. This chapter also provides a summary of the mappings used in traditional and twentieth century music notation as a baseline.

These scores fit into two main categories. Some, such as *Projection #1* and *Drift*, are for performers to play, whereas others, such as *Williams Mix* and *Solitude*, are scores to be realized by electronics, tape edits, or other means. These two classes, on the one hand, are very different forms (some realization scores are only created after the original piece has been performed), and often look very different.

Table 4.1: List of Composers and Scores

Composer	Piece	Year of Composition
Morton Feldman	*Projection #1*	1950
John Cage	*Williams Mix*	1953
John Cage	*59 1/2 Seconds For A String Player*	1953
Karlheinz Stockhausen	*Studie II*	1954
Karlheinz Stockhausen	*Kontakte*	1958
Gyorgy Ligeti	*Piece Electronique No. 3*	1959
Christian Wolff	*For Pianist*	1959
Sylvano Bussoti	*Siciliano*	1962
James Tenney	*Beast*	1971
Anthony Braxton	*Composition #76*	1977
Wendy Reid	*Tree Piece #8*	1985
Hans-Christoph Steiner	*Solitude*	2001
Steven Roden	*Pavilion Scores*	2005
Andrea Valle	*16 Nodi*	2006
Halim El-Dabh	*Canine Wisdom*	2007
Douglas Wadle	*Drift*	2010

On the other hand, as this chapter will show, both types of score use similar mapping techniques. In addition to the analysis of each score, this chapter provides summaries of mapping trends in graphic scores (pitch mapped vertically vs. pitch mapped horizontally, for example), and list notable or important exceptions to these trends.

4.2 Traditional Notation

Calling the current system of notation used in western classical music 'traditional' is, of course, a misnomer. Western notation has been evolving since before the first neumes were written down for sacred chants in the ninth and tenth centuries. Indeed, it would take until the twelfth century for Guido of Arezzo to systemize a four-line staff, and until the seventeenth century for a five-line staff with modern noteheads to become standard [24].

The five-line staff notation has sustained itself for several hundred years, however, and has remained the dominant notational paradigm, despite the drastically new world of twentieth century music. This section discusses how this vitally important paradigm maps pitch, time / rhythm, volume, articulation, and timbre.

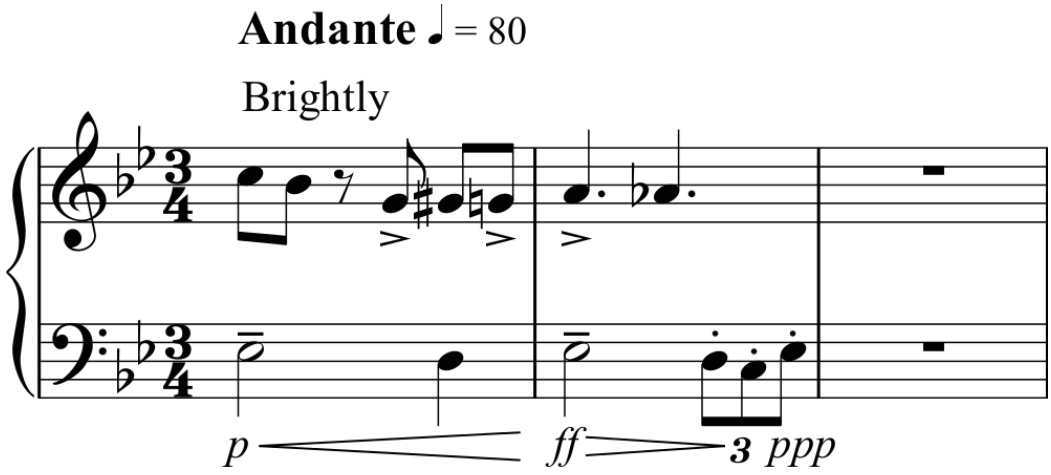


Fig. 4.1: A trivial example of traditional music notation, by the author.

4.2.1 Pitch

Pitch is delineated vertically: each line and space of the staff represents a note: G, A, B, C, D, E, F, G, A in the bass clef; and E, F, G, A, B, C, D, E, F in the treble. Interestingly, access to sharps and flats is done by adding symbols in front of the note in question [24]. Thus, taking key signature into account, notes are only mapped vertically within a diatonic scale: a chromatic scale, or any chromatic additions to a diatonic scale requires the addition of a symbolic mapping.

4.2.2 Rhythm & Time

Rhythm is delineated horizontally, but does not move at a steady rate. Instead, the style of each notehead indicates the duration of each note in time, and thus the rate of the passage of time. To generalize, the more ink used on the notehead, the shorter the duration: a filled notehead with a stem and many flags happens very fast, whereas an unfilled notehead lasts very long. (A more complimentary comparison may be that the more detailed the note, the more detailed the rhythm and the timespan). Additionally, tuplets subdivisions are indicated with Arabic numerals on top or below groups of notes [24]. Thus, even though time moves from left to right, it is represented in an entirely symbolic manner.

4.2.3 Volume

Volume is notated by text or abbreviated text: *p*, *f*, *sfz*, *crescendo*, *decrescendo*, and hairpin symbols indicated short, exact crescendi and decrescendi [24]. Although the abbreviations are textual or symbolic (*ppppp*, for example, has no real translation), the hairpin symbols suggest a size-based mapping of volume: wider is louder, thinner is softer.

4.2.4 Timbre

Timbre is a notorious problem in western music. While contemporary music has various and myriad extended techniques, traditional notation must rest largely on text descriptions: 'darkly','brightly'. These descriptions invariably include parameters that are not timbral, and thus are very difficult to pin down.

4.2.5 Articulation

Articulations, such as stacatto, tenuto, marcato, and so on, are indicated by symbols placed above or below noteheads. These symbols are suggestive of the desired envelope: stacatto is indicated by a dot, the smallest possible sign, whereas the drawn-out tenuto is a line, and marcato is a triangle pointing towards the notehead [24]. Despite this connection, these and other articulations are symbolic representations, not an x/y representation of the envelope.

4.2.6 Summary

Traditional notation offers clear directionality only for pitch, which it maps vertically. All other parameters are mapped symbolically, though time does proceed from left to right and from top to bottom. Articulation, volume, and timbre are notated via symbolic or textual means.

4.3 Twentieth Century Notation

Over the course of the last century, music has changed dramatically, from the rhythmic innovations of Stravinsky and the atonal works of Schönberg, through the electronic works of Cage and Henry, and to the contemporary spectral works of Murail. The notation for

this music has remained, with the exception of graphic scores, remarkably similar to that of previous centuries. This section will enumerate some of the changes and additions, and classify the mapping strategies used by them.

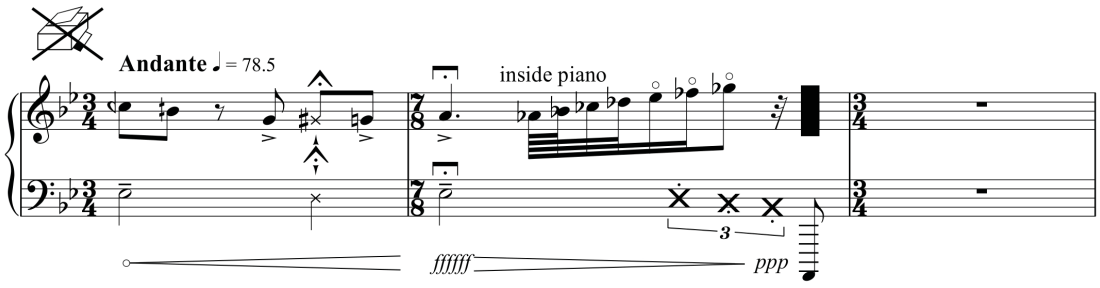


Fig. 4.2: A trivial example of twentieth century music notation, by the author.

4.3.1 Pitch

A key aspect of pitch notation in the twentieth century is the concern with finer gradations of pitch than traditional staff notation can easily deal with. An example of this is the various notation symbols used for microtonality. Quarter tones and sixth tones are represented by variations on traditional sharp, flat, and natural signs [24].

Other commonly used symbolic notations includes arrows indicating highest-possible or lowest-possible pitch. Similarly inexact vertical representations of pitch include describing heavy vibrato, or entire melodic lines with curved lines drawn over the staff [25].

Finally, cluster notation replaced literal clusters of noteheads with heavy black lines indicating the pitches to be played. This is an exact, vertical mapping of pitch, like traditional staff notation, simply using a single line rather than multiple noteheads [25].

4.3.2 Rhythm & Time

Many aspects of rhythmic notation in the twentieth century are simply the extrapolation of traditional notation to its logical extreme: 128th-note rhythms, constantly changing time signatures, and nested tuplets, to name a few.

An important change in the notation of time and rhythm, however, was actually a simplification. Proportional notation replaces the symbolic horizontal motion of time with an exact mapping of horizontal space to time passed. Each bar thus takes up the same

amount of space on the page, regardless of the notes contained within [25]. A variation on this is a method used by Earle Brown: extending each notehead horizontally to indicate how long to hold the note [24].

Other twentieth century temporal innovations include fermatas of varying length - the size of the symbol relates to its length, but the shape of the fermata varies well. Vertical arrows are also used for ritardandos and accelerandos, though the vertical axis is clearly still used for pitch. Another potential vertical mapping is that of feathering beams: slowly moving from one beam to many, on a diagonal. Although the vertical height increases with speed, this is simply a by-product of beaming conventions, and not a mapping of rhythm to a vertical axis [25].

Lastly, many twentieth century scores, Stravinsky's among them, begin to replace bars of rest with white space, erasing the staff entirely when an instrument is not playing. This furthers the idea that the amount of detail / ink directly relates to the speed and complexity of the music: pure silence is notated by no ink at all [24].

4.3.3 Volume

As with pitch and rhythm, dynamics in the twentieth century are concerned with extreme ranges and extreme detail. On the detail side of things, scores have been written with exact decibel markings, as well as with '+' or '-' signs beside traditional *p* or *f* markings. These symbolic / textual mappings are similar in method to traditional notation. The twentieth century also features many, many *ppppp* and *fffff* markings, though these extreme ranges were pioneered in the late 19th century [25].

More interestingly, a trend of mapping the size of the notehead to the volume becomes apparent in the middle of the century, across works by Berio, Browne, and Stockhausen [24]. In these cases, the stem and flag are removed from the notehead, and the size of the ellipse or circle is used to determine volume. These are not exact notations: a 5 mm circle does not equal mezzopiano, but two different 5 mm circles should represent the same volume during a performance of the piece.

4.3.4 Timbre

The majority of timbral indications are symbolic, and are applied either to the notehead or to the stem of the note. While the symbols attached to the stem are essentially arbitrary,

the modification of the notehead has some overall level of consistency to it. Specifically, as the notehead moves away from being circular, the timbre produced moves away from pure pitch and towards noise [25]. Some examples include the removal of noteheads to indicate sprechstimme, 'X' noteheads in percussion scores, triangle noteheads to indicate more air and less pitch in flute scores and so on [24]. This is clearly a high level view, rather than an exact mapping, but it suggests a relation between the 'perfection' of the circular notehead and the 'imperfect' triangular or square notehead.

4.3.5 Articulation

Assorted articulations and techniques of the twentieth century are too varied to list here - string techniques alone could fill a document ten times this size. However, as in traditional notation, articulations are indicated by a series of symbols and text abbreviations, with no strong trends to speak of [24].

4.3.6 Summary

Notation in the twentieth century has moved much less dramatically than the music that it notates. Despite many attempts at reworking the notation system entirely, the traditional staff-and-note model, and the mappings associated with it, have mostly remained the same. Even innovations like cluster notation and microtonal accidentals maintain the same mappings. Key differences, however, include note size notation for dynamics and proportional notation for time and rhythm. These represent new mappings, rather than variations on old themes - only time will tell if they will become part of the ever-evolving "standard" world of music notation.

4.4 Graphic Score Review

This section will examine the sixteen scores listed at the start of the chapter. Each composer and score is briefly introduced, and the score is analyzed in terms of how it notates and maps pitch, time/rhythm, volume, timbre, and articulation.

It will be noted that tablature-style scores, such as Lachenmann's *Gran Torso* and Aaron Cassiday's *Second String Quartet* have not been included. These scores are wonderful examples of the detail and precision of contemporary music, but their notation is of the

performer's actions, rather than of the music that the performer is to produce. As such, they are not pertinent to the discussion at hand.

Indeed, the lack of exactness in graphic notation needs to be discussed. The twentieth century also brought a trend for less and less control of performers. Nowhere is this more evident than in Earle Brown's *December '52*.

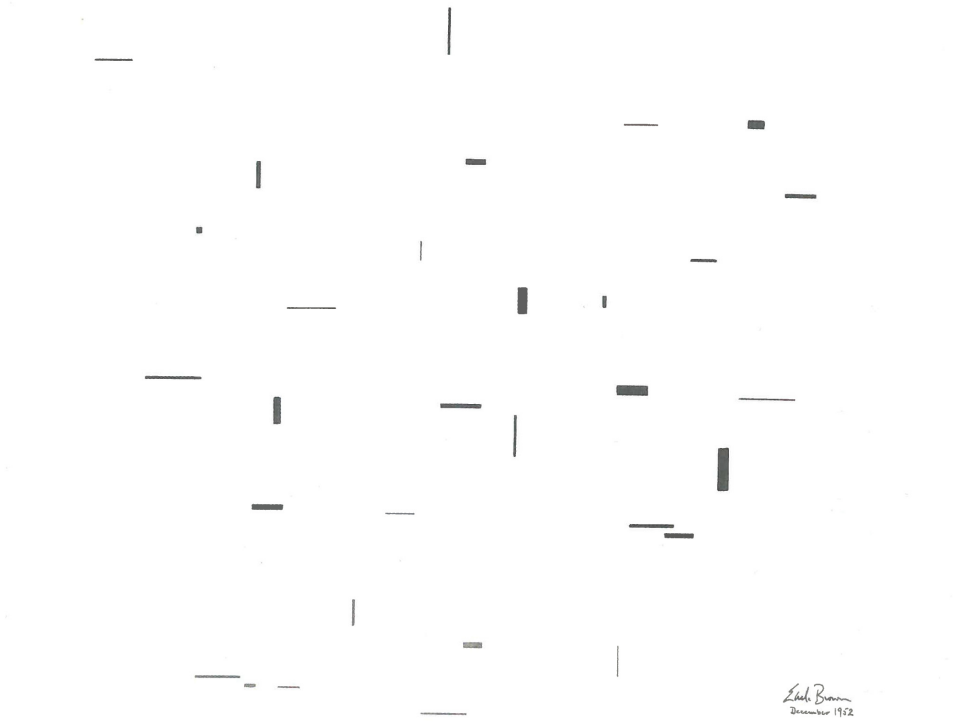


Fig. 4.3: The score for *December '52*, by Earle Brown. Used by permission of the Earle Brown Foundation, © 2006 Associated Music Publishers

December '52 (Figure 4.3) is a landmark piece for both contemporary music and graphic scores. It lacks almost any prescription by the composer for how the stark geometric figures are to be interpreted.

In the score [26], Brown states that line thickness "indicates the relative intensity and/or (where applicable instrumentally) cluster". No other prescriptions are given. When David Tudor later performed the piece, he worked out the exact pitches of each rectangle, according to a mapping of his own devising. Brown mentions, in a speech about *December '52* in 1970, that when he conducted the piece at Darmstadt in 1964, he defined the vertical height of the page as the entire range of each performer's instrument, and defined time as

moving from left to right...but adds that "continuity can be from any point to any other point". Line thickness again represented intensity. [27]

As the majority of mappings of *December '52* are mutable with the performance, it is not germane to the topic at hand. Rather, it is included here for the sake of historical completeness, and the recognition that many graphic scores are made to avoid the strict mapping of parameters.

Other such scores include John Cage's *Cartridge Music*, Cornelius Cardew's *Treatise*, Will Redman's *Book*, Joe Pignato's *Paprika King*, Anthony Braxton's *Composition #108B*, and many others [28] [29] [30]. *Treatise* (Figure 4.4) in particular is interesting, as Cardew, while eschewing any parameterization himself, exhorted the performer to work out their own self-consistent system, and perform the piece strictly to that system, rather than performing a totally free improvisation [31].

Although the above scores do not relate to the topic of this thesis, their influence cannot be overstated. Many of the below scores include inexactness in their graphic notation, and many of them ignore some parameters entirely.

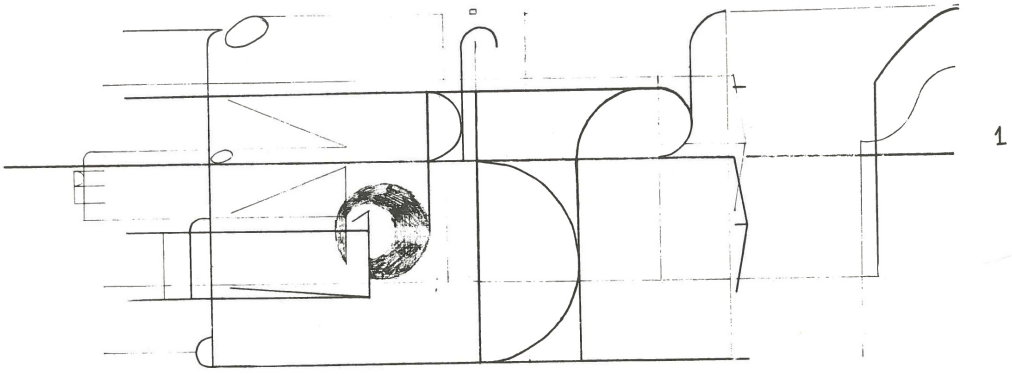


Fig. 4.4: A sketch for *Treatise*. Reprinted in Cage 1969 [23]

4.4.1 Morton Feldman - *Projection #1*

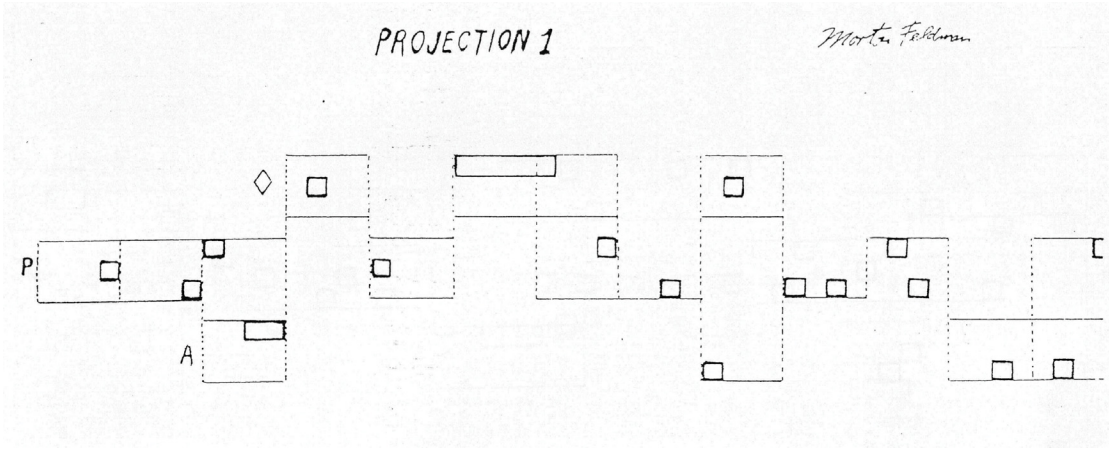


Fig. 4.5: An excerpt from *Projection #1*, by Morton Feldman (New York: C.F Peters Corp, 1962).

Feldman's *Projection* pieces, written in the 1950s, were some of the first experiments with graphic scores in contemporary composition. Feldman wrote the first of them, *Projection #1* (Figure 4.5), for solo 'cello, in 1950, at John Cage's apartment.

Pitch: Exact pitch is left to the performer, but relative pitch is indicated by the vertical position of the square in each box.

Time: Time moves proportionally from left to right. Each box is specified as being "potentially" four pulses at a tempo of "72 or thereabouts". Empty space in each box is silent.

Volume: Not specified by the notation.

Timbre: Each row of boxes indicates a different timbral technique: the top is harmonics, the middle, pizzicato, and the bottom, arco. This is both a symbolic mapping and a vertical one: the timbre moves away from a "pure" string tone, on the vertical axis.

Articulation: Not specified by the notation.

4.4.2 John Cage - *Williams Mix*

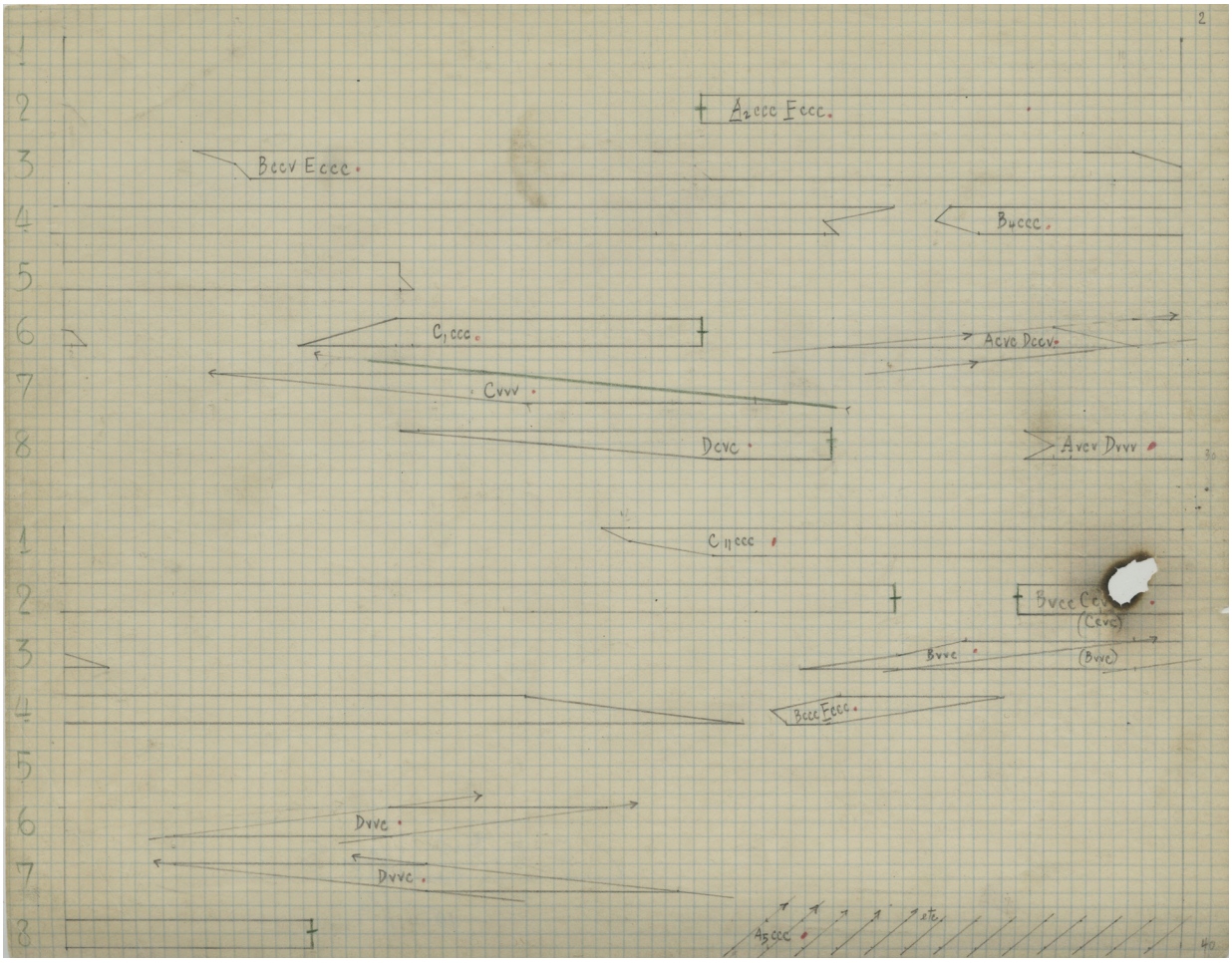


Fig. 4.6: An part from *Williams Mix*, by John Cage. © 1953 by Henmar Press, Inc. Used by permission of C.F. Peters Corporation. All rights reserved.

Cage's *Williams Mix* (Figure 4.6) is a short tape piece for eight tracks of tapes. Completed in 1953, the score is a pattern - it details where each tape should be spliced together. The source material was organized into six categories, and then spliced according to chance operations.

Pitch: Not specified by the notation. Although each section of each tape is detailed in the score, the pitch of the sounds on them are not.

Time: Horizontal and proportional, as each part of the tape is displayed at full size in the score.

Volume: Absolute volume is not specified, but the relative volume of each tape segment is shown by the vertical size of the segment at any given moment.

Timbre: Not specified by the notation. Although each section of each tape is detailed in the score, the nature of the sounds on them are not.

Articulation: Horizontal and proportional; as relative volume is defined by the total vertical size of each tape part, the attack and decay are defined by the change in vertical size over the horizontal axis.

4.4.3 John Cage - *59 1/2 Seconds For A String Player*

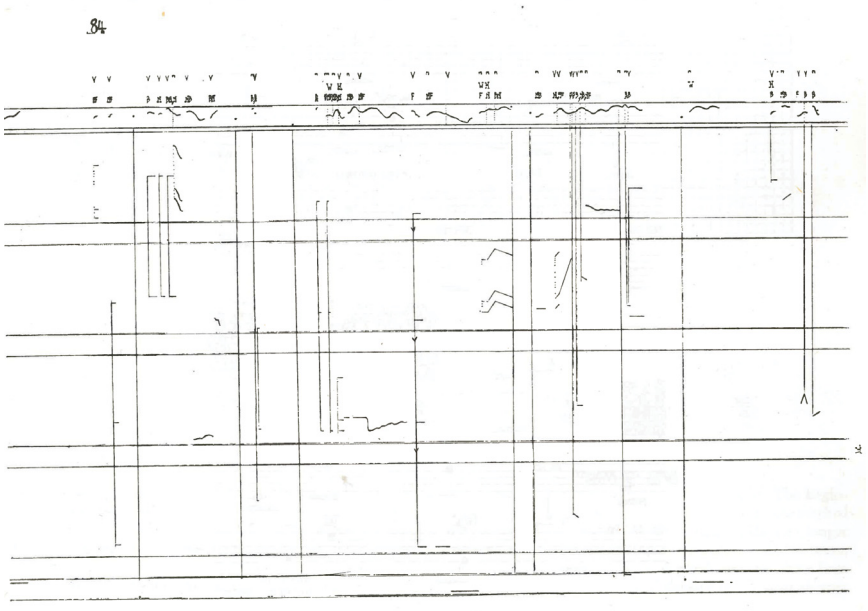


Fig. 4.7: An excerpt from *59 1/2 Seconds For A String Player*, by John Cage (New York: Henmar Press, 1960). Reprinted in Karkoschka 1972 [29]

One of Cage's earlier chance works, *59 1/2 Seconds* (Figure 4.7) was written in 1953. It was followed by *34'46.776" For a Pianist*, which was also composed using chance operations.

Pitch: Each of the four rows represents a string, with the small row under each indicating "other noises". Pitches are given as a gamut across one or more strings, with some relative pitch-lines also being indicated. Pitch is thus mapped vertically and proportionally, with "other noises" being treated below pitch.

Time: Horizontal, from left to right, proportional, but with traditional numeric tempo markings - the proportionality thus changes throughout the piece.

Volume: Not specified by the notation.

Timbre: Symbolic, with various letters indicating where on the instrument the performer is to play, and vertical, with a line indicating the amount of bow pressure.

Articulation: Indicated with traditional up/down bow symbols.

4.4.4 Karlheinz Stockhausen - *Studie II*

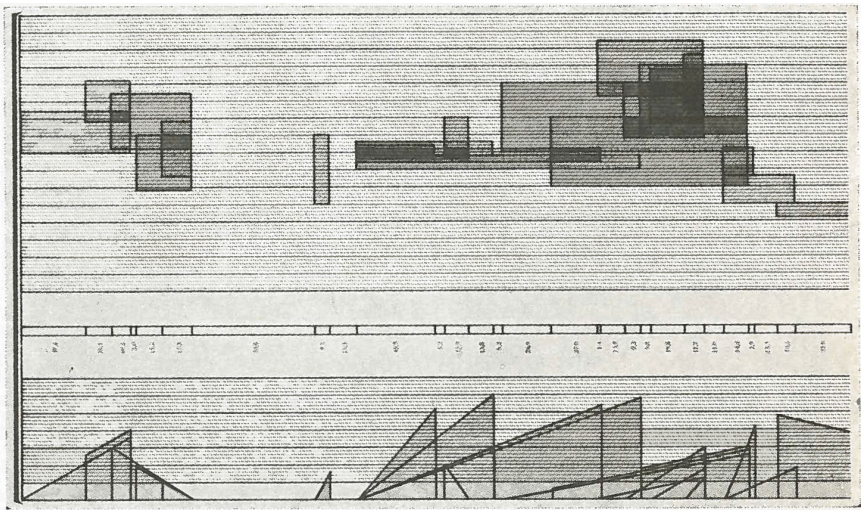


Fig. 4.8: An excerpt from *Studie II*, by Karlheinz Stockhausen (London: Universal Edition, 1956). Reprinted in Karkoschka 1972 [29]

Studie II (Figure 4.8) was realized by Stockhausen in 1954, in the West Germany Radio Electronic Music Studio in Cologne. It was one of two studies in using only pure sine waves as the compositional material - what would now be called additive synthesis.

Pitch: Vertical, but in proportional frequency rather than pitch. Each rectangle on the upper system represents the given range of frequency.

Time: Horizontal and proportional, with added numerals. The middle system displays exact timings for each rectangle on the upper system.

Volume: Vertical also, on the lower system. Each envelope maps directly to a pitch-rectangle on the upper system.

Timbre: Timbre is here entirely created by the layering of sine tones, and timbral density can roughly be mapped to saturation: darker sections on the upper system, caused by overlapping rectangles, will be more timbrally dense.

Articulation: Horizontal, and concomitant with the representation of volume. The mapping of volume to the vertical dimension allows the proportional, horizontal dimension to show how the volume envelope changes with time.

4.4.5 Gyorgy Ligeti - *Piece Electronique No. 3*

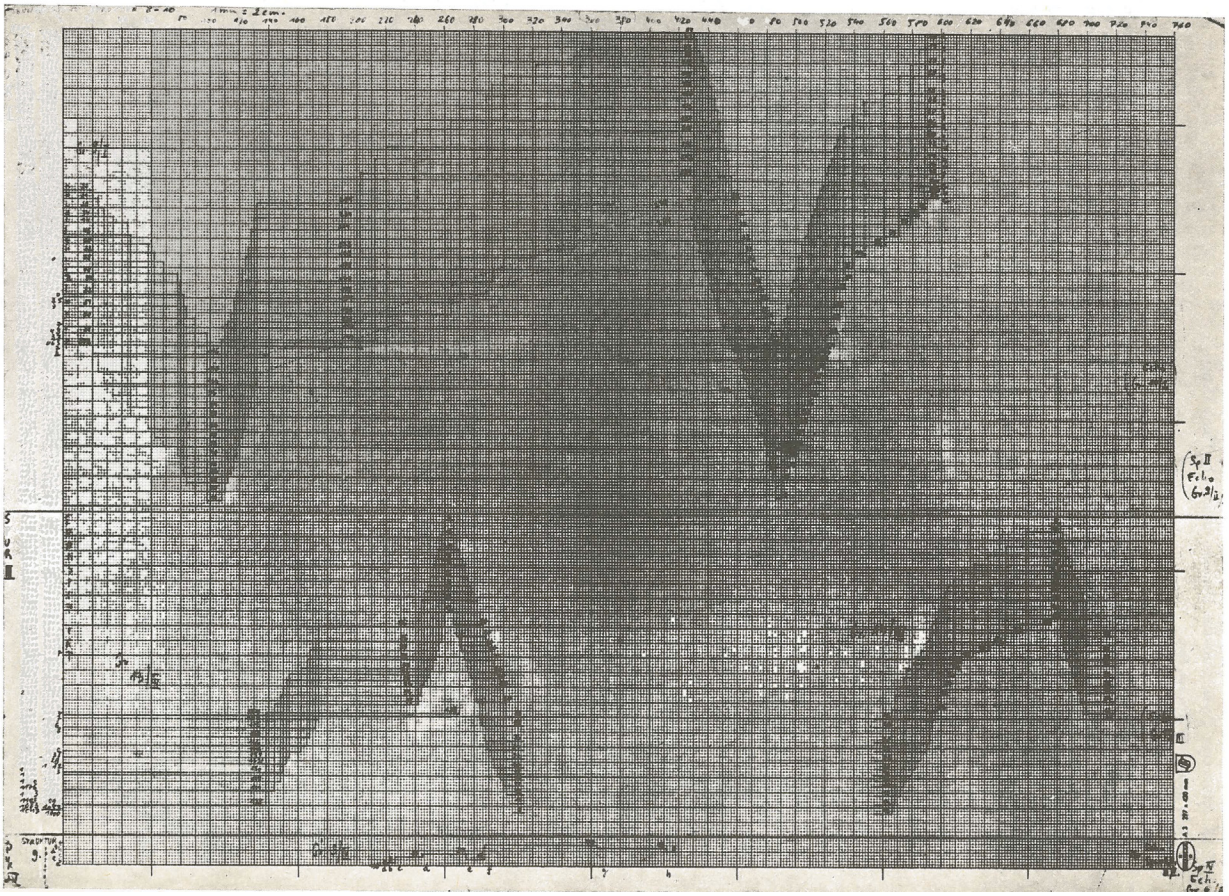


Fig. 4.9: An excerpt from *Piece Electronique No. 3*, by Gyorgy Ligeti (Berlin: Ahn & Simrock, 1959). Reprinted in Karkoschka 1972 [29]

Composed by Ligeti while at the West Germany Radio Electronic Music Studio in Cologne, *Piece Electronique* (Figure 4.9) was originally titled "Atmospheres", and was not technically possible with the equipment available. Kees Tazelaar and Johan van Kreij finally realized it in 1996 - long after Ligeti's 1961 *Atmospheres*, for traditional orchestra, had been completed. The work is for 4-track tape, two tracks of which are shown here.

Pitch: Vertical and proportional: the frequency numbers are visible on the left side of the score.

Time: Proportional and horizotnal, from left to right: one millimeter of the original score was equal to two centimeters of tape.

Volume: Not specified by the notation.

Timbre: Timbre is not specified by the notation, but it is implicity defined by stacking sine waves, thus suggesting a vertical mapping. Some effects are also written in ("Echo", for example).

Articulation: In contrast to Stockhausen's *Studie II*, the envelopes for each wave are not specified by the notation.

4.4.6 Karlheinz Stockhausen - *Kontakte*

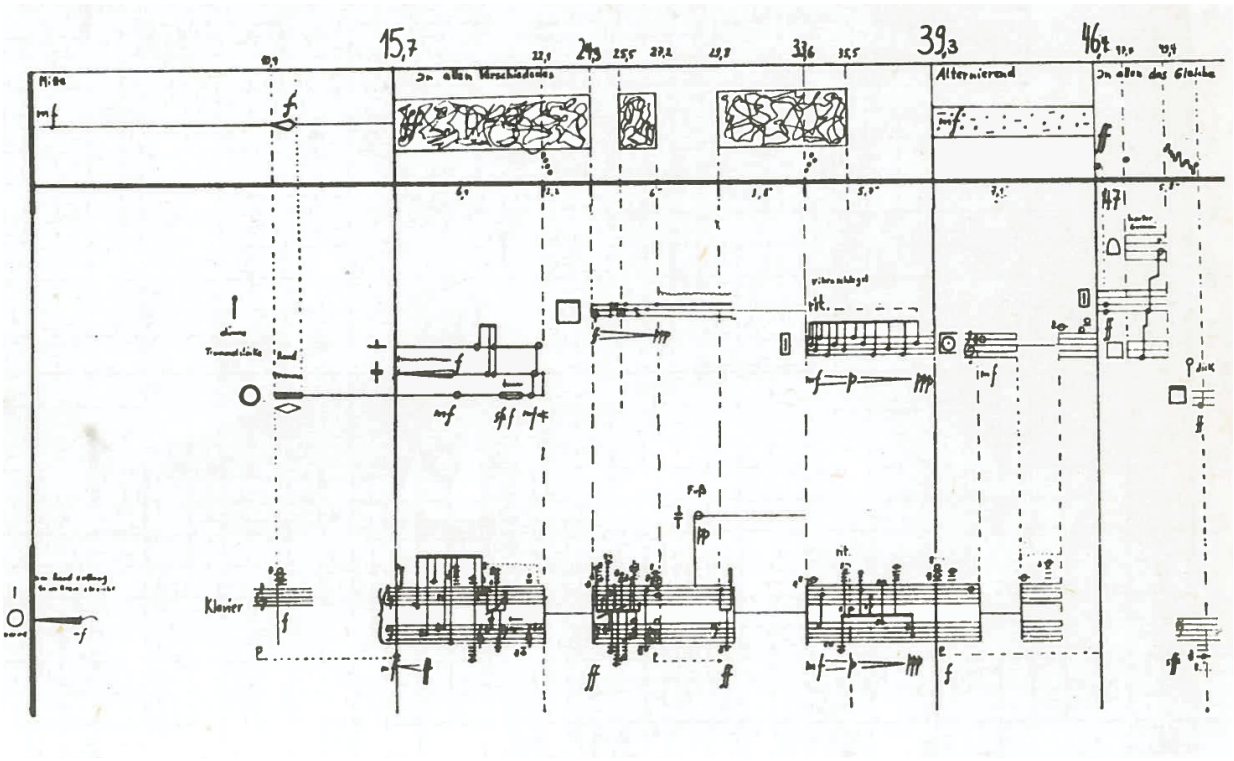


Fig. 4.10: An excerpt from *Kontakte*, by Karlheinz Stockhausen (London: Universal Edition, 1966). Reprinted in Karkoschka 1972 [29]

Stockhausen's *Kontakte* (Figure 4.10) is one of the early masterpieces of academic electronic music. It was created at the West Germany Radio Electronic Music Studio in Cologne between 1958 and 1960. Two versions of *Kontakte* exist. One is a tape piece purely for electronic sounds and the other adds piano and percussion to the tape part. This review uses the score for the piano / percussion version. The score is in two parts: the electronic score runs along the top of the page, with the piano and percussion parts below it in traditional notation.

Pitch: Pitch is mapped vertically and proportionally on the electronic score, and is integrated with timbre: due to the small size of the electronic score, pitch is not exact. The lines thus represent the approxmiate pitch and the approximate timbral character.

Time: Horizontal, proportional, with lengths in seconds. Stockhausen also adds the tape lengths (38.1 cm / 15" per second), though these are as an aid to rehearsal.

Volume: Traditional abbreviations, vertical size, and decibal numbers. Panning is indicated by treating each speaker as its own subsystem (numbered I to IV) of the electronic part. Panning can thus also be mapped to the vertical axis, with added. text descriptors, such as "Alternierend".

Timbre: As with pitch, timbre is inexactly notated on the electronic score. Stockhausen's figures attempt to show the approximate spectral envelope of the electronic part in time.

Articulation: As in *Studie II*, the horizontal axis of the electronic score shows the envelope of each sound.

4.4.7 Christian Wolff - *For Pianist*

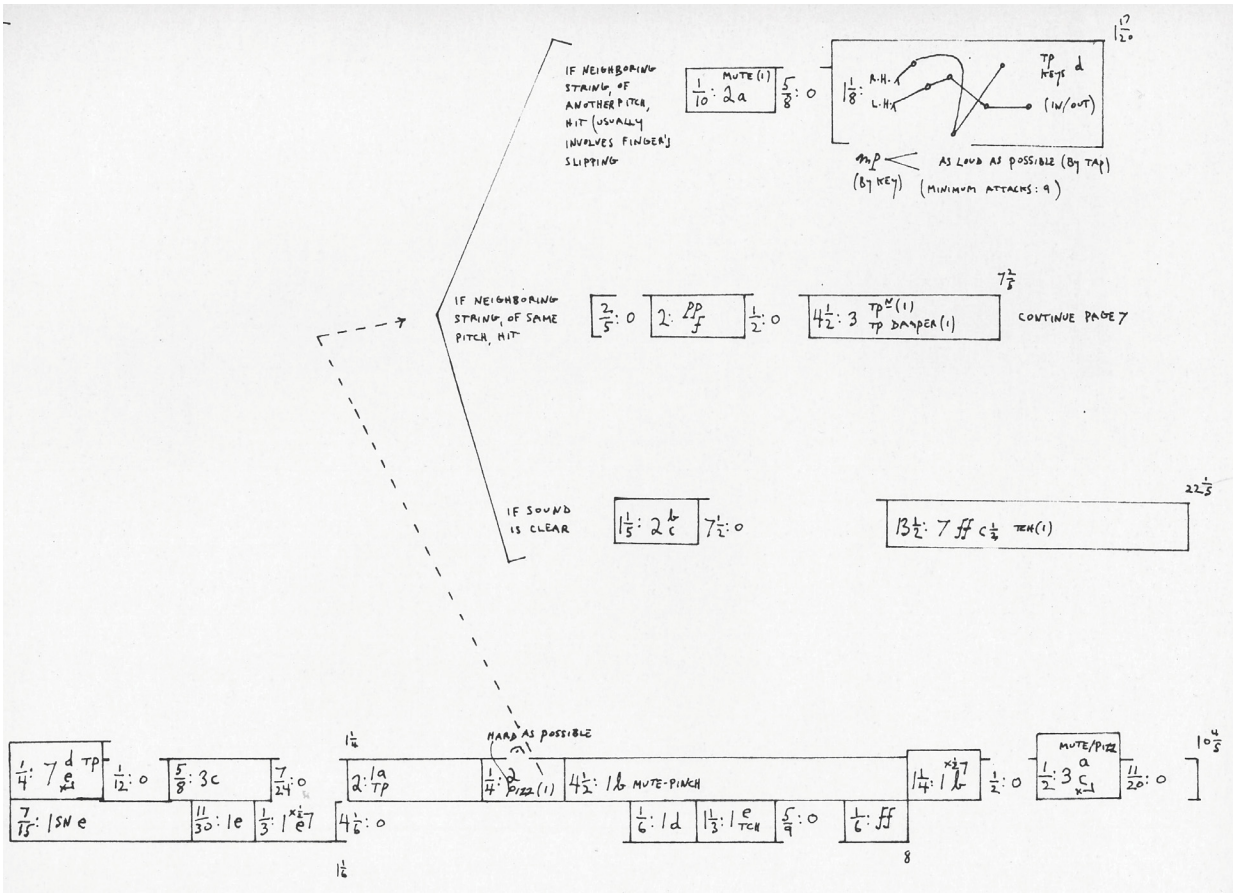


Fig. 4.11: An excerpt from *For Pianist*, by Christian Wolff. (New York: C.F Peters Corp, 1965)

Christian Wolff is an American composer, and is associated with Cage, Feldman, and the New York experimental scene of the 1950s and 1960s. Born in 1934 in France, he is largely self-taught. He was a professor at Dartmouth from 1971 to 1999. *For Pianist* (Figure 4.11) is a strictly organized improvisation, written in 1959.

Pitch: Wolff uses two systems to describe pitch. In each box, after the duration and the number of events, there is a letter. This letter indicates which of several gamuts of notes, defined in traditional notation in the preface, are to be used by the performer. This is a symbolic and vertical mapping. Wolff also uses minor variations on traditional *8va* notation to move the notes up or down, and to vary them by semitones. The second system of pitch notation refers to the location of the performer's hands on the piano. Circles indicate when notes should be played, the vertical axis indicates broad, proportional pitch, and the horizontal axis indicates moving inside the piano. Thus, this quasi-tablature notation also makes strong suggestions of timbre, as the performer's hands tap the body of the piano and strike the strings inside the piano.

Time: Each box / bounded area includes a number of seconds in which the events within it are to be performed, and a number indicating the number of events (though the rhythm within each box is left to the performer). They are roughly proportional, but by no means exactly so. Wolff also specifies that if multiple systems need to be played at once, time may be frozen at "tempo 0" in order to perform them all. More interestingly, many paths through the score are defined by waiting for musical events to occur: "when inaudible", or "when starts to produce harmonics". More interestingly still, other paths are defined by the performer's actions themselves. For example, an instruction to play a sound "as soft as possible" is given. If the performer makes the sound inaudible, she follows it with system A. If she plays it as desired, system B. If the sound is louder than as soft as possible, she moves to system C. In terms of mapping, this overrides the usual left-to-right nature of time!

Volume: The piece is generally "Free unless specified", with occasional traditional abbreviations.

Timbre: Traditional abbreviations / symbols, and horizontal/tablature notation.

Articulation: Traditional abbreviations / symbols, and horizontal/tablature notation.

4.4.8 Sylvano Bussotti - *Siciliano*

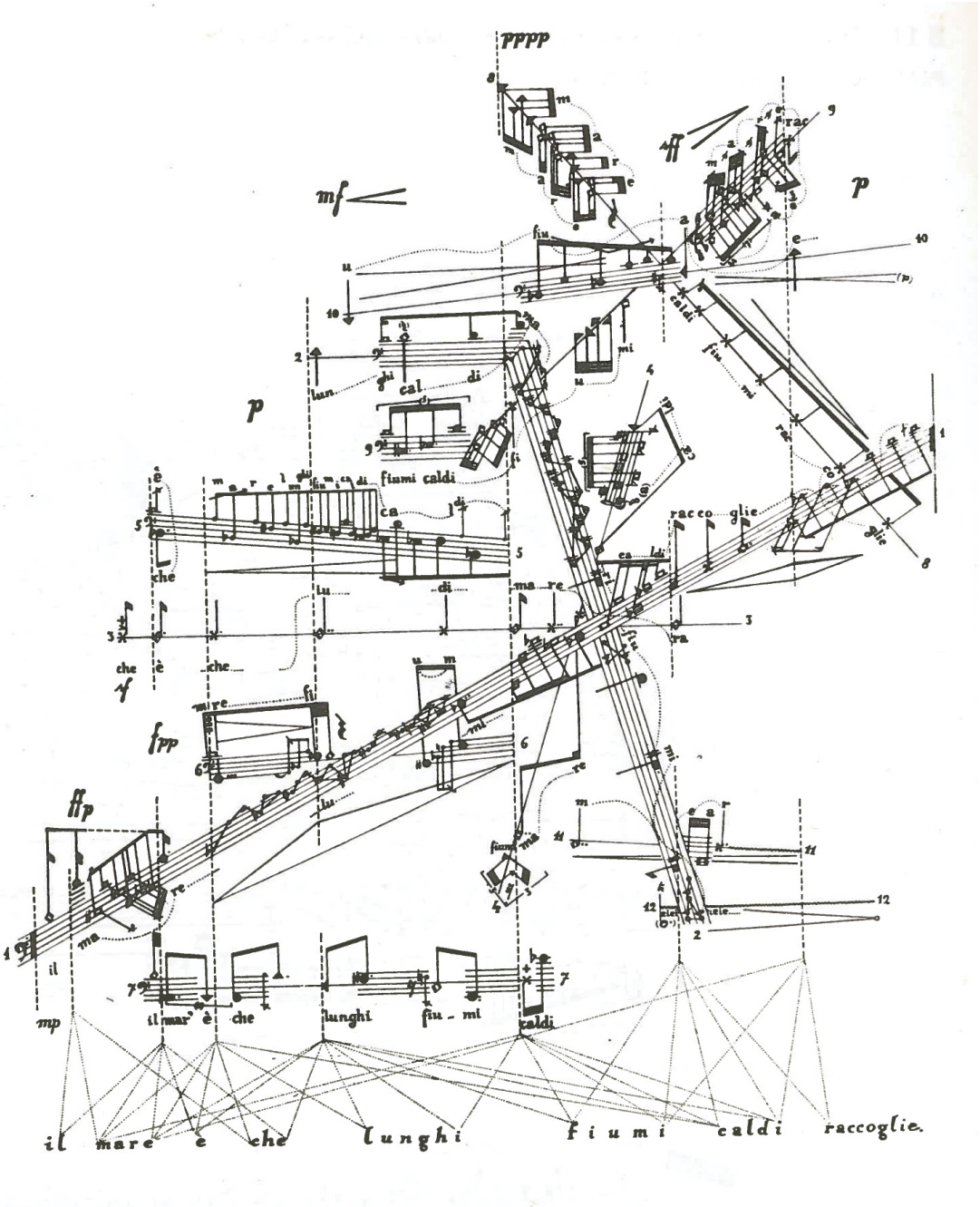


Fig. 4.12: An excerpt from *Siciliano*, Sylvano Bussotti (Florence: Aldo Bruzzichelli, 1962). Reprinted in Karkoschka 1972 [29]

Bussotti is an Italian avant-garde composer, painter, and graphic artist. Born in 1931, and influenced by Cage and Webern, he was a contemporary of Luciano Berio and studied with Lugi Dallapiccola at the Florence Conservatory. He taught at Fiesole in the 1980s, and has also been a director, actor, and singer. *Siciliano* (Figure 4.12), written in 1962, is indicative of his fusion of graphic and traditional notation.

Pitch: Traditional, vertical symbolic notation - except that the staves themselves are angled. Pitch is always 'vertical', relative to the angle of the staff.

Time: The score as a whole runs from left to right, with dotted vertical lines indicating points of synchronization. However, the angle of each staff indicates the rate of acceleration of the material on each staff. Tempo and rhythm are thus mapped vertically, even though time is horizontal: angles above 0 radians are accelerating, while angles below are deccelerating. Each staff also contains music written in traditional, symbolic rhythmic notation.

Volume: Traditional abbreviations.

Timbre: Traditional symbolic notation.

Articulation: Traditional symbolic notation.

4.4.9 James Tenney - *Beast*

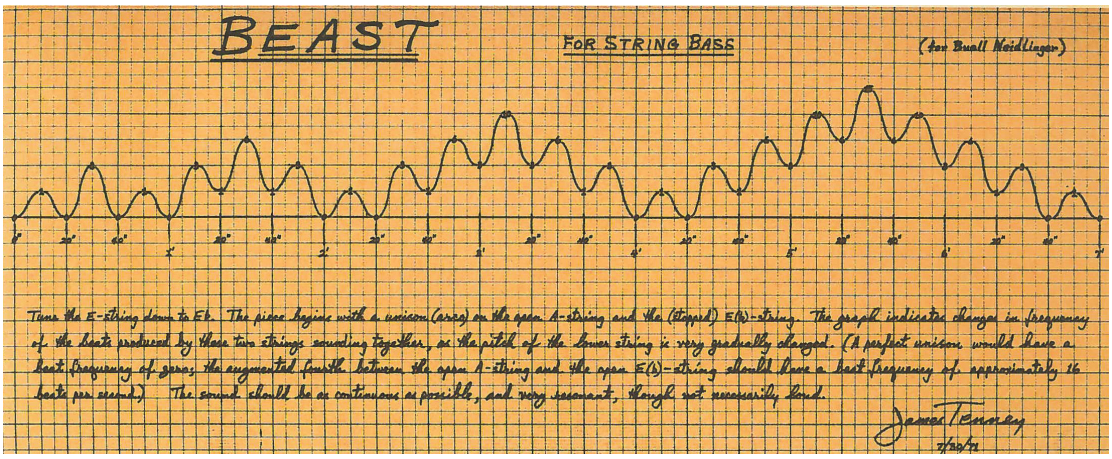


Fig. 4.13: The score for *Beast* from POSTAL PIECES, by James Tenney. Copyright Sonic Art Editions, 1984. Used by permission of Smith Publications, 54 Lent Road, Sharon, Vermont 05065. Reprinted in Sauer 2009 [28]

Beast (Figure 4.13) is a work for solo string bass, written by Tenney in 1971. It was dedicated to Buell Neidlinger. One of his *Postal Pieces*, the entire work fits on a single postcard.

Pitch/ Timbre: The piece is one long double stop on the string bass. The vertical line indicates the number of beats per second produced by the two strings sounding together. On the one hand, this clearly requires changing the pitch of one of the strings (Tenney specifies the lower string). On the other hand, as the beat frequency increases to a high point of 15, the beats become a timbral characteristic. Thus, both pitch and timbre can be said to be mapped vertically and proportionally. Specifically, timbral 'roughness' increases vertically.

Time: Time is mapped from left to right, in proportional chunks of twenty seconds each. The piece runs for a total of seven minutes.

Volume: Not specified by the notation. The text indicates that the piece should be "very resonant, thought not necessarily loud".

Articulation: Not detailed in the notation - Tenney's text say that the sound should be "continuous", suggesting a single attack over the entire piece.

4.4.10 Anthony Braxton - *Composition #76*

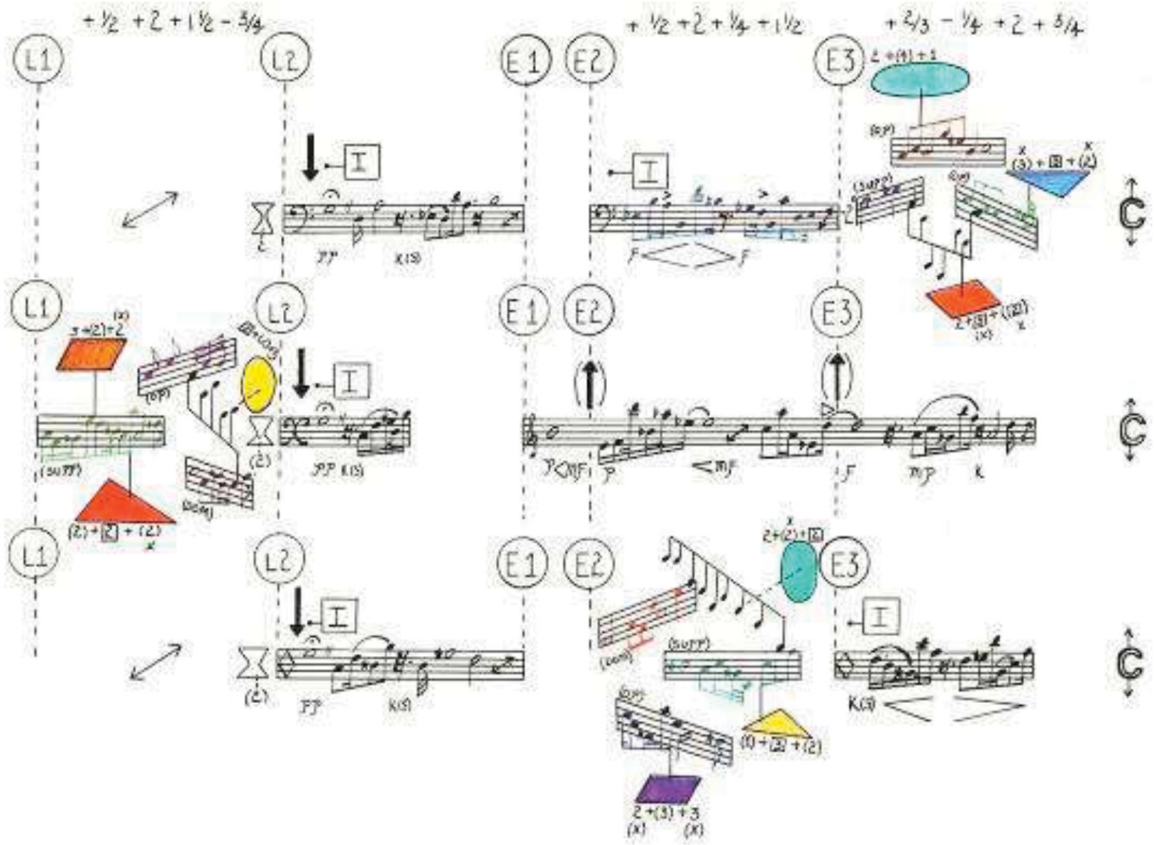


Fig. 4.14: An excerpt from *Composition #76*, by Anthony Braxton. Reprinted in Lock 2008 [30]

Anthony Braxton is a legendary post-everything composer - enumerating his various compositional trends and practices would take more pages than this thesis has. An American, Braxton recently retired from teaching at Wesleyan. He studied philosophy at Roosevelt University. *Compostion #76* (Figure 4.14), an "extended structure for three multi-instrumentalists" [32] was written in 1977, when Braxton was exploring colour and cultural dynamics - each colour is associated with an astrological sign, and music assigned to a given colour is to be played in that character.

Pitch: Traditional vertical, symbolic notation, with some indeterminate, clef-less vertical notation.

Time: Traditional left-to-right, symbolic notation, but with considerable freedom.

Volume: Traditional notation + colour notation - the more intense each colour, the louder it is played.

Timbre: Colour. Braxton maps colours on to various astrological signs and their respective cultural qualities: "Taurus, for example, is linked with green, and with feelings of calm and restraint". This is one of few colour mappings that attempts to be culturally general.

Articulation: Traditional symbolic notation, with added influence from the colour notation.

4.4.11 Wendy Reid - *Tree Piece #8*

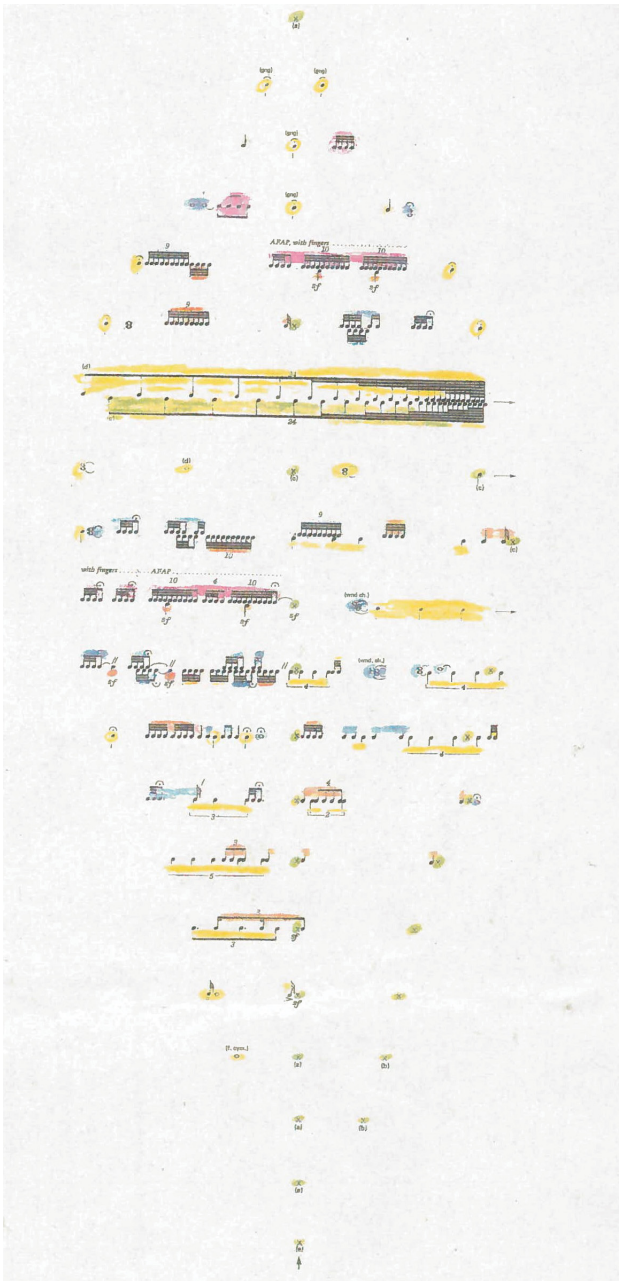


Fig. 4.15: The score for *Tree Piece # 8*, by Wendy Reid. Reprinted in Sauer 2009 [28]

Wendy Reid's *Tree Pieces* (Figure 4.15) are a continuing set of works "that attempt to reflect nature's manner of operations". An American, Reid herself studied at Mills, USC, and CCRMA at Stanford, and now teaches at Mills. Her teaches include Terry Riley and Robert Ashley.

Pitch: Not specified by the notation.

Time: The piece uses traditional symbolic noteheads, with time moving from left to right. The score, however, is read from bottom to top, rather than from top to bottom.

Volume: Not specified by the notation.

Timbre: By colour, with one colour for each instrument. There is, however, no strong mapping of hue to instrument (red being lower instruments, blue being higher, etc).

Articulation: Traditional symbols.

4.4.12 Hans-Christoph Steiner - *Solitude*

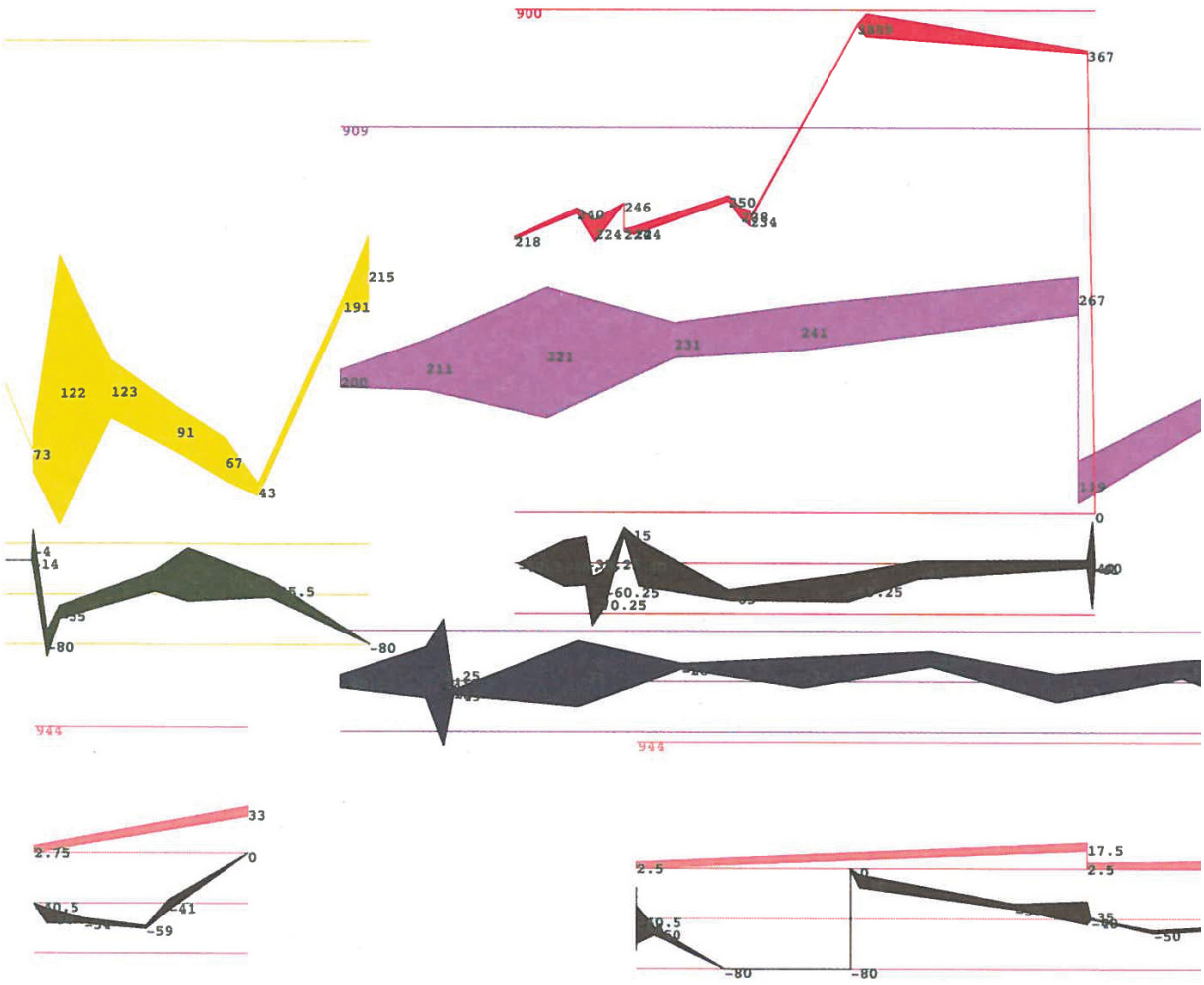


Fig. 4.16: An excerpt from *Solitude*, by Hans-Christoph Steiner. Reprinted in Sauer 2009 [28]

The German Hans-Christoph Steiner is a composer, programmer, and teacher, who studied at NYU. Steiner is self-taught in composition, and is currently based in New York City. His *Solitude* (Figure 4.16) was created entirely with Pure Data[1], the free and open source patching language. The patch follows the score, which was also created in Pure Data.

Like *Studie II*, there are two systems in the score. Each instance of a sound has a bright polygon, and then a darker polygon below it. The upper, bright polygon controls how the patch moves through the sound, whereas the lower polygon controls the amplitude and panning of the sound.

Pitch: Not specified by the notation. While many sounds have a pitch, their pitch is not notated.

Time: Time moves from left to right, proportionally.

Volume: Size. The height of the lower polygon, at any point in time, represents the volume of the sample at that point. This is not a mapping of vertical height, as stereo panning is mapped to the vertical centerpoint of the polygon. Rather, the vertical breadth of the polygon is indicative of the volume.

Timbre: Each color of the score represents a different sound, but there is no strong correlation between the timbre of each sound and a representation of color space. Rather, each shape controls the timbre: motion through each sound is controlled vertically, in a granular fashion. To quote Steiner: "The lowest point of the sample array is the beginning of the sample, the highest is the end, and the height of the array is how much and what part of the sample to play starting at that point in time". One can picture each sound, as originally recorded, as a diagonal line from lower-left to to top right.

Articulation: Like Stockhausen's *Studie II*, the horizontal mapping of the envelope of each sound is part and parcel of the volume mapping.

[1] http://puredata.info/

4.4.13 Steve Roden - *Pavilion Scores*

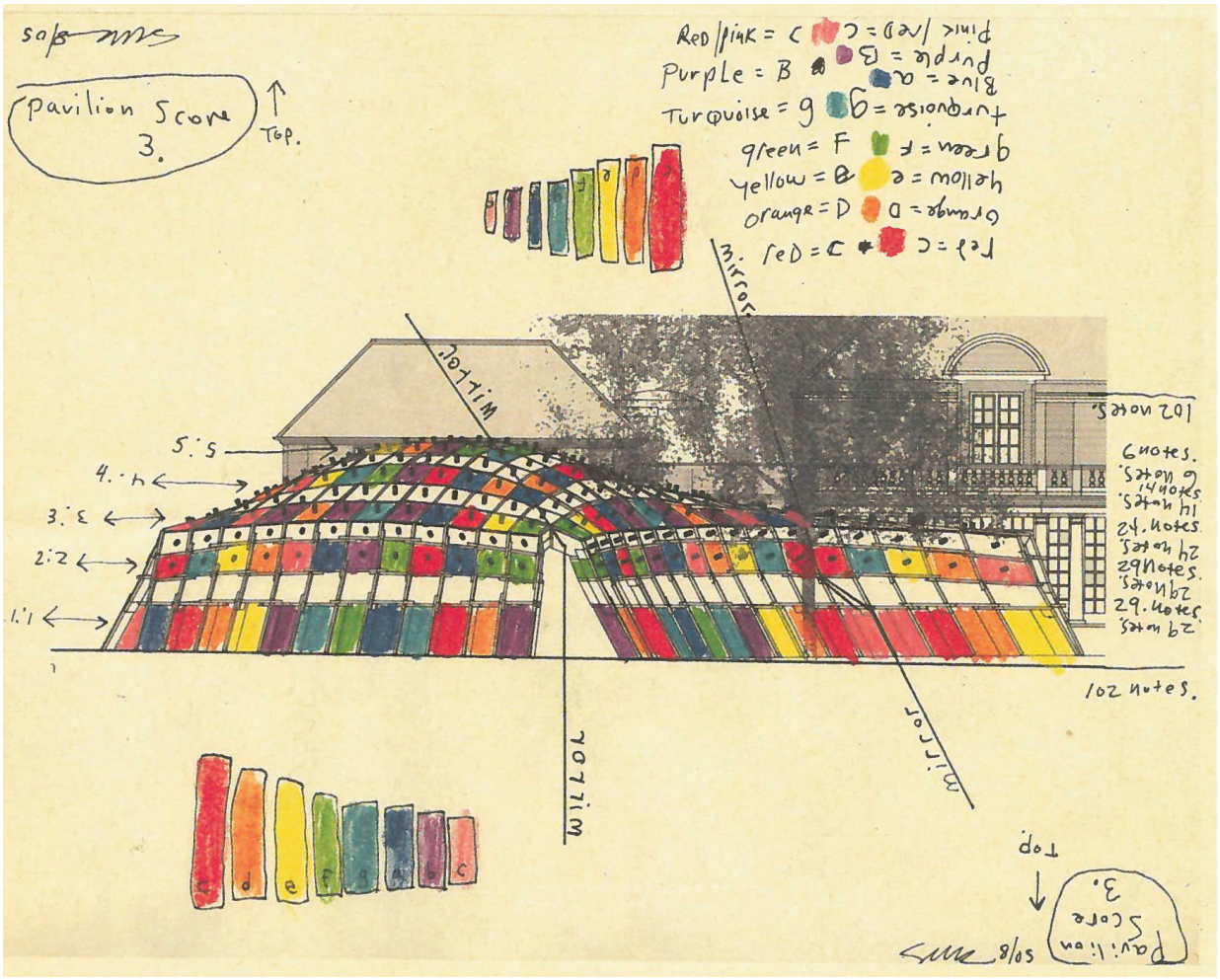


Fig. 4.17: A part from *Pavillion Scores*, by Steve Roden. Used by permission of Steve Roden, © 2005. Reprinted in Sauer 2009 [28]

Steven Roden is an American sound and visual artist who lives in Pasadena. He has taughed at many California institutions, including UCLA and UC Santa Cruz. He studied at the Art Center College of Design and the Otis College of Art & Design. *Pavillion Scores* (Figure 4.17) are part of a larger sonic work for the Serpentine Gallery[2], which included field recordings, found objects, contact mics, and glockenspiel performances by non-musicians. These scores, based on the structure of the building itself, were used by the glockenspiel players. Exact timing and repetition are up to the performer.

Pitch: Colour, matching the frequency of light. A low C is red, up to purple being a B, and a pale pink being a high C. This also suggests a brightness mapping: a pale orange could, in theory, be a high D, and so on.

Time: Generally left to right, but some score may also be played from top to bottom. Rosen also defines some squares of color as a single note and some as two note clusters. Time is thus mostly proportional, but occasionally symbolic.

Volume: Not specified by the notation.

Timbre: Not specified by the notation.

Articulation: Not specified by the notation.

[2]http://www.serpentinegalleries.org/

4.4.14 Andrea Valle - *16 Nodi*

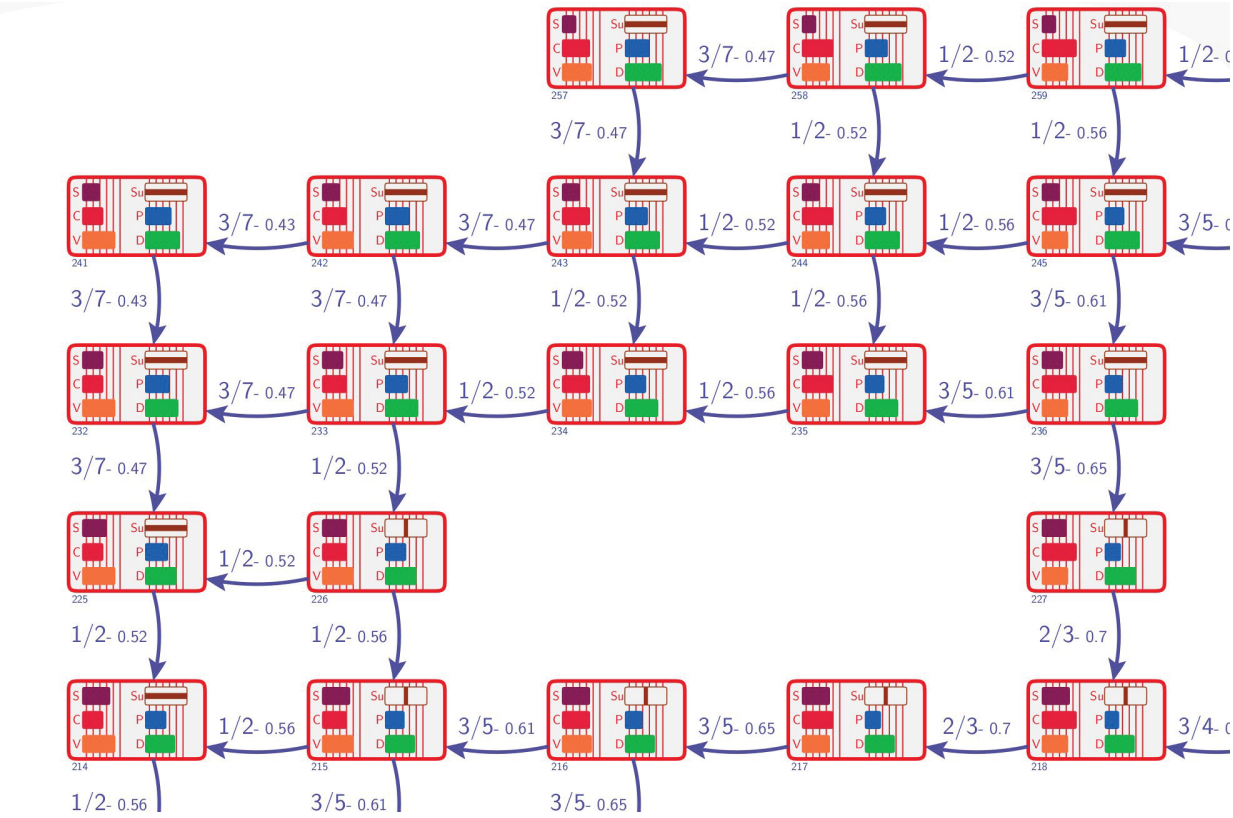


Fig. 4.18: An excerpt from *16 Nodi*, by Andrea Valle. Reprinted in Sauer 2009 [28]

An Italian, Andrea Valle is a researcher and musician at the University of Turin, and has studied composition with Azio Corghi. *16 Nodi* (Figure 4.18) was written in 2005/2006, and is for any (limited, as defined by the performer) set of sound objects.

The performer begins by selecting a node at the edge of the graph, and then moves to subsequent nodes. The performer may define their own plan for ending the piece, or stop when a dead end is reached.

Pitch: Pitch is defined by the Site, Calibre, and Variation parameters, all of which increase proportionally from left to right. Site indicates the register of the sound. Calibre indicates the bandwith of the sound - a sinewave has the lowest possible calibre and white noise the highest. Variation indicates how much these two parameters can be changed in any given node.

Time: Time moves omnidirectionally: the score is a directed graph, and the performer moves from node to node. Time spent on each node is described relative to a metrical pulse, in text, in both fractional and decimal values. It is interesting to note that the colour of the arrows changes with the direction and directionality of each arrow. Rhythm within each node is controlled by the Sustain (Su) and Profile (P) parameters. The Sustain parameter is defined symbolically: a long line indicates "energy is supplied continuously", a set of thin lines indicates "iterative" energy, and a single, short line indicates "impulsive". The Profile parameter defines "the way sound evolves in time", on a continuum between sounds that are pure impulses, on the left, and sounds that are infinite, on the right. This is thus a left-to-right mapping of both rhythmic length and articulative character.

Volume: Defined by the Dynamic parameter, increasing from left to right.

Timbre: Not specified by the notation.

Articulation: The Profile parameter defines "the way sound evolves in time", on a continuum between sounds that are pure impulses, on the left, and sounds that are infinite, on the right. This covers both local and medium-term articulation.

4.4.15 Halim El-Dabh - *Canine Wisdom*

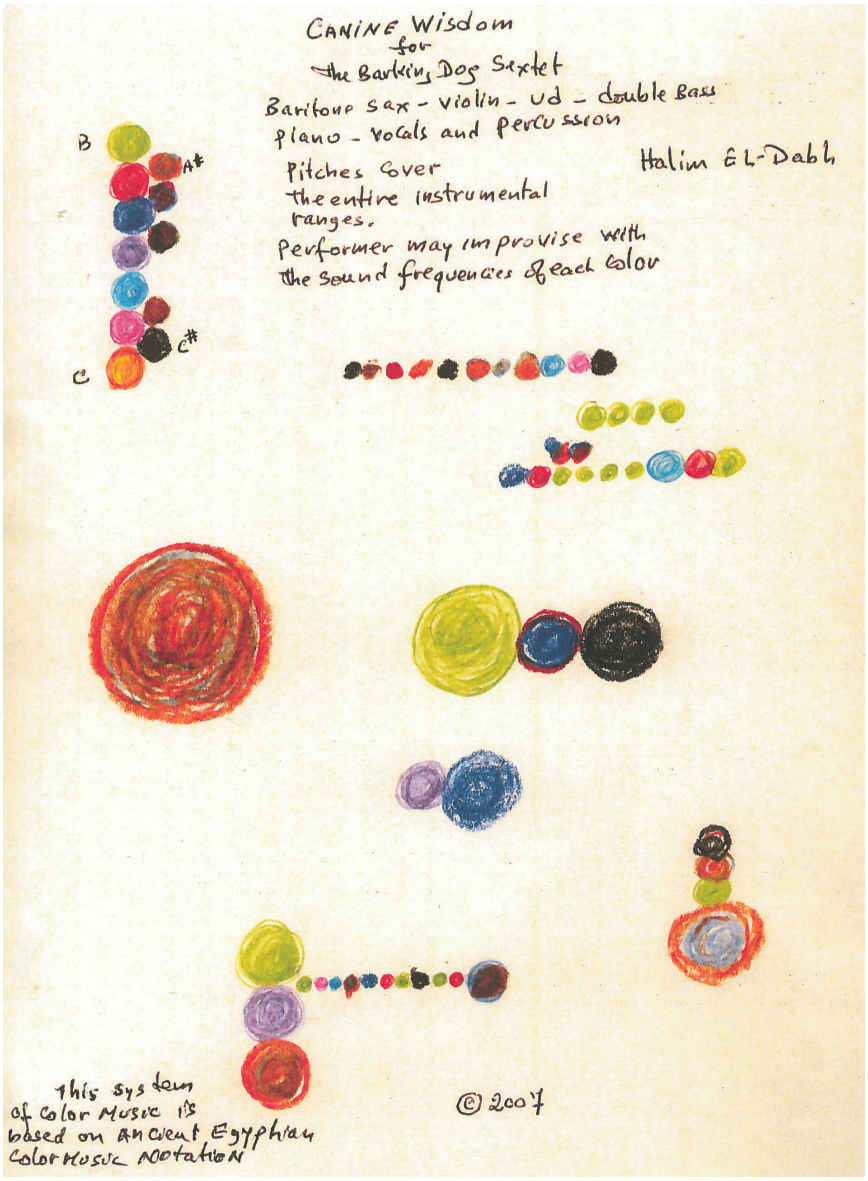


Fig. 4.19: The score for *Canine Wisdom*, by Halim El-Dabh. Used by permission of Halim El-Dabh Music LLC. Copyright by Halim El-Dabh Music LLC. May not be reproduced in any manner. Reprinted in Sauer 2009 [28]

Halim El-Dabh is an Egyptian-American composer and piano player, who is currently a professor emeritus at Kent State University. Born in 1921, El-Dabh studied with John Robb and Ernst Krenek at the University of New Mexico, among many others. *Canine Wisdom* (Figure 4.19) is a structured, meditative improvisation for baritone saxophone, violin, oud, double bass, piano, percussion, and vocals. It is a movement of the larger work, *The Dog Done Gone Deaf*, which was written for the Suoni per il Popolo festival in Montreal, Quebec, in 2007.

Pitch: Pitch is the only clearly defined parameter. In the top-left of the score, a piano keyboard of coloured circles is shown, with the piano keyboard displayed vertically. C is a bright orange, C# is black, D is magenta, and so on up to B, which is a light green. In the attached text in Notations 21, El-Dabh talks about his synesthesia and about ancient Egyptian notation, which also used a mapping of colour to pitch. This indicates that the colour, rather than the verticality of the 'keys' of the piano is the important mapping. The score also indicates that each pitch-circle covers the entire range of the instrument. However, the mapping of colour to pitched sound is not linear through hue space.

Time: Time is not specifically defined in the score, which consists of columns and rows of coloured circles. Given that El-Dabh displays the piano 'keys' vertically, it is reasonable to consider time moving horizontally and proportionally, and the stacks of circles to suggest harmony, but this is not specified. Rhythm, although not specifically detailed, is mapped to the size of each circle - a larger circle indicates a longer note than a smaller circle. Both this and the colour mappings are the same as in ancient Egyptian notation, which strongly inspired El-Dabh's notational method.

Volume: Not specified by the notation.

Timbre: Not specified by the notation.

Articulation: Not specified by the notation.

4.4.16 Douglas Wadle - *Drift*

Fig. 4.20: One of the parts from *Drift*, by Douglas Wadle. Used by permission of Douglas Wadle, © 2010.

Douglas Wadle is a Los Angeles based composer and trombonist, who has studied at Cal Arts and UCLA. An American, his teachers include James Tenney and Marc Sabat. *Drift* (Figure 4.20) is a partially improvised, microtonal piece for two tubas, written in 2010.

Each performer selects one of the lines, and then draws a horizontal line through it. This line represents the pitch of the other performer. To quote Wadle:

"Attempt, via step-wise motion along the micro-interval scales provided below, to follow the contour of your curve in relation to the sounding pitch of the other player, represented by the horizontal line, even as it shifts"

Pitch: Pitch is mapped vertically and proportionally, from bottom to top, but not with with any exact specification. Furthermore, the microtonality of the piece means that the total ambit of pitch will be very small. Although compositionally interesting, Wadle's use of a horizontal line to represent the other performer's pitch is not a repudiation of the vertical mapping of pitch. Indeed, Wadle uses a traditional staff to demonstrate the desired microtonality, again using a vertical mapping.

Time: Time moves proportionally from left to right on each page, though the total duration of the piece is up to the performers, and the piece may loop if the performers run out of material before the agreed-upon duration is reached.

Volume: Not specified by the notation.

Timbre: Not specified by the notation.

Articulation: Not specified by the notation.

4.5 Results

4.5.1 Summary of Mappings

A summary of the mappings across all pieces can be see in Table 4.2. It is first worth noting that many scores leave various parameters to the player's discretion. Volume, for example, is ignored by almost half of the scores. It also must be noted that some pieces include multiple mappings for a given parameter: thus, the numbers in each column will not add up to the sixteen scores reviewed.

Table 4.2: Mappings

Mapping	Pitch	Time	Volume	Timbre	Articulation
Left to Right - Symbolic	1	4	1	1	2
Left to Right - Proportional	0	11	0	0	4
Omnidirectional	0	1	0	0	0
Bottom to Top - Symbolic	3	1	0	1	0
Bottom to Top - Proportional	7	1	1	6	0
Top to Bottom - Proportional	0	1	0	0	0
Colour - Cultural	0	0	0	1	0
Colour - Rainbow	1	0	0	0	0
Colour - Synesthetic	1	0	0	3	0
Saturation - Other	0	0	0	1	0
Symbolic	1	0	0	1	2
Text / Abbreviations	0	1	4	1	3
Vertical Size - Other	0	0	3	0	0
Circular Size - Other	0	1	0	0	0

With that said, the dominant paradigms of vertical pitch and horizontal time remain. Many graphic scores move away from the symbolic nature of time and set time up in a constant, proportional way. Likewise, many scores treat pitch as a pure, ungraduated continuum, increasing from bottom to top. Furthermore, many scores, especially electronic ones, treat timbre vertically, in some cases combining pitch/timbre into one display. Timbre is also frequently described by colour or brightness, in keeping with the traditional idea of "tone-color"

Volume remains generally mapped using traditional abbreviations, though many scores also map volume to the vertical size of the graphical object in question. Finally, the graph-

ical, horizontal display of articulations is also very common: a modern user of synthesizers would recognize these descriptions of articulations as attack/decay/sustain/release envelopes.

4.5.2 Mapping Outliers

Graphic scores, however, display a wide range of creative mappings outside of the above trends. El-Dabh's *Canine Wisdom*, for example, uses a mapping of colour to pitch that is highly personal, as opposed to Roden's rainbow mapping of colour to pitch. Colour, in general, is used for pitch / timbre in differing ways between composers. Braxton's *Composition #76* applies colour to articulation and timbre based on cultural ideas such as green being "calm". This mapping, like Roden's rainbow of pitch, is probably more generalizable than El-Dabh's personal mapping. On the other, green is, in Western cultures, also the colour of envy and of money. Care must be taken when using colour in order to not point the user, performer, or listener in the wrong direction.

Colour is often used as a descriptor of timbre, whereas Stockhausen's *Studie II* darkens sections of overlapping sine waves, thus mapping darkness to timbral density. This is belied by the typical use of "bright" and "dark" by audio engineers to indicate the frequency content of a sound. Stockhausen's mapping is, rather, closer to Tenney's vertical mapping of beat frequency in *Beast*. Both mappings place timbral roughness on a sonic contiuum, as opposed to the typical performative continuum of, for example, *Cage's 59 1/2 Seconds*. Cage indicates what the performer should do, using a vertical mapping for bow pressure, where Stockhausen and Tenney indicate what sound should occur.

Time is in general an unexciting parameter: it moves from left to right, usually proportionally, and rhythm moves with it. Valle's *16 Nodi*, however, presents a omnidirectional view of time: the performer moves through each node of the circuit-diagram-like score in all directions, accordingly to the arrows leaving each node. This is reminiscent of Wolff's *For Pianist*. Although Wolff moves through time in a typical left-to-right manner, *For Pianist* contains optional leaps between sections that often move "backwards" in time.

Other creative mappings in time include the vertical mappings of Roden and Reid, and El-Dabh's mapping of the size of each note to time. El-Dabh's mapping would fit in well with general trends if the notes were flat, as in Brown's *December '52*, but El-Dabh uses circles. Thus, time takes on both a vertical and horizontal component, even as it moves

from left to right.

Bussotti's *Siciliano* also maps time in both directions. While absolute time moves from left to right, with synchronous events marked by dashed vertical lines, the angle at which a given system is displayed at changes the rate of acceleration while performing it. Bussotti's systems thus display a form of integral mapping: the vertical and horizontal aspects of each bar of music are connected.

This connection is by no means a standard event: much of the work of the early electronic pioneers in particular was about the separation of and exact control over every individual parameter of every sound. Steiner's *Solitude*, on the other hand, is a modern, electronic example of integrality. Each coloured shape represents a different sound, and the score moves, horizontally, through each sound based on the vertical position of the shape that represents each sound. Steiner's score also exists as a Pure Data patch, in which the user can change the shape of each patch of colour: this changes both the sound made, and the position of the sound in time.

4.6 Conclusion

This chapter has presented a review of mapping methods in graphic scores, as well as reviews of traditional and twentieth-century scoring techniques, and the mappings used by them.

Traditional notation, broadly speaking, moves time from left to right (and from top to bottom), in a symbolic manner. Pitch moves vertically, increasing from bottom to top. All other parameters are dealt with using various symbols and text abbreviations. As this notation moved into the twentieth century, time begins to move proportionally, pitch becomes both more exact (via painstaking microtonal notation) and less exact (via semi-graphical representation of clusters and vibrato). Notation for timbre and articulation likewise exploded into a complex morass of signs, symbols, and sketches, as composers worked to exactly specify each sound [24].

Graphic notation borrowed some of these ideas, most obviously the proportional representation of time. Pitch, in general, remained in a vertical mapping, though often in frequency instead of pitch, and often in an exactly proportional manner, rather than with symbolic sharps and flats.

New representations of volume, timbre, and articulation appeared, however. Volume is

often mapped to the vertical size or height of an object, with articulation of that volume defined horizontally. Timbre is often integrated with vertical mappings of pitch, or is mapped to the colour of an object. This makes perfect sense in electronic scores, but can be tricky in interpreted scores: one performer's idea of a "red note" may not be another performers (and neither of them might do what the composer had in mind).

Indeed, many graphic scores avoid specifying details for one or more parameters, and many are entirely improvisational. In the case of these scores, the more interesting discussion is of how they are interpreted, rather than how they are defined. This chapter has touched on this issue in its discussion of Brown's *December '52* and Cardew's *Treatise*

Some of the innovations of the last 100 years, such as proportional use of time, have become common notational practice. Other methods, such as the use of colour to represent timbre, have not. Read's overview of various historical notation systems [33] shows that composers have always attempted to improve and update how music is written and mapped. This chapter has shown that, in addition to wildly creative mappings and notational methods, some mappings of graphical space to musical sound remain constant.

Chapter 5

Two-Dimensional Mapping Abstractions

5.1 Introduction

The previous chapters have given an overview of mapping strategies across touchscreen applications and graphic scores. This chapter will take the mappings and metaphors described previously, and abstract the mapping choices away from their associated metaphors. It will then establish the underlying layout of controls that leads to the mapping choices in question.

For example, rather than considering the piano keyboard, consider a row of five buttons atop a row of seven buttons. This is the chromatic piano keyboard, abstracted from its geometry to its topology, as a particular triangle abstracts to a generalized triangle. These abstractions seek to reach their simplest visual form. For example, most piano metaphors use white and black keys, but it is not the color of the keys that indicate the mapping choice used. As will be seen below, it is the position of the buttons relative to each other, rather than their shape or colour, that indicate that a set of buttons is mapped as a piano would be.

This process is influenced by Tufte's infographic theories about the simplest possible representation of information [34]. Tufte claims that as more and more extraneous visuals are stripped out of a graph or infographic, it becomes easier and easier to understand the actual information being displayed. This often includes removing legends, ticks, and even

axis lines. Van Nort et al and Tymockzo have also examined mapping and geometry as applied to music. Van Nort et al focus on the use of geometry in controlling continuous synthesis variables [35], whereas Tymockzo discusses the discrete world of harmony and pitch space [36].

These abstractions are, of course, not the only way to map musical parameters to visual parameters. Ashley [37] has investigated non-Western mappings of pitch space, which are not well represented on the iOS app store or in this thesis. Indeed, the multitude of mapping methods, as reviewed by Hunt et al [10], belie the concept of finding the 'best' mapping for a given interface or control abstraction. These are, however, the abstractions that are most common on the iOS app store and in graphic scores. As will be seen, many of these layouts present only pitch as their variable, projecting pitch space on to some sort of two-dimensional geometry. Given the Western focus on pitch over rhythm [38], and the high number of Western metaphors (piano, guitar, etc) presented in the iOS app store, this is not surprising. It is, however, also related to a technical limitation of most touchscreen devices: the inability to detect changes in force with capacitive sensing [39]. Most acoustic instruments translate input force into output volume, and typically to a brighter output timbre. As sensing that force is impossible, most abstractions simply ignore volume and timbre, rather than attempting a clever mapping solution.

This chapter enumerates fourteen abstractions[1]. Each is described in terms of its layout and its relation to real-world instruments. Its defining characteristics and variable characteristics are listed, with examples. Finally, potential variations or under-utilized parameters are conjectured. The abstractions are listed below:

- Diatonic Row / Pentatonic Row - The canonical equal-tempered piano keyboard.
- Diatonic Row / Size - A diatonic scale, with varying size for each note, like a xylophone.
- Multiple Rows - One or more rows of buttons, of the same length: like the fingerboard of a violin or the fretboard of a guitar.
- Column / Size / Shape - A column of buttons of varying lengths, like a zither or dulcimer.

[1]Example screenshots for this section were sourced from the automatic review in Chapter 3. As the screenshots were downloaded independently, the names of some apps are not available

- Small Grid - A rectangular grid of buttons, typically smaller than 3x3. An ocarina or flute app would be a good example.
- Centered Row - A row of buttons that increases in pitch from the center, like a kalimba
- Diatonic Column - The canonical musical staff.
- Diatonic Column / Pentatonic Column - The canonical piano roll.
- Hexagonal / Triangular Grid - One of many varieties of Tonnetzen.
- Circle - A circle of buttons, like the circle of fifths or a steelpan.
- Orthogonal Grid - A rectangular grid of buttons, typically larger than 3x3. An MPC or a Tenori-On sequencer would be a good example.
- Rotary Encoder - A single rotating button.
- Timbral / Radial - A rough semi-circle of buttons, like a drum kit.
- Multiple Columns - One or more columns of buttons, like the faders on a mixer

5.2 Diatonic Row / Pentatonic Row

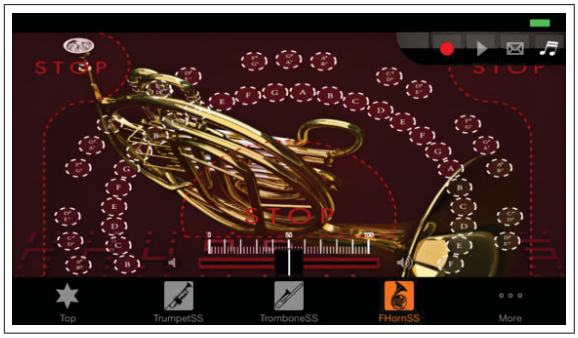
This is an abstraction of the traditional chromatic keyboard, used on pianos, harpsichords, synthesizers, organs, and on hundreds of touchscreen apps of various sorts. The canonical keyboard is two horizontal lines of buttons: seven below five, as in Figure 5.1. On iOS devices, keyboards simply play back various pitches: no timbre or volume controls are provided. The white keys are mapped, from left to right, to a diatonic scale,and the black keys to a pentatonic scale, filling out the chromatic scale across an octave. iOS apps that use this layout include pianos, organs, bells, and melodicas.

Fig. 5.1: A typical example of a Diatonic / Pentatonic Row layout (*Bell Piano*).

This layout is not defined by the shape of the buttons: two rows composed of almost any shapes will indicate the same mapping of pitch (Figure 5.2a). Nor does the rotation of the buttons (or the rows) matter: half-circle and spirals are also used (Figure 5.2b). Colour is often used to indicate 'black keys' vs. 'white keys', though the exact colour is not important (Figure 5.2c). The number of buttons and the relative location of the buttons matter: two rows of seven buttons are not a keyboard, nor is a row of five directly above a row of seven a keyboard. An abstracted keyboard can be seen in Figure 5.3.

Further variations based on this paradigm might include two rows of equal length indicating two opposing whole-tone scales. In general, two rows of X and Y buttons could indicate a equal-tempered (and possibly microtonal) scale with X+Y notes.

(a) Unique button shapes *(Racing Piano - Motorcycle Sound)*.



(b) Curved rows *(Brass instrumentSS)*.

(c) Unique colours *(Animal Piano)*.

Fig. 5.2: Three atypical Diatonic Row / Pentatonic Row layouts.

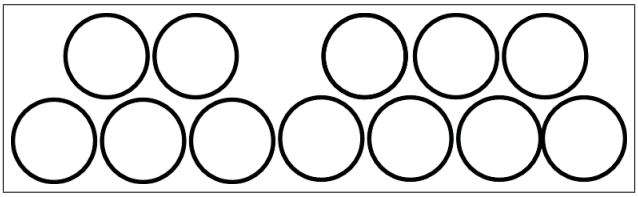


Fig. 5.3: The Diatonic Row / Pentatonic Row abstraction.

5.3 Diatonic Row / Size

This is an abstraction of a diatonic xylophone or diatonic keyboard. Like a xylophone, it consists of a single row of buttons, typically seven or eight, and typically decreasing in size from left to right. Figure 5.4 shows an example. On iOS devices, xylophones provide no timbre or volume control, and simply play back the appropriate pitch. The buttons are mapped to a diatonic scale, from left to right. If the buttons decrease in size, the decrease in size is not a precise mapping - a 2:1 increase in pitch does not equal at 2:1 decrease in visual size. iOS apps that use this layout include xylophones, harps, harmonicas, water glasses, and gamelans.

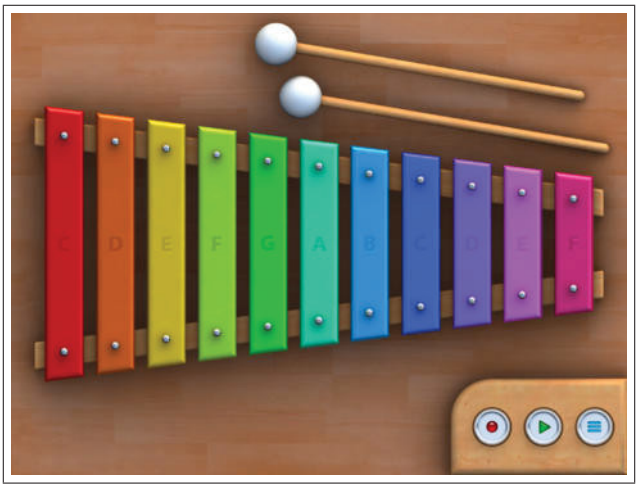
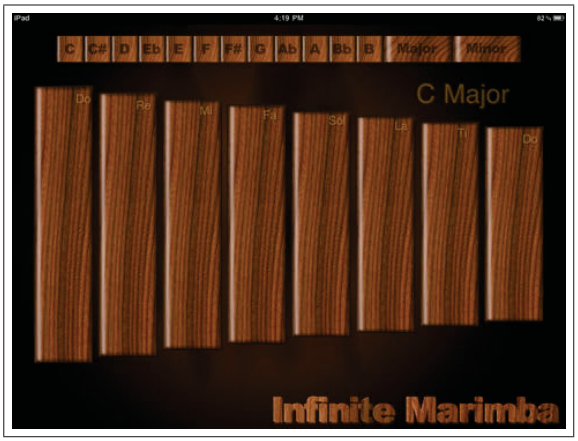


Fig. 5.4: A typical example of a Diatonic / Size layout *(Xylophone.)*.

As with the Diatonic Row / Pentatonic Row, this layout is not defined by the colour of the buttons (Figure 5.5a), nor by the shape of the buttons (Figure 5.5b). The number of buttons matters, as does their layout into some sort of row or line, though that line does not have to be orthogonal to the frame (Figure 5.5c). The size of buttons, when present, also matters: larger buttons are never higher pitches. An abstracted xylophone can be seen in Figure 5.6.

Continuations of this paradigm might change the number of buttons: five buttons indicating a pentatonic scale, eight indicating an octatonic,and so on. A row of X buttons could indicate an equal-tempered (though possibly microtonal scale) with X notes.



(a) Monochrome buttons *(Infinite Marimba)*.

(b) Unique button shapes *(Bottle Music Free)*.

(c) Angled row *(Let's Xylophone)*.

Fig. 5.5: Three atypical Diatonic Row / Size layouts.

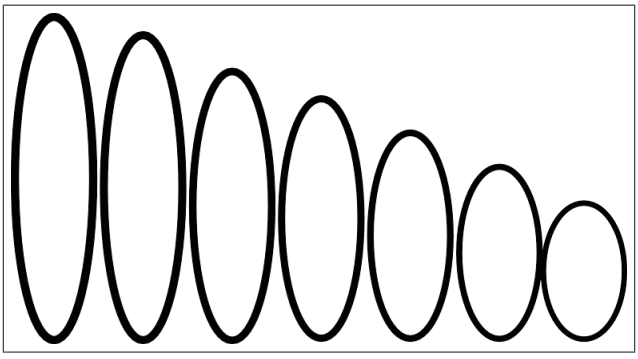


Fig. 5.6: The Diatonic Row / Size abstraction.

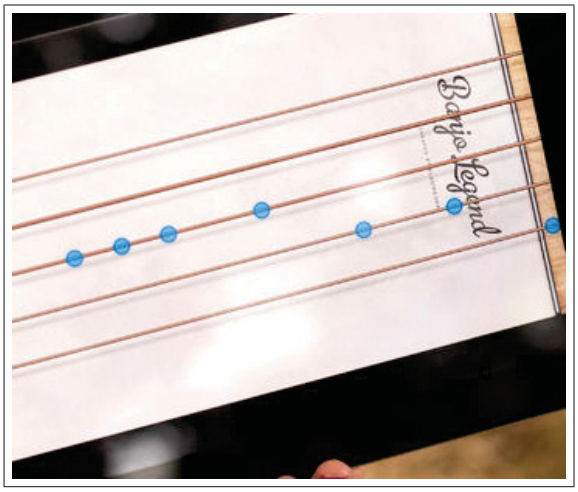
5.4 Multiple Rows

This abstraction covers both single-string and multiple string instruments, such as guitars and violins. Each row of buttons represents a string on, for example, a guitar fretboard. Each button in a row increases in pitch, chromatically, from left to right. Consecutive strings increase in pitch from bottom to top, usually by large intervals such as perfect fourths or perfect fifths. Figure 5.7 provides an example. On iOS, these layouts offer no control over timbre or volume - they simply play back the appropriate pitch (though some use these buttons to select pitch, triggering it with a separate touch). Other iOS apps that use this layout include 'cellos, ukeleles, lutes, and other stringed instruments.

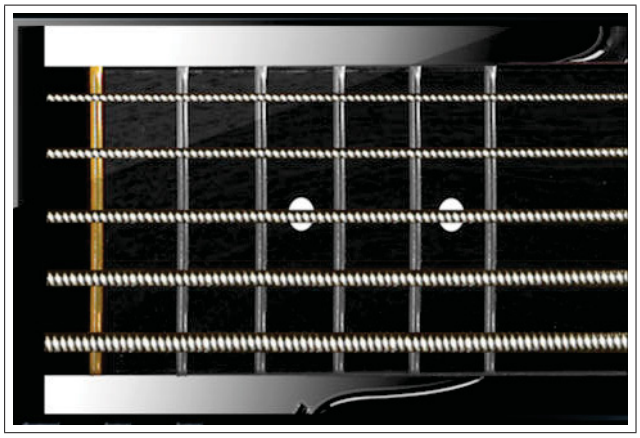
Fig. 5.7: A typical example of a Multiple Row layout *(App name not available).*

The defining characteristic is not the number of buttons per row, or the number of rows (Figure 5.8a). Indeed, the buttons are usually not visible under the visual metaphor of the guitar or violin (Figure 5.8b). Each button, however, is of uniform size. The main characteristics of this abstraction are that the number of buttons per row must be sufficiently large (four or more, in general), and the rows must be parallel to each other. An abstraction of a guitar can be seen in Figure 5.9.

As with the prior two abstractions, the number of buttons per row and the number of rows - the total number of pitches - is the main variable parameter. However, multiple rows allows pitch to be mapped with either steps per-row and leaps per column, or vice-versa. Multiple rows also suggest a difference in timbre with each row, which further impacts the choice of pitch mapping: some pitches may not be playable with certain timbres.



(a) Varying numbers of rows / buttons *(Banjo Legend)*.



(b) Hidden buttons *(App name not available)*.

Fig. 5.8: Two atypical Multiple Row layouts.

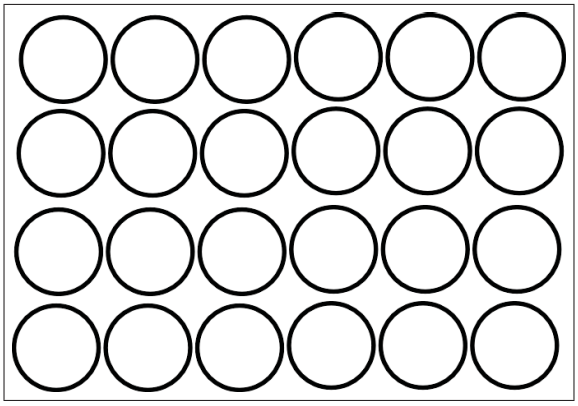


Fig. 5.9: The Multiple Rows abstraction.

5.5 Column / Size / Shape

This abstraction is similar to the Diatonic Row / Size abstraction turned ninety degrees. It includes both dulcimers and zithers, as well as similar vertical delineations of pitch. Each button in the column represents a string on dulcimer or zither, increasing in pitch in some sort of tuning, typically diatonically or chromatically. Figure 5.10 is an example, as seen on a dulcimer app. These layouts offer no control over timbre or volume, on iOS.

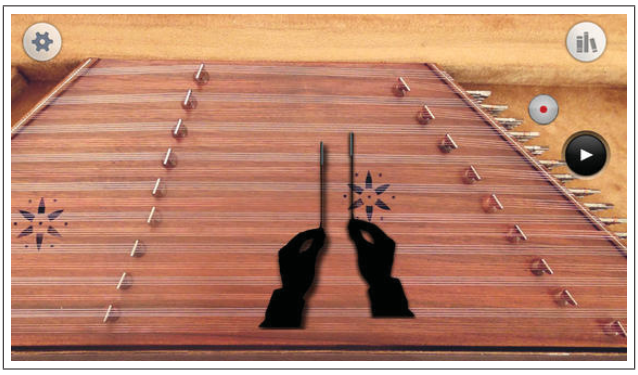
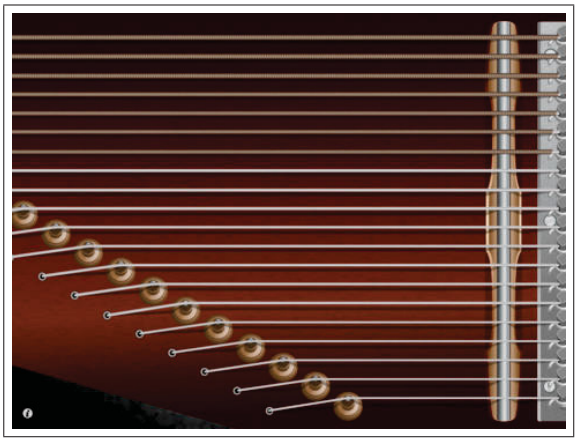


Fig. 5.10: A typical example of a Column / Size / Shape layout *(Santoor)*.

Interestingly, this abstraction can map pitch from bottom to top (Figure 5.10) or from top to bottom (Figure 5.11a), though the number of buttons is not a defining feature (compare Figure 5.11b and Figure 5.11a). The button shape is elongated horizontally, and the size of the buttons indicates the direction of the mapping of pitch. The colour of the buttons is extraneous to the abstraction (Figure 5.11b), as is the left-right alignment of the buttons (compare Figure 5.10 and Figure 5.11b). An abstraction can be seen in Figure 5.12.

Variations on this abstraction again rest primarily on different scales and mappings of pitch - either an equal temperament with a number of notes equal to the number of buttons, or a more specific scale, such as octatonic or hexatonic.



(a) Pitch increasing from top to bottom *(Kacapi)*.

(b) Varying colours *(Magic Zither)*.

Fig. 5.11: Two atypical Column / Size / Shape layouts.

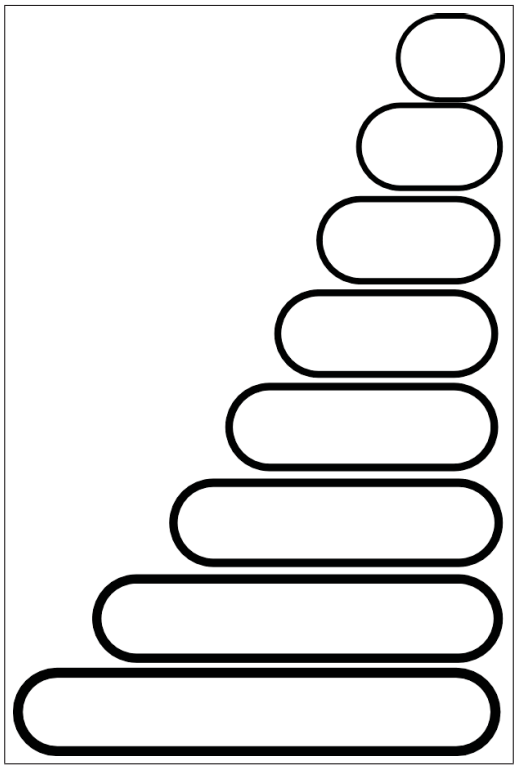


Fig. 5.12: The Column / Size / Shape abstraction.

5.6 Small Grid

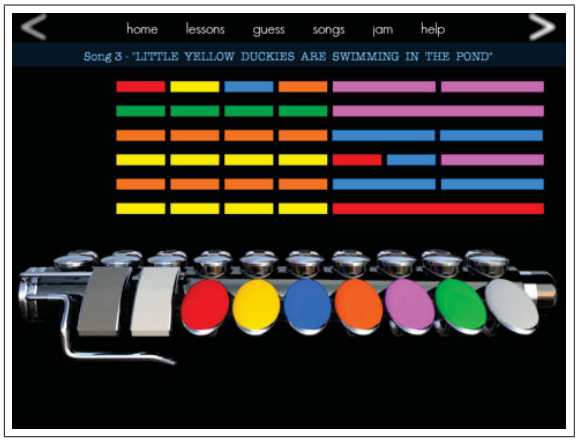
Representing various wind instruments (ocarina, flute, clarinet, etc) as displayed on touch-screens, a 'small grid' is defined as having fewer than ten buttons, and may have unequal dimensions: 1x6, 2x4, and 3x3 are all valid small grids. Unlike all other abstractions, pitch is often mapped additively: if pressing Button 1 plays a D and pressing Button 2 plays an F, pressing them both will play a G. The relationship between button location and increase in pitch is not consistent. This abstraction can also map pitch subtractively: the lowest pitch is triggered by pressing all the buttons. On iOS, these layouts offer no control over timbre or volume. Figure 5.13 is an example, from an ocarina app. Other iOS apps that use this layout include bagpipes, trumpets, and french horns.

Fig. 5.13: A typical example of a Small Grid layout *(Ocarina)*.

The main characteristic of this layout is the small number of buttons, and their arrangement into a grid, or something very close to a grid (Figure 5.14a). The colour of the buttons do not matter (Figure 5.14b), although the size and shape of the buttons are typically similar. An abstraction can be seen in Figure 5.15.

Potential variations to this abstraction could focus on different scales and mappings of pitch for different numbers and layouts of buttons. A higher number of buttons implies either a larger range of pitches or more detailed gradations of pitch.

(a) Informal grid *(Bagpipes)*.



(b) Unique button colours *(Magic Flute for Little Composers)*.

Fig. 5.14: Two atypical Small Grid layouts.

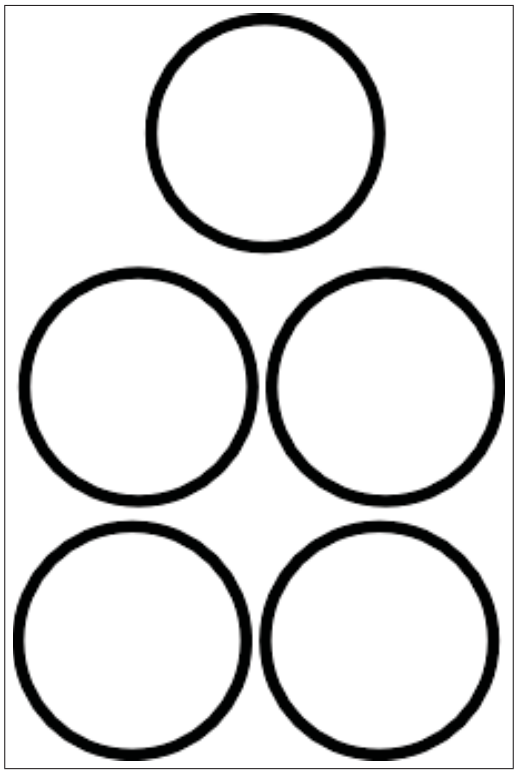


Fig. 5.15: The Small Grid abstraction.

5.7 Centered Row

This is an abstraction of a kalimba, mbira, or thumb piano. Much like the Diatonic Row / Size abstraction, this layout is a row of buttons of varying sizes, but with the largest buttons (representing the lowest pitches) in the center of the row, as seen in Figure 5.16. On iOS devices, kalimba apps simply play back pitches, without timbre or volume controls. Although tunings vary, the buttons increase in pitch out from the center, usually alternating intervals: a major second on the right, a major third on the left, a perfect fourth on the right, and so on. iOS apps that use this layout include kalimbas, sansas, and other variations.

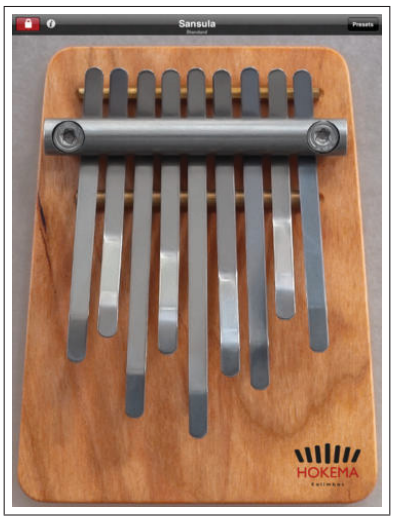


Fig. 5.16: A typical example of a Centered Row layout *(Sansula)*.

The two defining characteristics of this abstraction are the arrangement of buttons by size, with the largest/lowest buttons in the center and the long, thin shape of the buttons. The number of buttons can vary wildly, as can their colour (Figure 5.17a). Indeed, multiple rows of buttons are also possible (Figure 5.17b), with the higher row having smaller buttons, and thus a higher pitch. An abstraction can be seen in Figure 5.18.

Variations and expansions to this abstraction largely include varying tunings and varying numbers of buttons. A mapping that applies different timbres or volume envelopes to different rows would also be intriguing.

(a) Varying number of buttons and colours *(PercussionSS IA Vol.2)*.

(b) Multiple rows *(S4-Kalimba)*.

Fig. 5.17: Two atypical Centered Row layouts.

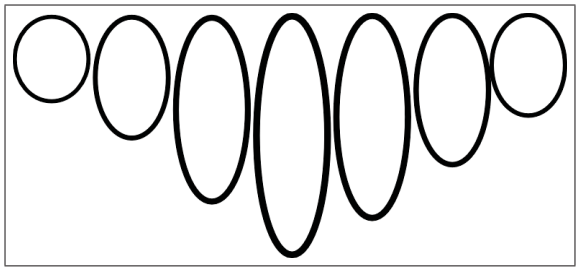


Fig. 5.18: The Centered Row abstraction.

5.8 Diatonic Column

Based on the five-line staff that dominates Western music, this abstraction maps pitch vertically, increasing by one note of a diatonic scale with each additional line or space. In iOS notation apps, timbre and volume are controlled using text or symbols, not the staff itself. Indeed, although notation apps give access to a chromatic scale through the use of accidentals, the staff (and this abstraction) do not: they are entirely diatonic. Time would move from left to right, were it to be involved. One can imagine a grid-like variation on this abstraction, as per the Orthogonal Grid, below. An example of a Diatonic Column can be seen in Figure 5.19

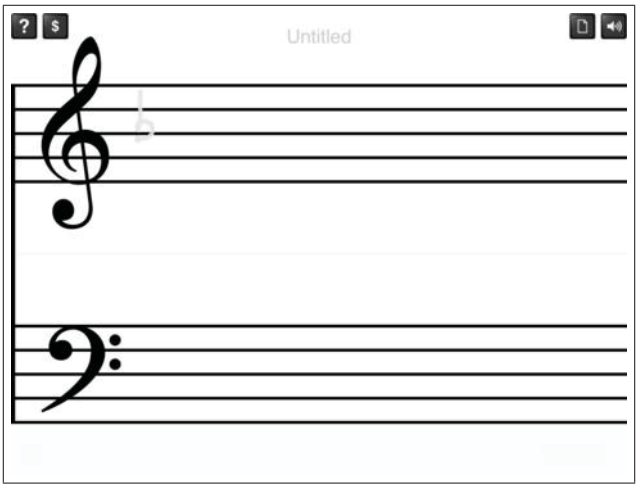


Fig. 5.19: A typical example of a Diatonic Column layout *(QuaverPad)*.

The two defining characteristics of this abstraction are the vertical nature of the buttons, their alternating nature, and the five line / four space convention. The Diatonic Column also has the dubious honour of being the least varied of all abstractions. Despite the fact that, in theory, additional lines and spaces can be added *ad infinitum*, no examples of such an 'innovation' could be found. Likewise, buttons of any sort could be used, as long as they alternate between 'line' and 'space' button, but no such examples could be found. Figure 5.20 thus simply presents an abstracted staff. The obvious variation on this mapping is to change the tuning system used: either moving to a chromatic scale with twelve buttons, moving to a microtonal, equal-tempered scale, or moving to an hexatonic / octatonic scale, with an appropriate number of buttons.

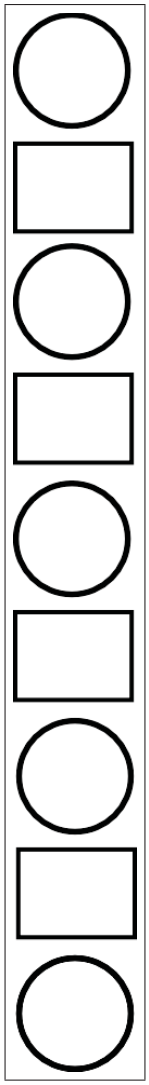


Fig. 5.20: The Diatonic Column abstraction.

5.9 Diatonic Column / Pentatonic Column

This abstraction is simply a vertical piano, based on the piano roll used in many sequencers. Pitch increases from bottom to top. On iOS, this layout is generally used for sequencing, rather than performance, as can be seen in Figure 5.21. Timbre and volume are generally controlled by other aspects of the app. As with the Diatonic Column, this could be tiled to include time, moving from left to right. Apps that use this layout include DAWs, sequencers, and synths.

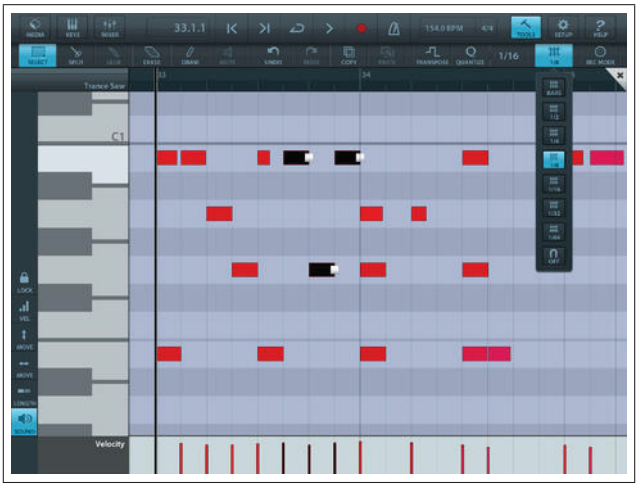


Fig. 5.21: A typical example of a Diatonic Column / Pentatonic Column layout *(Cubasis)*.

Unlike the Diatonic Row / Pentatonic Row abstraction, this abstraction does not often change the shape, size, and colour of the buttons that represent each note. On the other hand, the two rows of buttons can sometimes become a single column, with only colour to disambiguate them (Figure 5.22a). Like Diatonic Row / Pentatonic Row, however, the exact number of buttons displayed varies (Figure 5.22b). An abstracted piano roll is displayed in Figure 5.23. Variations on this are similar to those of a horizontal piano: varying the number of buttons per octave, and thus varying the tuning of the mapping.

(a) A single column *(LivKontrol)*.

(b) Varying number of buttons *(Music Studio)*.

Fig. 5.22: Two atypical Diatonic Column / Pentatonic Column layouts.

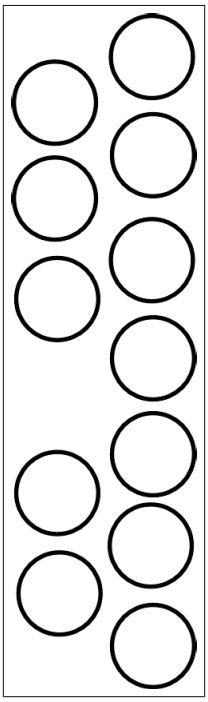


Fig. 5.23: The Diatonic Column / Pentatonic Column abstraction.

5.10 Hexagonal / Triangular Grid

This is an abstraction of a Tonnetz. The Tonnetz, is, of course, already an abstraction from music theory. A grid of buttons at 60-degree angles, it provides a mapping of pitch space that is suitable for describing Romantic harmonies [40]. Grids similar to a Tonnetz are also seen on accordions and on various isometric keyboards. The typical mapping is for one line to increase in minor thirds, one in major thirds, and one in perfect fifths, as seen in Figure 5.24. Other tunings are more than possible: Park and Gerhard have examined various tunings and the math behind them [41]. On iOS, this layout only handles changes in pitch (though sometimes playing chords rather than pitches), ignoring volume and timbre.

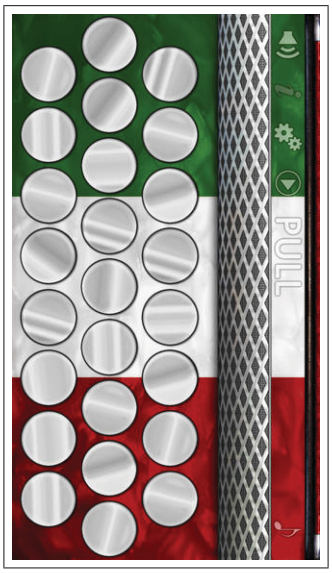
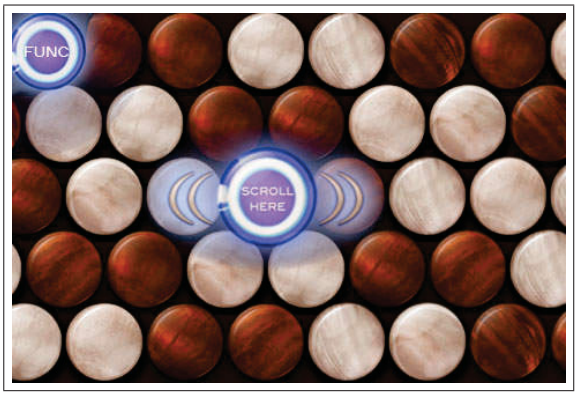
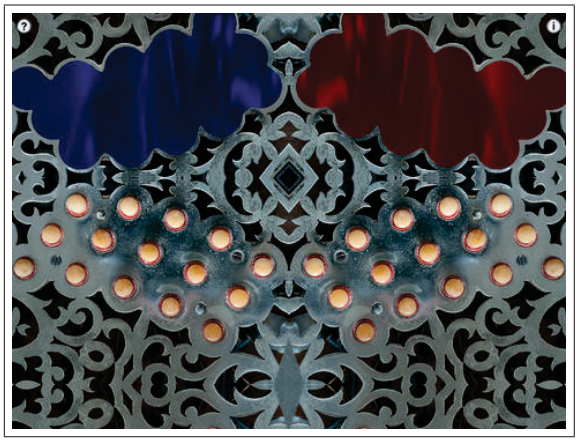


Fig. 5.24: A typical example of a Hexagonal / Triangular Grid layout *(App name not available).*

The grid nature of this layout is paramount. Buttons colour, size, and shape can vary (Figure 5.25a), and the rotation of the overall grid is also not set (Figure 5.25b), though the grid itself cannot be changed signifigantly. An abstraction can be seen in Figure 5.26. The sundry tunings discussed by Park and Gerhard [41] point to many possible methods for controling pitch space with a Hexagonal / Triangular Grid. Variations between melody and harmony are also interesting. One can imagine a Tonnetz where each button plays a chord rather than a note, and chord complexity varies with some dimension (out from the center, radially, etc). Timbral mappings of this sort are also possible.



(a) Varying button colour *(Accordio)*.



(b) Varying grid rotation *(Concertina XL)*.

Fig. 5.25: Two atypical Hexagonal / Triangular Grid layouts.

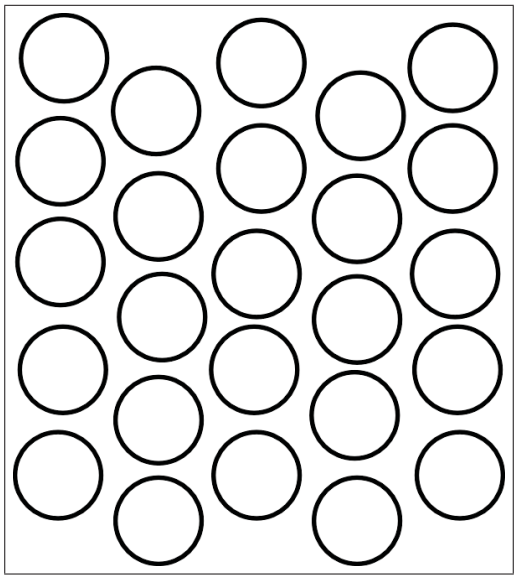


Fig. 5.26: The Hexagonal / Triangular Grid abstraction.

5.11 Circle

An abstraction of the theoretical circle of fifths, and the real-world steelpan, the Circle abstraction increases in pitch clockwise, though the exact tuning and starting point vary. Figure 5.27 shows an example. These layouts offer no control over timbre or volume, on iOS.

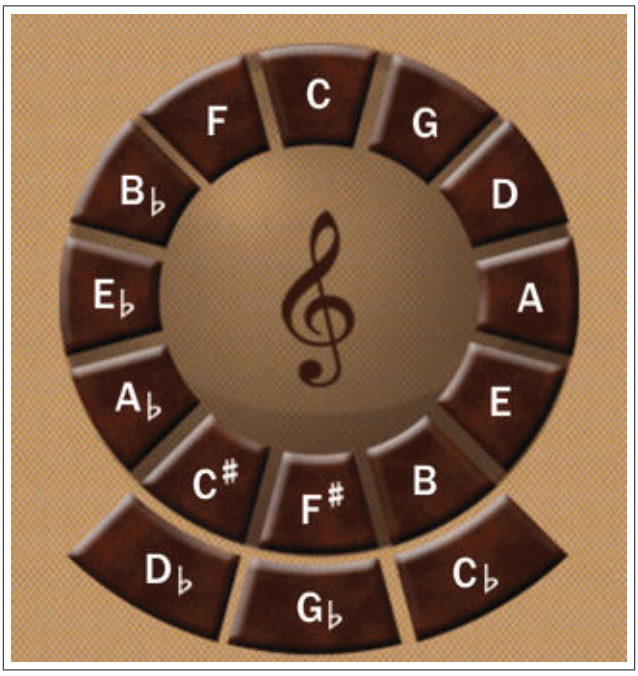
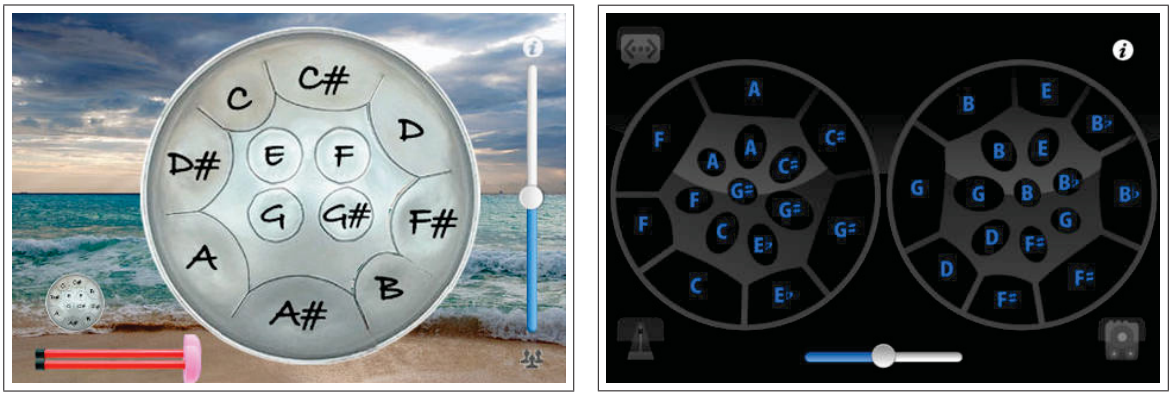


Fig. 5.27: A typical example of a Circle layout *(Major Circle of Fifths)*.

Unsurprisingly, it is the circular nature of this layout that is key: button sizes, shapes, and colours can vary (Figure 5.28b), though steelpan apps generally use button size to represent register. The number of buttons also varies (Figure 5.28b]). An abstraction can be seen in Figure 5.29. Various tunings are possible, depending on the number of buttons (the traditional circle of fifths requires twelve buttons). Concentric circles are also possible (and are a feature of steelpans), and suggest the mapping of other parameters, such as timbre or volume.



(a) Varying button shape, size, and colour *(Dancing Steel Drum)*.

(b) Varying numbers of buttons *(Double Tenor - Steel Pan Orchestra)*.

Fig. 5.28: Two atypical Circle layouts.

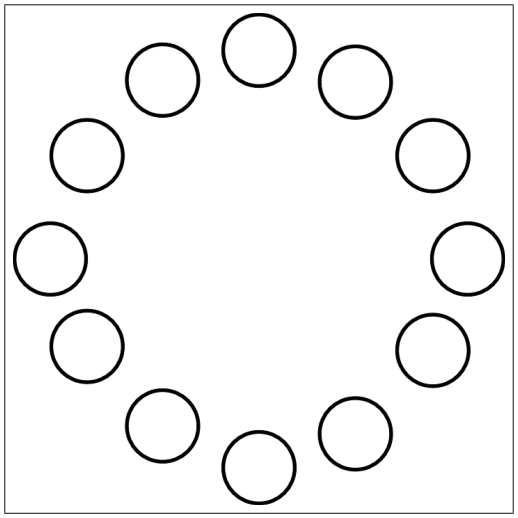


Fig. 5.29: The Circle abstraction.

5.12 Orthogonal Grid

A grid of buttons, larger than 3x3, arranged in perpendicular rows and columns. Similar to the Multiple Rows and Multiple Columns abstractions, this layout is unique in that it does not lead to a single mapping. Like the Monome [42], the Tenori-On [43], or the MPC [15], there are many potential ways to map this layout. An example can be seen in Figure 5.30.

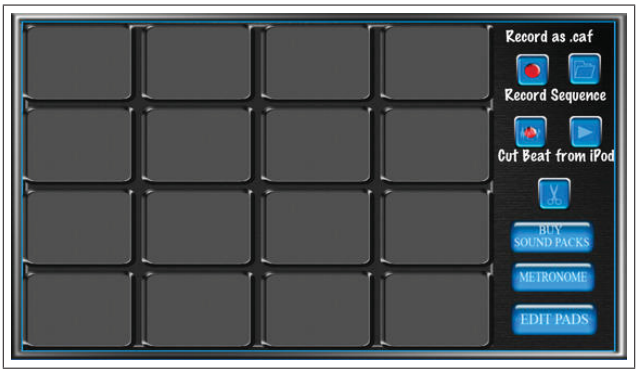


Fig. 5.30: A typical example of a Orthogonal Grid layout *(App name not available).*

Three common uses for this abstraction are as a sequencer, as an MPC / Percussion Input, and as an X-Y Pad. The Sequencer (Figure 5.31a) maps time from left to right, quantized to the columns, and pitch, quantized to the rows, from bottom to top. The exact tuning and time resolution vary with the number of buttons, which is typically higher: 8x8 or more. The MPC / Percussion Input (Figure 5.31a), classically a 4x4 grid, maps various samples, typically percussion, to each button on the grid. Like most iOS apps, this precludes any control of volume or timbre. More 'core' drum sounds (kick, snare, etc) are typically mapped to the bottom left, but the mappings vary widely. The X-Y Pad (Figure 5.31c) provides control over two parameters, one on each axis. A typical mapping for note input is of pitch from left to right, and of filter or volume envelope from bottom to top. Alternatively, a typical mapping for an effect might be delay time from left to right and delay feedback from bottom to top. Although most X-Y Pads appear to offer continuous control, this is often not the case, especially for pitch input. An abstraction of the Orthogonal Grid can be seen in Figure 5.32.

(a) Sequencer *(Beat Beat Play)*.

(b) MPC / Percussion Input *(Beat-Machine)*.

(c) X-Y Pads *(76 Synthesizer)*.

Fig. 5.31: Three use cases for the Orthogonal Grid layout.

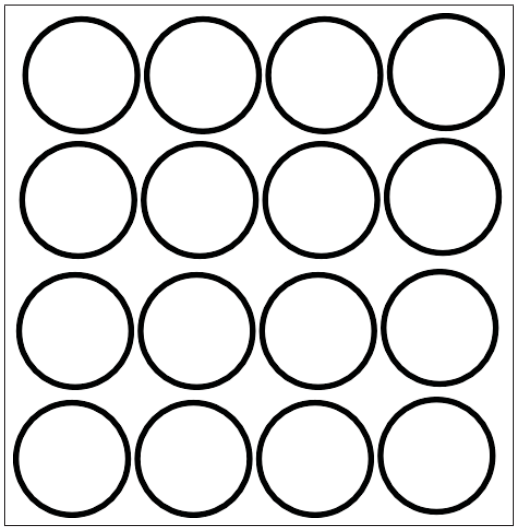


Fig. 5.32: The Orthogonal Grid abstraction.

5.13 Rotary Encoder

This is a peculiar abstraction. The use of rotational knobs to modify timbre is almost universal in both the world of iOS and in countless real-world synthesizers and mixers, as can be seen in Figure 5.33. These controls typically deal with a single aspect of the synthesis or processing of sound. Most of the layouts discussed here focus on pitch, but many apps that use them also provide timbral controls, in the form of these rotary encoders.

Fig. 5.33: A typical example of multiple Rotary Encoders *(76 Synthesizer)*.

Button size and colour can vary (Figure 5.34a), though the shape is almost always circular. There is no connection between the layout of multiple Rotary Encoders and what parameters they control (Contrast Figure 5.34a and Figure 5.34b). A perhaps extraneous abstraction can be seen in Figure 5.35. Two potential variations on this venerable design would be to first assign an encoder to more than one parameter, thus increasing the integrality of the mapping, and, second, to vary the change of the parameter with the extremity or speed of the rotation.

(a) Varying button size and colour *(Auria iSEM)*.

(b) Varying layout of multiple rotary encoders *(Auria)*.

Fig. 5.34: Two atypical Rotary Encoder layouts.

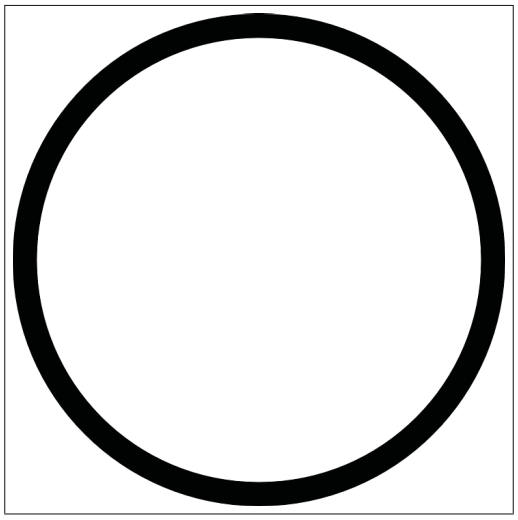


Fig. 5.35: The Rotary Encoder 'abstraction'.

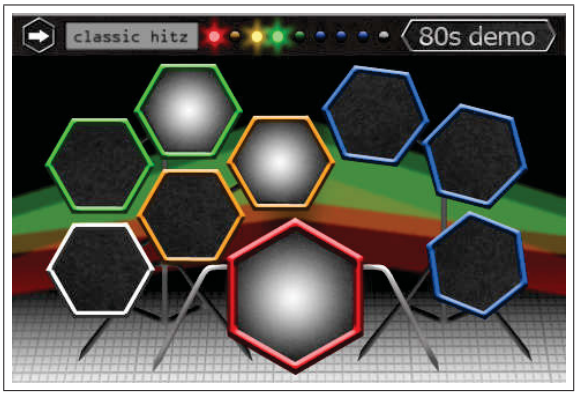
5.14 Timbral / Radial

This abstraction, based on the drum kit, is unique in that it deals only in unpitched sounds. A rough semi-circle, with a varying number of buttons, drum sounds are not mapped to it in any geometric way (though one could argue that low sounds move out from the center), but rather by cultural context. An example can be seen in Figure 5.36. On iOS, each button / drum produces a single and unique sound, with no controls for volume and no way of altering the timbre of each button.

Fig. 5.36: A typical example of a Timbral / Radial layout *(Cool Drums)*.

This layout depends on its lack of geometric cohesion: it is not a strict semi-circle, but does tend to follow certain standards, with buttons representing hi-hats, cymbals, and so on in the positions that would be expected of them. Button size, shape, and colour are not standardized (Figure 5.37a), and nor is the number of buttons (Figure 5.37b). An abstraction of a drum kit can be seen in Figure 5.38

Potential expansions to this mapping might include radically increasing the number of buttons, in order to differentiate between, for example, the bell or the rim of a cymbal. The iOS app store also has many drum apps that deal with percussion outside of the drum kit, such as tabla, cajón, and congas - a detailed review of their layouts is, however, beyond the scope of this thesis.



(a) Varying button size, shape, and colour *(80s Drumr)*.

(b) Varying number of buttons *(BaDaBing)*.

Fig. 5.37: Two atypical Timbral / Radial layouts.

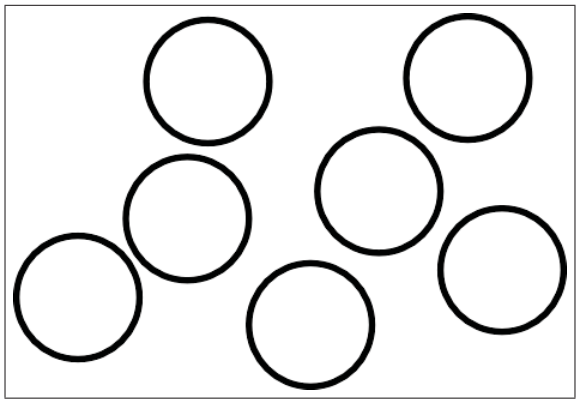


Fig. 5.38: The Timbral / Radial abstraction.

5.15 Multiple Column

This abstraction represents the faders on a mixer, quantized to individual buttons. As with the Rotary Encoder discussed above, mixers are very common in both iOS apps of varying sorts, and in real-world music making. An example can be seen in Figure 5.39. As most touchscreen applications lack a sensor for determining the force of a user's touch, the use of one or more faders is typical to control the output level. Although this layout uses faders rather than buttons, it is prevalent enough to be mentioned.

Fig. 5.39: A typical example of a Multiple Column layout *(AC-7 Core HD)*.

The number of faders varies with the number of inputs or outputs under control - sometimes in extreme ways (Figure 5.40). An abstraction can be see in Figure 5.41. Potential variations might involve changing colours to indicate higher or lower volumes, or adding integral timbral mappings to match the amplitude mapping: a 'presence' fader, rather than a volume fader.

Fig. 5.40: Varying number of faders / columns *(Allen & Heath iLive Tweak)*.

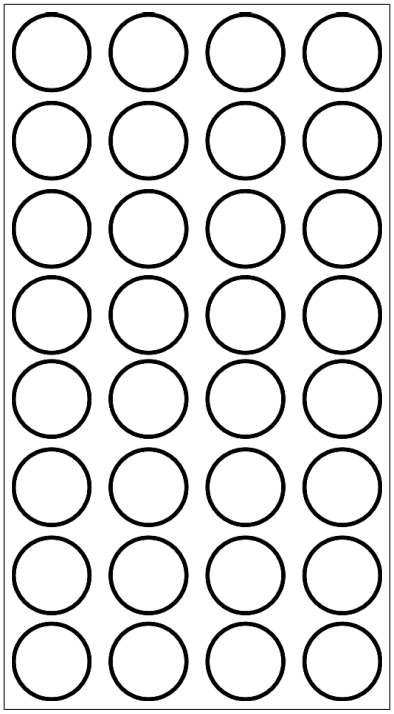


Fig. 5.41: The Multiple Column abstraction.

5.16 Conclusion

This chapter has listed and discussed the most prominent abstractions of control layouts, on iOS devices and in graphic scores. Each abstraction has been defined in terms of its positive and negative attributes, ranging from the topological button layout to the colour, shape, and size of the buttons. Despite the distance from acoustic instruments, some vestiges of the physical world still remain: larger buttons tend to play back lower pitches. Likewise, many abstractions take their inspiration from physical, electronic synthesizers, including mixers for volume and rotary encoders for timbre. Many conventions from the theoretical world, such as the Tonnetz and circle of fifths, appear as abstractions. Other abstractions, such as the piano roll or piano keyboard are based on real-world instruments.

Regardless of the source of a given abstraction, pitch is by far the most prominent parameter mapped, and it is usually mapped in a linear fashion. Abstractions that deal with timbre or volume, such as the Rotary Encoder or the Multiple Column abstraction exist apart from pitch control. This reflects the separability of parameters used in both modern music theory [38] and in current music applications. Much variation is possible here: the concentric circles that are typical of Steelpan apps could include variations in timbre and volume as well as in register. X-Y Pads could also control multiple parameters, rather than the typical pitch and envelope control. Interestingly, the shape of the buttons is unimportant, despite the results of research around the use of shape in synesthesia [44].

These abstractions attempt to reach the simplest, most topological versions of themselves: the layouts that express the most information about their musical mappings in the least amount of data. These examples can be used as both the basis of sundry application designs, or as jumping-off points for new and varied touchscreen applications. As with the data enumerated in previous chapters, this chapter can drive incredibly typical music applications or, with a few graceful twists and variations, deeply novel applications and interfaces.

Chapter 6

Software Tools for Two-Dimensional Mapping

6.1 Introduction

This chapter describes three pieces of related software, all built with the goal of automating the mapping process for two-dimensional interfaces. Automatic mapping of this sort is a pertinent problem. Real-world iOS applications like BeatSurfing [45], TouchOSC [46], and Lemur [47] allow end users to construct their own control layouts. As these control layouts scale up, mapping them becomes non-trivial, requiring the user to point and click dozens upon dozens of times. The software discussed in this chapter will provide an algorithm for automating this process, returning mapping data to the end user based on the nature of the control layout.

The first piece of software is an algorithm for classifying layouts of buttons as one of the abstractions defined in the previous chapter. Once a layout has been classified, pitches can be mapped to each button, based on the nature of the abstraction (a circle of buttons maps pitch clockwise, as the circle of fifths, for example). This classification is achieved using machine learning, with some heuristic adjustments to the task at hand. The second piece of software is an open, web-based classification API that provides access to the algorithm. Built in Python as a RESTful web service, this API takes data about the control layout, modifies it, and passes it to the classification algorithm, and then returns the resulting classification and mapping. The third is *Pattern Recognition*, a sample application. *Pattern*

Recognition allows users to create their own control layouts and send their data to the classification API. It then applies the returned mapping to the control layout.

Pattern Recognition and the underlying classifier do not seek to provide an iconically perfect answer to what the 'best' possible mapping for a given layout is. Rather, they act as an implementation of the principles described in the previous chapters. As the abstractions defined in Chapter 5 mostly deal with pitch, the classification algorithm likewise mostly deals with pitch. It was hoped that mappings for parameters such as timbre and volume would be encoded in the layout of buttons, as mapping for pitch is. As this was not the case, the engineering in this chapter has limited itself accordingly.

6.2 Related Work

Tools to aid the mapping process include McGill's own LibMapper [48] and Digital Orchestra Toolboox [49]. The Digital Orchestra Toolbox is a collection of Max / MSP objects that simplify the mapping process. Examples include simple type conversion tools; maximum, minimum, and peak detection tools; MIDI and OSC wrappers; and various filters and integrators. LibMapper allows for arbitrary connections to be made over OSC, with arbitrary processing of the data being sent over OSC. Those signals can then be easily remapped, transformed, and modified, using one of several user interfaces.

MnM [50] provides Max/MSP-based tools for mapping and machine learning around mapping, including functionality for implicit mapping via machine learning. Along these lines, the Wekinator [51] provides implicit mapping via human input of almost any sort, from audio input to webcam data. These tools are primarily focused on interfaces and instruments that include more than two dimensions. urMus [52], in contrast, offers a patch-based mapping environment specifically for mobile devices, inclusive of touches, inputs from gyroscope and accelerometer inputs, etc. urMus also allows users to define two-dimensional control layouts using the Lua[1] scripting language.

The above tools allow users to construct complex, sophisticated mapping processes, but are agnostic in terms of the controls presented to them. Many touchscreen apps allow users to design their own control systems, and then map them to MIDI or OSC messages. Apps of this sort include BeatSurfing [45], TouchOSC [46], Lemur [47], and Control [53]. The Lemur application grew out of the original hardware Lemur [54], which provided

[1] http://www.lua.org/

the same level of customization in bespoke hardware (and with a resistive, force sensitive touchscreen). The Lemur app allows the user to populate their screen with various buttons, faders, and other controls, and then map them accordingly. TouchOSC and Beatsurfing provide similar functionality, though with a slightly more limited set of objects. Control allow users to create both their own control layouts and define their own control objects, from complex radial arrays to particle-based controls. Control, like Lemur and TouchOSC, can send MIDI or OSC, whereas BeatSurfing sends only MIDI data.

These applications provide the user with a high level of control both in terms of creating the controls and mapping the controls. Mira [55], on the other hand, creates touchscreen controls 'for free', based on already extant controls in Max/MSP. Any object inside a particular frame in Max/MSP will automatically be sent to the touchscreen, and will be controllable from there. The speed and transparency of this process was an influence on the design of the classification API. The user interface and concept of BeatSurfing also influenced the design of *Pattern Recognition*.

6.3 Classification & Machine Learning

The first step was to classify any given layout of buttons as one of the abstractions listed in Chapter 5. As close matches for a given abstraction need to be assigned correctly, machine learning was an obvious choice. This classification algorithm did not have to be tightly coupled to the demonstration software, so the Python programming language, with its exceptional data-processing abilities, was a good choice. Python also has many fine libraries for scientific computing, including the SciKit-Learn [20] library. SciKit-Learn was selected for this task due to its open source nature, and the large number of supervised and unsupervised learning algorithms that it implements.

A key issue was selecting what features to use to train the machine learning model. As the model had to be invariant with regards to absolute position, an array of features containing the difference between each button was constructed, in terms of the location, size, shape, rotation, and colour. This data was used to train a Support Vector Machine (SVM), which performed well on initial tests. Specifically, five of the abstractions (Diatonic Row / Pentatonic Row, Diatonic Row / Size, Small Grid, Diatonic Column / Pentatonic Column, Column / Size / Shape) were included in the model, which performed with 100% accuracy on 20 instances of test data.

An examination of the data from Chapter 5 suggested that many of the features used were, in fact, extraneous. Because many of the abstractions listed rely only on the relative locations of the buttons, all features except the normalized X distance and normalized Y distance between each button were removed from the model. This again performed at 100% accuracy on the twenty instances of test data.

When adding the Orthogonal Grid abstraction, however, things went wrong. The test data fell to 20% accuracy, and validations against the data that trained the model began to fail. It was discovered that this was caused by the large increase in dimensions. A test Diatonic Row / Pentatonic Row has 264 dimensions, as it has twelve buttons. A 5x5 Orthogonal Grid, however, has 1200 dimensions. As the number of dimensions must be constant, smaller instances were padded with zeros. These zeros lead to all instances with a relatively small dimensionality being classified as the same.

In order to fix this, new features were tried: the number of buttons, the mean distance between buttons, and the standard deviation between buttons. It was hoped that encoding the number of buttons while keeping the dimension constant would lead to better results. This did not significantly improve on the 20% accuracy. The normalized mean distance was tried, with similar results. Different algorithms within SciKit-Learn were also tried, with Nearest Neighbours[2] giving the best results.

After some soul-searching, it was decided to dodge the issue. Rather than making a one-size-fits-all model, several different models would be built, depending on the dimensonality of the input. This may raise the hackles of machine learning experts, but it has several benefits. As several abstractions, such as the Diatonic Row / Pentatonic Row can vary in size (typically at the octave), it is reasonable to build different models for them. Layouts with under twenty buttons were passed to the first model, and buttons with twenty or more buttons were passed to the second model. This solved the problem: adding a model for a twenty-four-button Diatonic Row / Pentatonic Row and twenty-five-button Orthogonal Grid led to 100% accuracy across both models, encompasing twenty-eight instances of test data. These two models covered seven abstractions, using the Nearest Neighbours algorithm mentioned above. The next step was to create an easy way for other applications to access the models.

[2]http://scikit-learn.org/stable/modules/neighbors.html

6.4 API & Server

An Application Programming Interface (API) provides a set of formalized ways to access an algorithm or data source. In this case, the classification algorithm itself is being accessed: the API layer sends data to the algorithm, and returns the result to the user. In order to make the classification algorithim accessible to the internet at large, the classification API was built as a web service, using the Flask[3] web framework, and hosted on the Heroku[4] cloud platform. The end user sends JSON-formatted[5] data to the server in the body of an HTTP POST request. The server then passes the data to the algorithm defined above, and returns the resulting classification from the algorithm to the API to the end user.

While a web-based API suggests a Representational State Transfer [56] (REST) model, the classification API is more REST-ish than RESTful. It does not define the typical Create/Read/Update/Delete verbs, and indeed misuses POST to return data, rather than to create or update data. With that said, it fits many of the other criteria for a REST API: it has a client / server model, it is stateless, and it presents a consistent interface. This consistent interface was one primary reason for building an API, rather than implementing the machine learning within the example *Pattern Recognition* application itself. In addition to allowing public access to the algorithm, building a separate API allowed for loose coupling between the front and back end of the software: the machine learning code could be totally re-written without touching the front end, and vice-versa.

This goal led to a change in the structure of the data sent to the classification API. The model uses the normalized distance between buttons to select a classification. However, this requires large amounts of heavy lifting on the client side. It was decided to simply send the raw button data to the API, and allow the API to work out the button distances, or whatever features are required by the model. This made the API easier to use, and increased the separation between the client and server side software. It also lowered the amount of data to be sent, as can be seen in the comparison of example data on the next page.

[3] http://flask.pocoo.org/
[4] https://www.heroku.com/
[5] http://www.json.org/

Old Data Format

```
{0: {1: {``location'': {``y'': 0.0, ``x'': −0.166}},
2: {``location'': {``y'': 0.0, ``x'': −0.333}},
3: {``location'': {``y'': 0.0, ``x'': −0.5}},
4: {``location'': {``y'': 0.0, ``x'': −0.666}},
1: {0: {``location'': {``y'': 0.0, ``x'': −0.166}},
2: {``location'':{``y'': 0.0, ``x'': −0.833}},
3: {``location'': {``y'': 0.0, ``x'': −0.5}},
4: {``location'': {``y'': 0.0, ``x'': −0.833}},
2: {0: {``location'': {``y'': 0.0, ``x'': −0.333}},
1: {``location'':{``y'': 0.0, ``x'': −0.833}},
3: {``location'': {``y'': 0.138, ``x'': −0.75}},
4: {``location'':{''y'': 0.0, ``x'': −0.833}}, etc}
```

New Data Format

```
[{``location'': {``x'': 75,``y'': 200}},
{``location'': {``x'': 195,``y'': 200}},
{``location'': {``x'': 315,``y'': 200}},
{``location'': {``x'': 435,``y'': 200}},
{``location'': {``x'': 555,``y'': 200}}]
```

The API functions as a wrapper around the primary classification algorithm. The initial design of the API left the mapping choices to the client side, based on the classification returned. While this is desirable functionality in cases where the sound production technique used by the client is very particular, in many cases the client is using a sound production technique that can use MIDI numbers or raw frequency as an indication of pitch. The API was thus updated to return both the classification, as a string, and an array of mappings. The array of mappings provides a MIDI number and a frequency for each button. These are returned in the same array that the user sent in, in order to make applying them easier on the client side. These mappings were defined based on the abstractions in Chapter 5. A classification of Diatonic Row / Pentatonic Row would simply map a chromatic scale

to the buttons, increasing from left to right, with some corrections for range based on the number of buttons. An example of the returned data is below. Note that the mapping string returned is simply 'piano': this was done in order to make the API usable by those who have not read this thesis.

Return Data Format

```
{''mapping'': ''piano'',
buttonData: [
{''location'':{''x'':75,''y'':200},
''noteFreq'':261.63, ''noteMIDI'':60},
{''location'':{''x'':195,''y'':200},
''noteFreq'':293.67, ''noteMIDI'':62},
{''location'':{''x'':315,''y'':200},
''noteFreq'':329.63, ''noteMIDI'':64},
{''location'':{''x'':435,''y'':200},
''noteFreq'':349.23, ''noteMIDI'':65},
{''location'':{''x'':555,''y'':200},
''noteFreq'':392.00, ''noteMIDI'':67},
]}
```

The API also recognizes an optional 'adventure' parameter. As many of the mapping choices defined in Chapter 5 are rather straight-laced, a higher adventure value returns more interesting mappings: hexatonic scales, microtonal pitches, and so on. Further extensions to the API could include passing in the synthesis parameters to be mapped, within reason. Likewise, returning a list potential mappings is also a possibility, especially if the input data does not clearly match one abstraction. The interface to the classification API itself, however, is well-defined and flexible, allowing for the underlying algorithm to be changed without impacting the client application. The next section discusses a sample client application, *Pattern Recognition*.

6.5 Front End & Client

Pattern Recognition is a simple browser application, designed to offer an example of how to use the classification API. It allows users to create their own layout of buttons, and then obtain a mapping for those buttons from the API. It is built in JavaScript and HTML5, for portability and ease of distribution, and can be found at tide-pool.ca/pattern-recognition. Synthesis is provided by the WebAudio API, and is based on Stuart Memo's Sympathetic Synthesizer project [57].

Figure 6.1 shows the starting interface. As with similar applications like BeatSurfing, the user can add buttons to the main layout area by left-clicking on it. The radius and shape of the button are selected via the sliders in the upper left, and the button colour is selected via the text box just under the sliders. The current setting is displayed below, in the appropriate colour. Example layouts for each of the currently working abstractions can be displayed using the buttons in the bottom left. At the time of writing, working abstractions include Piano, Xylophone, Zither, Piano Roll, Small Grid, Large Grid, and Big Piano (a two-octave piano).

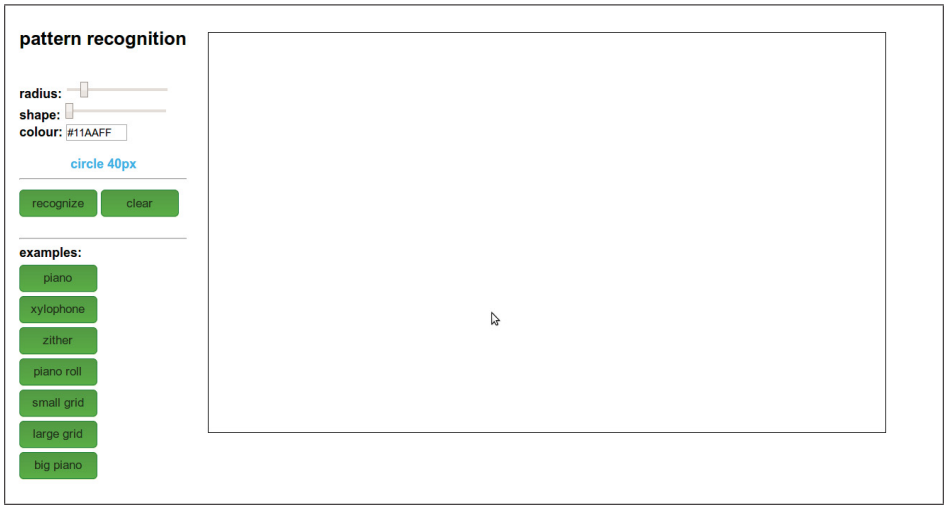


Fig. 6.1: The interface for *Pattern Recognition*.

Once a layout has been created, the 'recognize' button is pressed, and the client sends the collected button data to the server. Figure 6.2 shows an example layout created by a user, with annotations added from the mapping process. The layout has been correctly classified as a Small Grid. Figures 6.3 shows another example.

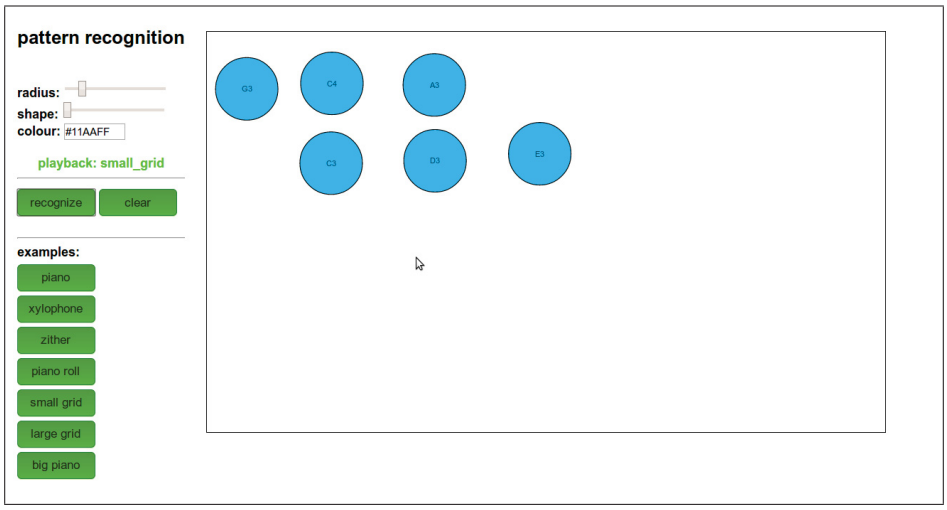


Fig. 6.2: A user-defined Small Grid mapping.

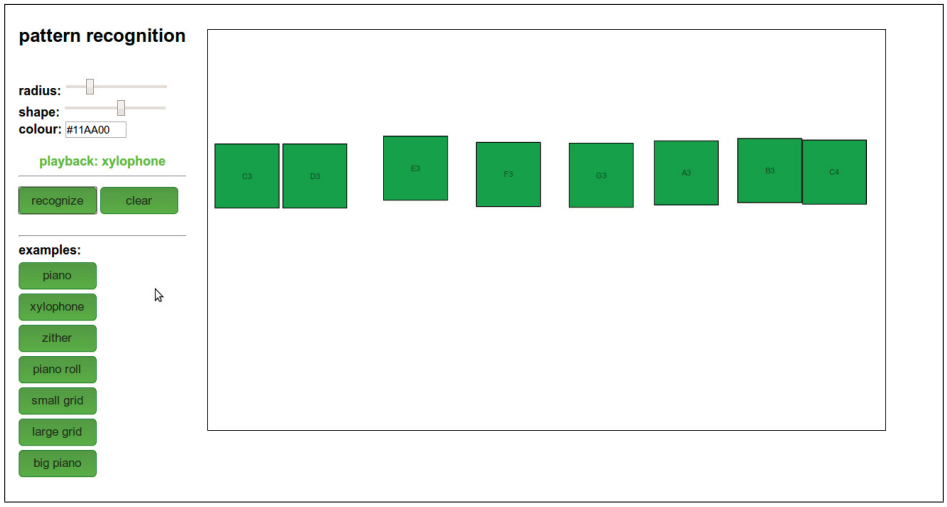


Fig. 6.3: A user-defined Diatonic Row / Size mapping.

The above images offer examples of the tool working correctly. Some inputs, however, cause problems. Figure 6.4 shows an incorrect classification, and the poor mapping that results from it. This is an example of the classification algorithm failing: a Diatonic Row / Size (or 'xylophone') would be more correct here. This is probably a result of the single very low button. Figure 6.5 shows a correct classification, with a flaw in the mapping algorithm. The bottom-right button is lower than the bottom-left button, which, as Small Grids are mapped from bottom-left to top-right, results in an incorrect mapping.

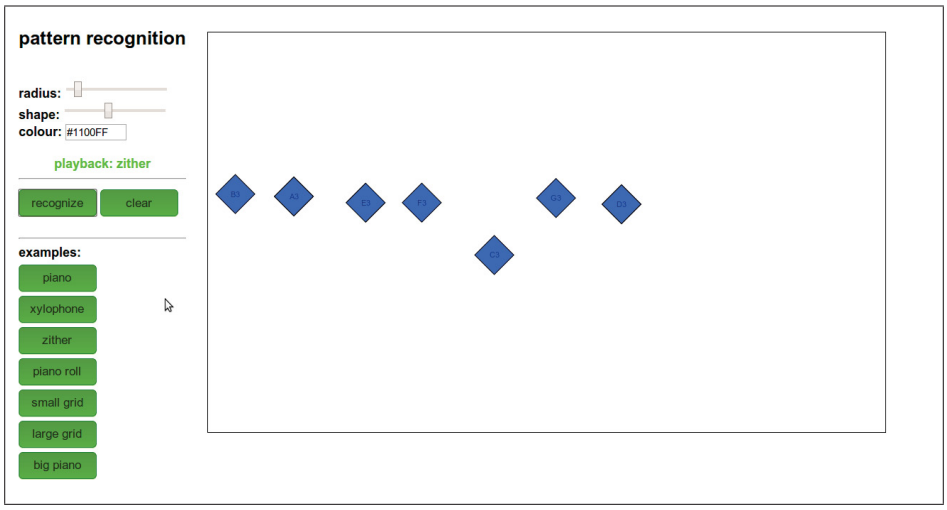


Fig. 6.4: An incorrect classification of Column / Size / Shape.

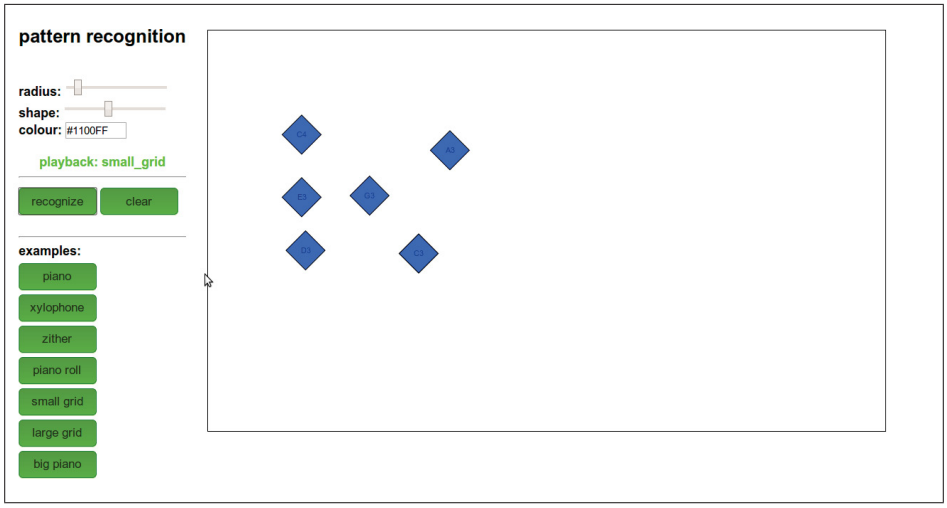


Fig. 6.5: An incorrect mapping of a Small Grid layout.

Pattern Recognition is thus not perfect - though the mistakes can be as interesting as the correct results. As a tool for making music, it is somewhat lackluster, largely due to the static nature of the synthesis. A similar program with better synthesis, and multitouch capabilities would be a far more interesting interface. *Pattern Recognition* does, however, provide a working prototype of automatic mapping, and acts as an example of how to make use of the classification API.

6.6 Conclusion

This chapter has presented the engineering results of the thesis. A machine learning based classification algorithm was created, capable of discriminating between the various abstractions discussed in Chapter 5. That algorithm was then wrapped in an open HTTP API, in order to make the classification algorithm widely available. In the course of building the classification API, the exact nature of the interface changed, moving more processing away from the client and to the server-side API. Likewise, it was decided to alter the API so that it could return the mapping, in terms of both MIDI note number and absolute frequency, as well as the classification. The *Pattern Recognition* sample application was built on top of this API. It provides simple tools to allow a user to construct her or his own layout of buttons. It then maps those buttons based on the response of the classification API.

These tools, and *Pattern Recognition* in particular, serve as a proof of concept for the data and abstractions discussed in this thesis. As has been mentioned, mapping is difficult. *Pattern Recognition* does not provide a foolproof 'best' mapping for a layout, but it does provide a layout that expresses the trends and mappings discussed in this thesis.

Indeed, *Pattern Recognition* would benefit from a more robust input method, and a higher fidelity synthesizer. The classification API could increase the detail of its mappings to include various synthesis parameters in both input and output, or to return a list of possible mappings. The underlying classification algorithm should be made more robust while also including more abstractions. A wholesale rework of the classification algorithm is also possible: removing the heuristic around dimensionality in favour of a pure machine learning approach would result in much simpler code, while also being more satisfying in principle.

The code discussed herein has met the primary technical goal of the thesis: automatically mapping two dimensional layouts of buttons. The classification API makes this possible - it simply needs to be used by developers.

Chapter 7

Conclusion

This thesis has researched musical mapping techniques in two dimensions, with the underlying goal of attempting to automate the process of mapping two-dimensional layouts of controls. Software for creating such control layouts already exists; however, assigning each of many controls to a musical event is often a slow and unwieldy process. The final engineering aspect of this thesis is a first step to solving this problem, providing an open API for automatic classification and mapping. Data for the API's underlying algorithm was gathered from Chapter 2, 3 and 4, comprising several reviews of mapping trends and techniques.

Chapter 2 presented an in-depth review of the most popular music apps in the iOS App Store, investigating their metaphors and mappings. A majority of the applications examined presented the user with metaphors based on real-world instruments or devices: pianos, drum kits, mixers, synthesizers, and so on. In these applications, a preponderance of simple, one-to-one mappings was found, with pitch generally increasing from left to right, and volume / timbre increasing vertically, from bottom to top. Time, when it was a parameter, moved from left to right. This chapter also examined several novel applications that did not present a real-world metaphor to the user. These applications had the most creative mappings, with generally fewer left-right and up-down mappings. Some highlights included diagonal / Tonnetz-based mappings of pitch and radial mappings of volume. The use of additional sensors on the device also became prevalent here: tilting mapped to timbre, shaking mapped to pitch vibrato, and so on.

In contrast to this detailed review, Chapter 3 performed a high-level overview of *every*

music app in the iOS App Store, classifying them based on their descriptive text. Though machine learning techniques were attempted, the underlying text data was simply too noisy for automatic techniques to give good results. Thus, all 38,750 apps were classified by hand. This had the happy result of creating a human-classified dataset that can be used for further investigations into the iOS app store ecosystem, or for investigations into automatic classification of text. More pertinently, this process found fifty-six new types of music apps, here organized by mapping rather than by metaphor. As in the previous chapter, mappings were enumerated, and, as in the previous chapter, simple, one-to-one mappings dominate. Some interesting mapping techniques were discovered, however: the use of two touches in rapid succession as a timbral modifier is a good example.

Chapter 4 takes the examination of mappings into new territory, moving to the graphic scores of the twentieth century. Sixteen scores were examined, ranging from the stark blocks of Earle Brown and Karlheinz Stockhausen to the subtle colours of Halim El-Dabh and Wendy Reid. Pitch here is often mapped vertically, as time generally moves from left to right. Many scores show the same focus on the separation of parameters that is the norm in iOS applications. A large number of scores, however, leave one or more parameters undefined, trusting to the discretion of the player. Likewise, many scores combine pitch, timbre and articulation into a single, almost spectral graph. Wildly creative mappings are more common here, as is more of a focus on timbre. Tenney's *BEAST*, with its combined mapping of pitch and roughness, provides a creative solution, as does Hans-Christoph Steiner's *Solitude*, which relates shape to the granular synthesis parameters used.

Chapter 5 summarized the research of Chapters 2, 3, and 4 by boiling down the most common and iconic layouts of controls into their topological abstractions: the minimum set of constraints for a given layout of buttons to use a certain mapping. Fourteen of these abstractions were listed, ranging from the canonical piano keyboard to the simply rotary knob. Examples were provided for each, along with potential expansions of the typical mappings.

Finally, Chapter 6 discussed the engineering aspects of the thesis: developing a machine learning based classification algorithm to recognize the abstractions discussed in Chapter 5; developing an open classification API to the classification algorithm; and building an example application that makes use of that API to classify and map pitch to a user-designed layout of controls. This chapter acts as a proof of concept for the stated goal of the thesis - that the automation of mapping is both possible and useful.

7.1 Contributions

The largest contribution is the reviews of Chapters 2 and 3, which focus specifically on iOS applications, and the various mappings used by them. Chapter 3 provided a high-level overview of *all* apps in the music category, and crucially provided the raw, human-classified text data for each app. This data will aid future investigations of the app store and provide ground truth for future work around automatic text classification techniques. Chapter 2 provided an in-depth examination of the most popular music apps, including novel and unique applications. Both of these chapters listed numerical demarcations of mapping trends, in order to point interface designers and future researchers to the most or least common ways of mapping a given parameter.

The two technical contributions, described in Chapter 6, are the algorithm for classifying and mapping layouts of buttons based on the location of the button, and the open classification API that provides access to this algorithm. While the *Pattern Recognition* example application is not unimportant, and provides a hands-on example of how automatic mapping might be used, the classification API and the backing algorithm are much more important. The API provides access to automatic mapping / layout classification from a simple HTTP POST request, allowing the classification algorithm to be accessed by anyone, for any project.

The code created in Chapter 6 was based on the fourteen abstractions defined in Chapter 5, which will prove useful as theoretical constructs in terms of both interface design and pitch space research. They were based on the raw data collected in the previous chapters, including the graphic score review in Chapter 4. While less directly applicable to music technology, this review holds value for composers and interface designers alike. It provides a compendium of graphic techniques for composing, and relates those to the mapping of input parameters in a novel way.

7.2 Limitations & Future Work

The key limitation of the work presented here is scope. Though the detailed iOS review in Chapter 2 and the graphic score review in Chapter 4 covered a large number of items, each of them would have been improved by including more apps or scores. Likewise, the high-level iOS review would benefit from a much more detailed examination of each type

of application. These improvements would, in turn, lead to more abstractions in Chapter 5, and make the resulting software deeper and more powerful.

Even more so, limiting the review to smartphone / tablet applications instills a massive bias towards the mappings and media used in Western music. As Ashley [37] has alluded to, there are many other ways of projecting music into two dimensions. Though this thesis has touched upon this by mentioning the Kalimba, this in no way ameliorates the implicit bias in the data. An additional limitation of using the iOS App Store as a source for data is that new applications are constantly being added. The data that Chapter 2 is based on was gathered in early 2013, and Chapter 3 is based on data from early 2014. Much water has passed under the bridge in the world of music technology since then.

In addition to expanding the scope of the data used, a major step would be to perform psychological studies on the various mappings presented here. Showing an abstracted layout to a large number of users and asking them for the 'best' mapping for it would provide an important counterpoint to the empirical, review-based methods of this thesis. Investigations of more specific parameters would also be useful, examining the impact of visual parameters such as button size, shape, colour, and rotation on how users define mappings. This may also turn up useful data on how musical parameters other than pitch are mapped.

The improved data from widening the review or from grounding the review with user studies could be used to improve the machine learning algorithm that the classification API uses. The space for improvement here is large: training the model on more abstractions, delivering more robust results, or dealing with parameters other than button location. Along these lines, many of the objects that can be created in applications such as TouchOSC and Lemur are not buttons: faders, knobs, and X/Y pads abound. Including these in the classification API would be a key step. The classification API could also be broadened to deal with multiple sets of buttons, rather than assuming that every button is part of the same control layout. Likewise, if more research on musical parameters other than pitch is done, the results could be incorporated into the output of the classification API.

Despite the limitations and technical improvement discussed above, the underlying goal of automatic mapping of control layouts has been achieved. Even if the use of the algorithm discussed here does not become common practice, mapping remains a key step in the creation of any interface. The various ideas and tools discussed in this thesis will hopefully shine a light on the subtle process of mapping for designers, engineers, and end users.

References

[1] T. Kell and M. M. Wanderley, "A quantitative review of mappings in musical ios applications," in *Proceedings of the Sound and Music Computer Conference 2013*, pp. 473–480, 2013.

[2] T. Kell and M. M. Wanderley, "A high-level review of mappings in musical ios applications," in *Proceedings of the Sound and Music Computer Conference 2014*, pp. 565–572, 2014.

[3] J. Chadabe, "Electric sound:{The} past and promise of electronic music," 1997.

[4] B. Reed, "2014 looks like the year when smartphones finally crush PCs." `http://bgr.com/2013/12/10/pc-versus-smartphone-install-base/`, 12 2013. Accessed: 06/05/2014.

[5] E. van Buskirk, "Developer Explains Why Android Sucks for Some Audio App." `http://evolver.fm/2012/05/23/developer-explains-why-android-sucks-for-some-audio-apps/`, 05 2012. Accessed: 24/02/2013.

[6] N. Ingram, "Apple announces 1 million apps in the App Store, more than 1 billion songs played on iTunes radio." `http://www.theverge.com/2013/10/22/4866302/apple-announces-1-million-apps-in-the-app-store/`, 10 2013. Accessed: 06/05/2014.

[7] A. Hunt, M. M. Wanderley, and M. Paradis, "The importance of parameter mapping in electronic instrument design," *Journal of New Music Research*, vol. 32, no. 4, pp. 429–440, 2003.

[8] J. B. Rovan, M. M. Wanderley, S. Dubnov, and P. Depalle, "Instrumental gestural mapping strategies as expressivity determinants in computer music performance," in *Proceedings of Kansei-The Technology of Emotion Workshop*, pp. 3–4, Citeseer, 1997.

[9] I. Bowler, A. Purvis, P. Manning, and N. Bailey, "On mapping n articulation onto m synthesiser-control parameters," in *Proceedings of the International. Computer Music Conference*, pp. 181–184, 1990.

[10] A. Hunt, M. Wanderley, and R. Kirk, "Towards a model for instrumental mapping in expert musical interaction," in *Proceedings of the 2000 International Computer Music Conference*, pp. 209–212, 2000.

[11] S. Fels, A. Gadd, and A. Mulder, "Mapping transparency through metaphor: towards more expressive musical instruments," *Organised Sound*, vol. 7, no. 2, pp. 109–126, 2002.

[12] D. Wessel and M. Wright, "Problems and prospects for intimate musical control of computers," *Computer Music Journal*, vol. 26, no. 3, pp. 11–22, 2002.

[13] P. McGlynn, V. Lazzarini, G. Delap, and X. Chen, "Recontextualizing the multi-touch surface,"

[14] Apple, "Apple Updates iOS to 6.1." `http://www.apple.com/pr/library/2013/01/28Apple-Updates-iOS-to-6-1.html`, 01 2013. Accessed: 24/02/2013.

[15] A. Professional, "Akai MPC Series." `http://www.akaipro.com/category/mpc-series`, 01 2014. Accessed: 21/07/2014.

[16] N. F. Arner, "Investigation of the use of Multi-Touch Gestures in Music Interaction," Master's thesis, University of York, 2013.

[17] A. Tanaka, A. Parkinson, Z. Settel, and K. Tahiroglu, "Survey and thematic analysis approach as input to the design of mobile music guis," *Proceedings of the International Conference on New Interfaces for Musical Expression*, 2012.

[18] H. Zhu, H. Cao, E. Chen, H. Xiong, and J. Tian, "Exploiting enriched contextual information for mobile app classification," in *Proceedings of the 21st ACM international conference on Information and knowledge management*, pp. 1617–1621, ACM, 2012.

[19] M. Chen and X. Liu, "Predicting popularity of online distributed applications: itunes app store case analysis," in *Proceedings of the 2011 iConference*, pp. 661–663, ACM, 2011.

[20] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[21] H. Zhang, "The optimality of naive bayes," in *Proceedings of the FLAIRS Conference*, vol. 1, pp. 3–9, 2004.

[22] B. Whitman and S. Lawrence, "Inferring descriptions and similarity for music from community metadata," in *Proceedings of the 2002 International Computer Music Conference*, pp. 591–598, Citeseer, 2002.

[23] J. Cage, *Notations*. Something Else Press New York, NY, 1969.

[24] G. Read, *Music notation: a manual of modern practice*. Taplinger Publishing Company, 1979.

[25] K. Stone, *Music notation in the twentieth century: a practical guidebook*. WW Norton New York; London, 1980.

[26] E. Brown, *Folio and Four Systems*. Associated Music Publishers, 1961.

[27] E. Brown, "On december 1952," *American Music*, pp. 1–12, 2008.

[28] T. Sauer, *Notations 21*. Mark Batty Pub, 2009.

[29] E. Karkoschka and R. Koenig, *Notation in new music: A critical guide to interpretation and realisation*. Universal Edition London, 1972.

[30] G. Lock, ""what i call a sound": Anthony braxton's synaesthetic ideal and notations for improvisers," *Critical Studies in Improvisation/Études critiques en improvisation*, vol. 4, no. 1, 2008.

[31] V. Anderson, ""well, it's a vertebrate...": Performer choice in cardew's treatise," *Journal of Musicological Research*, vol. 25, no. 3-4, pp. 291–317, 2006.

[32] A. Braxton, *Composition notes*, vol. 4. Synthesis Music, 1988.

[33] G. Read, *Source book of proposed music notation reforms*. Greenwood Press New York, NY, 1987.

[34] E. R. Tufte and P. Graves-Morris, *The visual display of quantitative information*, vol. 2. Graphics press Cheshire, CT, 1983.

[35] D. Van Nort, M. M. Wanderley, and P. Depalle, "On the choice of mappings based on geometric properties," in *Proceedings of the 2004 conference on New interfaces for musical expression*, Shizuoka University of Art and Culture, 2004.

[36] D. Tymoczko, *A geometry of music: harmony and counterpoint in the extended common practice*. Oxford University Press, 2011.

[37] R. Ashley, "Musical pitch space across modalities: Spatial and other mappings through language and culture," in *Proceedings of the 8th International Conference on Music Perception and Cognition*, pp. 64–71, Causal Productions Adelaide, Australia, 2004.

[38] C. Dahlhaus *et al.*, "Harmony," in *The New Grove Dictionary of Music and Musicians* (S. Sadie, ed.), vol. 29, London: Macmillian, 2001.

[39] L. Baxter, *Capacitive sensors design and applications*. IEEE Press, 1997.

[40] R. Cohn, "Neo-riemannian operations, parsimonious trichords, and their tonnetz representations," *Journal of Music Theory*, pp. 1–66, 1997.

[41] B. Park and D. Gerhard, "Rainboard and musix: Building dynamic isomorphic interfaces," in *Proceedings of the 13th international conference on New interfaces for musical expression, Daejeon, Korea Republic*, 2013.

[42] B. Crabtree and K. Cain, "Monome." `http://monome.org`, 01 2014. Accessed: 21/07/2014.

[43] Y. Nishibori and T. Iwai, "Tenori-on," in *Proceedings of the 2006 conference on New interfaces for musical expression*, pp. 172–175, IRCAM—Centre Pompidou, 2006.

[44] V. S. Ramachandran and E. M. Hubbard, "Synaesthesia–a window into perception, thought and language," *Journal of consciousness studies*, vol. 8, no. 12, pp. 3–34, 2001.

[45] H. Lobby and Y. De Ridder, "Beatsurfing - An iPad Organic MIDI Controller Builder.." `http://beatsurfing.net/about/`, 03 2013. Accessed: 22/07/2013.

[46] Hexler, "TouchOSC - Modular OSC and MIDI control surface for iPhone / iPod Touch / iPad." `http://hexler.net/software/touchosc`, 03 2013. Accessed: 21/07/2013.

[47] Liine, "Lemur overview." `https://liine.net/en/products/lemur/`, 03 2014. Accessed: 12/05/2014.

[48] J. Malloch, S. Sinclair, and M. M. Wanderley, "Libmapper: A library for connecting things," in *Extended Abstracts on Human Factors in Computing Systems*, pp. 3087–3090, 2013.

[49] J. Malloch, S. Sinclair, and M. Schumacher, "Digital orchestra toolbox." `http://idmil.org/software/digital_orchestra_toolbox`, 07 2013. Accessed: 12/05/2014.

[50] F. Bevilacqua, R. Müller, and N. Schnell, "Mnm: a max/msp mapping toolbox," in *Proceedings of the 2005 conference on New interfaces for musical expression*, pp. 85–88, University of British Columbia, 2005.

[51] R. Fiebrink, *Real-time Human Interaction with Supervised Learning Algorithms for Music Composition and Performance*. PhD thesis, Princeton University, Princeton, NJ, USA, January 2011.

[52] G. Essl, *UrMus-an environment for mobile instrument design and performance*. Ann Arbor, MI: MPublishing, University of Michigan Library, 2010.

[53] C. Roberts, *Control: Software for end-user interface programming and interactive performance*. Ann Arbor, MI: MPublishing, University of Michigan Library, 2011.

[54] P. Joguet and G. Largillier, "Controller involving manipulation of virtual objects on a multi-contact touch screen," Sept. 30 2005. WO Patent 2,005,091,104.

[55] Cycling '74, "Mira." `http://cycling74.com/products/mira/`, 09 2013. Accessed: 12/05/2014.

[56] R. T. Fielding, *Architectural styles and the design of network-based software architectures*. PhD thesis, University of California, 2000.

[57] S. Memo, "Sympathetic synthesizer." `https://github.com/stuartmemo/sympathetic-synth`, 04 2014. Accessed: 12/05/2014.