Graphical Performance Software in Contexts: Explorations with *Different Strokes*

Mark Zadel

Music Technology Area, Department of Music Research
Schulich School of Music
McGill University
Montreal, Quebec, Canada

August 2012

A thesis submitted to McGill University in partial fulfillment of the requirements of the degree of Doctor of Philosophy

Abstract

This thesis proposes a novel approach to musical software analysis that prescribes testing a given software interface in a wide variety of hardware contexts, each providing unique insights into its design. This work is situated in the general context of *graphical software performance*, which we define as musical performance through manipulating an on-screen software interface to create music. The analysis strategy is investigated using the *Different Strokes* (DS) performance environment as a specific example. A series of software extensions to DS were undertaken to extend the application to new hardware contexts and use cases. These include extensions for the use of *Different Strokes* in an interdisciplinary performance work, *d_verse*; the adaptation of DS to work on a large multi-touch surface; the integration of a force-feedback input device; and the integration of the *libmapper* framework, allowing it to be easily interconnected with alternative input and output devices. The thesis also presents a historical overview of graphical software intended for live use, and a background on general issues in interface design for this usage context. An exploratory user test was performed with the force-feedback setup where participants used DS in the presence of simulated physical forces. While there was no clear preference for any of the haptic effects, the different physical forces present are demonstrated to have gestural implications. These kinds of implications should be taken into account when designing mappings from gesture to sound, and in the overall interaction design.

Résumé

Cette thèse propose une nouvelle stratégie d'analyse de logiciels pour la musique par l'utilisation d'interfaces graphiques dans divers contextes matériels informatiques. Notre travail s'inscrit dans la perspective globale de « graphical software performance » définie ici comme la manipulation d'interfaces utilisateur lors de performances musicales. La stratégie d'analyse que nous proposons a été explorée à partir de *Different Strokes* (DS), un logiciel conçu pour la création musicale. Pour permettre l'utilisation de DS dans plusieurs contextes matériels et contextes d'utilisation nouveaux, nous avons développé un ensemble d'extensions. Ces extensions ont été élaborées, entre autres, pour l'utilisation de DS dans la série de performances interdisciplinaires *d_verse*. Nous avons également adapté DS pour qu'il soit compatible avec une surface « multi-touch » de grande taille, nous y avons intégré une interface haptique, et nous y avons ajouté *libmapper*, une bibliothèque logicielle facilitant l'interconnectivité d'interfaces d'acquisition et de restitution. Cette thèse présente, en outre, un survol de l'histoire des logiciels graphiques conçus pour l'usage en direct et une discussion des problèmes liés à la conception d'interfaces utilisateur dans ce contexte précis d'utilisation. Un test exploratoire a été effectué avec le système haptique intégré à DS. Lors du test, les participants ont utilisé *Different Strokes* en présence de différentes forces physiques simulées. Bien que nous n'ayons pas observé de préférence claire pour l'un ou l'autre des effets haptiques, nous démontrons que les différentes forces physiques présentes influencent néanmoins la gestuelle. Nous concluons que leur influence doit être prise en considération dans la conception de mappings entre le geste et le son et dans la conception d'interaction en général.

Acknowledgements

First and foremost, thank you to my advisors, Marcelo Wanderley and Gary Scavone. They were exceptionally supportive over the course of this work and were fantastic mentors. Sincere thanks to my collaborators: Stephen Sinclair (with whom I worked on the haptics portion of the work), Bruno Angeles (multi-touch port) and Joseph Malloch (*libmapper* implementation design). Thanks to my good friends in the IDMIL and the CAML in the Music Technology area at McGill, for being generally inspiring and for the frequent coffee breaks. In particular, thanks to Avrum Hollinger, Stephen Sinclair, Joseph Malloch and Andrey da Silva. Thanks to the Natural Sciences and Engineering Research Council of Canada (NSERC) and to Hexagram-Concordia for funding over the course of this research. Finally, thanks to my family—Joe, Veronica, Lydia and Andrew—for their incredible support and patience, without which I would surely never have been able to complete this work.

Contents

List of Figures

Chapter 1

Introduction

This thesis is concerned with the use of computers in live musical performance. There are a variety of ways that this can be approached, including real-time audio synthesis and processing, the use of autonomous systems that listen and respond to a human performer, and sound control through novel, custom-designed physical interfaces. The particular focus of this thesis is the live, on-stage operation of a graphical user interface to perform music.

Laptops have been used for this style of performance for a number of years. However, the recent commercial popularity and wide proliferation of portable touch-screen devices, such as smartphones and tablet computers, has bolstered efforts around the use of graphical interfaces for creative applications, including music.

In spite of this popularity, music performance software is generally designed heuristically. Traditional Human-Computer Interaction (HCI) approaches to interface design and evaluation involve easily quantifiable performance metrics, like the time taken to acquire an on-screen target, or how many mouse clicks it takes to find a menu item. These do not adequately address the musical control context—involving real-time use, expressivity, and multiparametric control (Wanderley and Orio, 2002).

This thesis proposes a novel approach to the evaluation of software performance interfaces for music. It involves testing the application in question in a variety of different hardware contexts to learn general lessons about its design.

1.1 Context: Software-Oriented Music-Making and Performance

Software-oriented music-making is now common, and a culture exists of musicians who make music exclusively on their laptops. This style of creation is studio-centric, in the sense that it is composed using primarily studio manipulations of sound material, and is not necessarily recorded live. A composer builds up a composition iteratively, in layers, often making edits using non-real-time methodologies.

One consequence of this studio-centric model is that it is not inherently performative, and there is no natural transposition of the studio composition activity to the stage (Zadel, 2006, p. 2). The music is performed nonetheless, and as it is created using graphical interfaces, graphical interfaces are also often used live in front of an audience.

The underlying impetus behind the *Different Strokes* application was to address the performance question (Zadel, 2006, p. 3; Zadel and Scavone, 2006b). *Different Strokes* (DS) is a graphical music performance application based on drawing gestures. It is an attempt to bring "liveness" to a software interface, striking a balance between performer action and computer control. Its design aims to do sequencing in a novel, visual way that lends itself to on-stage improvisation and to simultaneously appeal to an audience. Projecting the application's screen serves to amplify some of the performer's otherwise imperceptible gestures, rendering her gestures explicit.

1.2 The Influence of the Hardware Context

Graphical software instruments are uniquely positioned in that they are defined entirely in code, and can generally only make assumptions about the hardware they are being run on. They might depend on a certain operating system, or that a pointing device is available, or that there is a screen, but those assumptions can often be met by a variety of devices. We could talk about computer hardware as being software-agnostic: the hardware that runs screen-oriented software is designed to run many different software applications, and so has to be relatively generic to accommodate the needs of all of them. Other digital musical instruments may have a custom, fixed physical interface, but graphical software interfaces are typically designed from the perspective of being used on commodity hardware.

Conversely, graphical interfaces can also be thought to be hardware-agnostic, in that

they can be run in a variety of hardware contexts. Hardware differences can however lead to differences in the experience and capabilities of the performer. Gesturally, tablet computers perform differently than traditional computer mice or other controllers: there are gestural implications to their form factors, and the fact that the user controls the interface with their fingers instead of a stylus or mouse. Consider, too, the difference between a graphical interface being projected, versus being seen on a desktop monitor, versus being seen on a smartphone. For developers who want to create musical applications, where gesture is of the utmost importance, it is important to be aware of the differences in control and overall experience that the hardware context engenders. Musical control gestures require precision and subtlety, and that control can be strongly influenced by the physical properties of the input device. How does the software work differently when using novel controllers, such as graphic tablets or touch surfaces?

We can go further and consider the implications of exploring a variety of hardware contexts in the interface design process. By testing the software in varied conditions, each can shed light on different aspects of the design, bringing new insights and suggesting potential improvements that could benefit the design overall.

1.3 Dissertation Project

This dissertation proposes a novel approach to the analysis of graphical software applications for musical performance. The approach prescribes testing a given software interface in a wide variety of hardware contexts, each providing unique insights into the interface design.

As a specific example of this approach, a set of feature extensions to *Different Strokes* were implemented that allow it to be used with a variety of new hardware devices. User testing was done on two of these setups, and DS was also used as part of an artistic performance piece as a real-world case study. The goal was to explore how the input hardware can influence the user's experience with the software.

The specific software extensions that were added are:

- extensions to the system to meet the needs of a real-world performance piece, including multi-channel audio output, MIDI support and robustness improvements;
- the adaptation of the software to be used on a large multi-touch table;

- integration with a haptics software framework to allow it to be used with force-feedback input devices; and
- integration with a mapping library to allow artists and practitioners to easily connect alternate input hardware and output synthesis modules.

1.4 Thesis Structure

This thesis is structured as follows:

Chapter 2, "Overview of Music Performance Interface Developments," surveys the developments in the field of performance interfaces, describing the hardware and software ensembles that are used, and some aspects of the performance culture that have arisen around this practice.

Chapter 3, "Interface Design Considerations," raises some of the issues that should be considered when designing graphical interfaces for live use, including perceptual and cognitive considerations, and lessons that we can draw from other fields.

Chapter 4, "*Different Strokes* Application Development," presents the *Different Strokes* project itself, along with two sets of feature extensions that were made to it. The projects described are a multi-disciplinary performance piece that DS was used in, and the creation of a multi-touch version of DS.

Chapter 5, "Exploratory Study Using Haptic Feedback," describes an extension to DS to allow it to be controlled by force-feedback input devices. This improvement is then used in user testing to compare participants' impressions of a variety of haptic effects.

Chapter 6, "*libmapper* Integration," presents the *libmapper* project, and describes its integration with DS. The benefits of this extension are then demonstrated through an exploratory test.

Chapter 7 concludes the thesis, summarizing its main contributions and describing future work.

Chapter 2

Overview of Music Performance Interface Developments

In this chapter, we examine the hardware and software that are used in laptop performance systems in general, from historical precedents to contemporary systems. This chapter is concerned with the overall problem of designing interactive software systems for music; in later chapters, we will demonstrate the use of varied hardware platforms in graphical interface design.

We outline the development of performance software designs, which parallels computers' increasing real time capabilities. We consider some elements of performance practice that have arisen to address issues endemic to laptop performance. We finally survey the hardware devices that are available for interacting with these screen-oriented performance applications. These input devices give an idea of the kinds of differences we can expect when testing our software designs across hardware platforms.

2.1 Graphical Software Performance

Graphical software performance interfaces are operated in the usual way through standard computer input and output devices (trackpads, mice, keyboards and computer screens). This equipment is generic and ubiquitous, and software is extremely easy to distribute and obtain. The main differentiating factor between the instruments is therefore the design of the graphical interface and the underlying software logic. This perfor-

mance setup is popular because there is a low barrier to entry: not only is the technology accessible, but the producer/performer does not specifically need musical training to be able to make coherent-sounding music. The computer can take care of most of the timing and pitch issues, so the producer primarily needs patience and a good ear when composing and subsequently performing.

A common trait of graphical software performance interfaces is that they offer the performer very high-level control of the musical material, on the macro level of sequences and note patterns. On the sound processing side, the interfaces typically allow scalar parameters for synthesis and effects to be manipulated in real-time, either through on-screen faders and knobs, or in other ways such as mapping from the current mouse position. With the introduction of low-cost USB MIDI controllers, these setups are often augmented with MIDI-style fader boxes or compact piano keyboards.

This style of computer music performance is sometimes called "laptop performance" (Collins, 2003; Cascone, 2003; Irving, 2006) due to the ubiquity of the laptop in this context, but a laptop is definitely not required. We can substitute the more generic phrase *graphical software performance.*

2.2 History

2.2.1 Precedent

Performing computer music with high-level control has been possible since the late 1960s. Mathews and Moore's GROOVE program (1970) is an oft-cited early example of a computer-based interactive performance system. GROOVE was a computer system that could generate and record scalar "functions of time" and use these to drive analogue synthesis units. The curves could be predefined by mathematical expressions in the program code, or recorded in real-time by manipulating physical knobs. These physical signals could also serve as input to the mathematical functions to generate more complex control effects. The curves could be replayed, manipulated and re-recorded in another pass, iteratively building up parallel control curves. The curves could be rendered graphically using an oscilloscope display. The system also included a 3D wand (possibly a precursor to Mathews' Radio Baton (Boulanger and Mathews, 1997)) and an array of switches that the operator could control. The audio synthesis was done with

analogue synthesizers, but the recording and manipulating of control curves was done digitally on the computer.

The relationship between the system operator and the music foreshadows similar philosophies that appear later in the evolution of computer music performance, as we shall see:

> Further thought convinced us that the desired relation between the performer and the computer is not that between the player and his instrument, but rather that between the conductor and the orchestra. The conductor does not personally play every note in the score; instead he influences (hopefully controls) the way in which the instrumentalists play the notes. The computer performer should not attempt to define the entire sound in real-time. Instead, the computer should have a score and the performer should influence the way in which the score is played. (Mathews and Moore, 1970, p. 716)

This is not graphical performance control, but it is a significant achievement and sets the stage for later developments.

2.2.2 MIDI Era

By the mid to late 1980s, computers had become fast enough to sequence MIDI in real-time. Coupled with an external synthesizer, these systems could be used to perform music via graphical interfaces.

Music Mouse (Spiegel, 1995) is one of the first interactive graphical performance programs, created by Laurie Spiegel and first released in 1986. It allows the user to generate musical patterns by moving the mouse and using the computer's QWERTY keyboard. The main idea behind the program was to leverage the computer's logical capabilities to create a software instrument that would collaborate with the user in real-time. The main screen consists of a 2D grid where the mouse can be moved to create gestures in different parameters, driving the program's compositional algorithms. Harmony is handled automatically by the program, subject to parametrization by the user. The intention was for the program to be quick and interactive, allowing a single user to make interesting and complex music through the graphical interface.

Music Mouse was intended to be used by non-expert musicians to create entire pieces of music in real-time. The program is thus charged with playing sequences procedurally

while the user manipulates the algorithms' parameters. Scales and harmony are built into the program, so the application itself takes care of "sounding good." The notes are output via MIDI to either an external sound module or the internal sound source on the Macintosh.

Spiegel considered the internal logic of the program to be a major factor in its musical output. Indeed, the musical assumptions and patterns in the system shape the musical output in large part, and thus Spiegel believed that the program should be credited at least partially for the resulting compositions. This is interesting in comparison to today's software climate, as algorithms to transform sound are taken for granted.

The programs *M* and *Jam Factory* (Zicarelli, 1987) were created by David Zicarelli and Joel Chadabe, initially developed beginning in 1986. They were examples of what Chadabe calls "interactive composing" (Chadabe, 1984), where the computer algorithmically transforms musical material under the direction of a human user. He likened the process to controlling an airplane, where the composer acts as a pilot to direct the composition, but the musical processes are ultimately driven by their own power. Part of Chadabe's idea is for the computer and composer to exert mutual influence on each other, and that the generative algorithms should have some aleatoric elements to foster a sense of surprise. This interaction between computer and composer leads to a challenging compositional experience.

The main M screen is a collection of widgets that the performer/composer can use to parameterize its processes. The controls allow the user to specify ranges and discrete option values that shape the output sequences. Parameters such as duration, pitch, accents, density, distributions and patterns can be influenced and controlled. Parameter sets can be saved as snapshots, and these can be called up to change all of the parameters at once.

Jam Factory attempts to make the computer a partner in improvisation by transforming incoming MIDI notes according to Markov chains. While the interface is not played solely using the mouse, graphical interaction makes up an important part of the performance and the program is therefore included in the present discussion. The transition probabilities are controlled through the GUI while the performer is playing, iteratively adjusting the transition probabilities to generate appropriate musical effects. The program includes four independent processes, each of which corresponds to a synthesis voice, that act simultaneously on the source material. The ultimate sound output is

rendered internally or externally via MIDI control.

The original *Max* system (Puckette, 1988) was designed for real-time, interactive MIDI processing. It was first released commercially in 1990. The interface consists of module boxes that are virtually connected together on-screen. Data (typically MIDI sequences) flow through this graph and are transformed at each node. Interactive widgets can be included in the graph, and these can be manipulated in real-time using the mouse. In this way, Max can be used to interactively control synthesis devices in performance.

2.2.3 Pervasive DSP Era

The arrival of affordable, general-purpose computers that were fast enough to perform real-time audio signal processing opened up the possibilities of using the computer on its own to create music in real-time. This approach started becoming practically useful "by 1996 or so" (Puckette, 2002, p. 35). The existing advances in graphical, direct manipulation interfaces could finally be applied to real-time music-making.

2.2.3.1 Max/MSP

An important turning point in this contemporary period was the introduction of Pure Data (Puckette, 1996) and the subsequent integration of DSP capabilities in Max with the addition of MSP. Max/MSP is the canonical example of graphical performance software. Now a single computer, typically a laptop, could act as an entire real-time experimental music studio. It enjoyed (and continues to enjoy) considerable popularity. The software allows an artist to iteratively and interactively build synthesis and sound-processing algorithms, and then use these algorithms in the same way on-stage. Again, we tend to see a notion of high-level control in Max patches, where the performer is directing generative procedures by manipulating their parameters in real-time. Puckette (2002) provides a good historical overview of the development of the Max/MSP platform.

2.2.3.2 FMOL

Sergi Jordà's *FMOL* program (2002) is another notable early entry in this category. It was created over 1997–1998 for use in a contemporary theatre production. It provides a graphical interface for controlling a variety of synthesis and processing algorithms,

whose interconnections are established by the user when the program is first started. The interface consists of a series of vertical "strings" which are virtually plucked and manipulated by the user via the mouse. The strings are animated with a sine-wave-like vibration when they are activated, making the character of the interface quite dynamic and active. Each string column represents a signal chain, and its activity is mapped to synthesis control parameters. The interface is designed to reflect the program's state in a "symbolic and non-technical way" (Jordà, 2002, p. 31).

Jordà argues that this kind of "mouse music" is still worth investigating, even though much research is directed toward designing hardware control devices (p. 25). He also argues that the ubiquity of the mouse as an input device serves to make music-making more "democratic."

2.2.3.3 Levin's work

Golan Levin's *Audiovisual Environment Suite* (AVES) (2000) is a fantastic example of the creative ways in which software can be used for interactive media performance. It consists of seven software systems that allow a performer to simultaneously create and manipulate abstract sound and visuals gesturally. There are no widgets on the screen to speak of; rather, the user's mouse gestures act to create dynamic "paintings." This model is a strong, aesthetically compelling departure from previous direct-manipulation interfaces. The AVES applications were designed to be used with standard computer input devices, and optionally, a graphic tablet.

The earliest system in the group is called *Yellowtail.* The user makes marks on the canvas, and each time the pen is lifted the current mark comes to life and wriggles around the screen. The temporal dynamics of the gesture inform the animation, allowing the user to elicit different kinds of motions. The canvas is designed with a toroidal topology, so marks that leave one side of the screen will reenter via the opposite side. The animation is sonified using an inverse spectrogram: a horizontal current-time line sweeps repeatedly from the bottom of the screen to the top, and any intersecting visual elements control the volume of a bank of sinusoidal oscillators. In this manner, the vertical axis is associated with time, and the horizontal axis is associated with pitch.

Another work, *Loom,* takes moving gestural marks and uses their properties to drive an FM synthesizer. Each mark is periodically redrawn according to the temporal dynam-

ics of the drawing gesture, and the local properties of the curve at its endpoint (velocity, pressure and curvature) are mapped to the sound control. There is one synthesis voice per stroke and the strokes redraw themselves repeatedly, creating sonic event patterns with complex phase relationships.

2.2.3.4 Ableton Live and Reason

Ableton Live (2012) is arguably the most popular graphical performance software available, first released in 2001. It was specifically designed from the outset to support live performance for loop-oriented musicians.

Performers pre-load the system with audio loops ("clips"), which are triggered, mixed and processed in performance. These can be grouped into rows in Live's performance interface so that related clips (for example, everything that is supposed to play for the song's chorus) can be triggered all at once. Performance with Live consists of controlling clip playback and manipulating effects parameters in real-time. Because clips can be synchronized easily to the downbeat, it is possible to always keep the music totally coordinated, regardless of the number of simultaneous voices.

The graphic interface consists of a control-panel style layout, with widgets for starting and stopping clips, scrolling through the grid of loops, adjusting volume levels, and modifying effects parameters. There is also a waveform view for indicating time warping points. The on-screen knobs, faders and buttons can be assigned to an external MIDI controller for easier access. This is a strategy to cope with having to manipulate many controls, since doing it exclusively via the mouse can be physically difficult.

Propellerhead Software's *Reason* (2012) targets a similar demographic of musicians as Live. It is billed as a self-contained software studio. The software mimics analogue rack-mount audio devices that are typically found in physical recording studios. Synthesizers, mixers, drum machines and effects units are included, and their on-screen representations are renderings of what the real, physical devices would look like. Their parameters are adjusted by manipulating the virtual knobs onscreen, and they are interconnected using realistic-looking "patch cords" at the "back" of the rack. This software is popular for production as it mimics a classic analogue timbre and style of working.

Reason contains analogue-style step-sequencing modules that can be used to drive the various synths. The software can be used in performance in this way, by mixing,

matching and triggering different sequences. Collections of synths can be combined into individual racks that can be switched between quickly in performance. This feature is typically used on-stage to call up different patches for different songs.

2.2.3.5 Other Applications

Many other modular and dataflow applications are available for sound production, and they are operated in performance in largely the same way as Max. For example, *Reaktor* (Native Instruments, 2012), *Bidule* (Plogue, 2012) and *AudioMulch* (Bencina, 2012) work in basically the same way: generation and processing modules are patched together, and then parameters are manipulated in real-time via on-screen widgets. Sometimes modules are also wired together *as* performance, which is reminiscent of the live coding approach (Collins, 2003). For example, Alexandre Burton created a Max/MSP patch from scratch as his solo performance at the Mutek 2002 festival in Montreal. SuperCollider (McCartney, 1998) is also a real-time, interactive, modular audio application. Instead of patches being created graphically, however, they are constructed textually. Graphical interfaces can then be created to parameterize and trigger the audio processes in performance. SuperCollider is also amenable to live coding due to its interactive and textual foundations.

The works of ixi software are interesting examples of the possibilities of screen-based graphical software instruments (Magnusson, 2006a,b). They create interactive, symbolic, dynamic interfaces where performers can alter the topology and properties of the on-screen agents to effect different control sequences and parameter changes. They do not feature typical control-panel widgets as such, but instead favour abstract, active objects that interact with one another. The interfaces do not specify the synthesis method, but rather send their control data out via OpenSound Control for synthesis in external software. (This is interesting, since it mimics the older MIDI style of working with external synthesis, but in this case sequencing and synthesis are done in software on a single computer, communicating via network loopback sockets.) The *SpinDrum* interface is a good example of ixi's work. Rotating wheels of boxes move around the screen, each acting as a circular step sequencer. Boxes can be activated with the mouse, and when an activated box reaches the twelve o'clock position it sounds the sample associated with that wheel. The animated wheels can have different numbers of "petals" (boxes) and

can have different periods, leading to interesting phasing patterns between the individual sequences. The *Connector* interface allows the user to create grid-like topologies of connected nodes. An "actor" (coloured square) moves between the nodes according to transition probabilities specified for the graph, and its movement is expressed as control data to drive the synthesizer. ixi offers a whole suite of open-source interfaces in the style of these two examples.

In the vein of Levin's work, "audiovisual instruments" are software instruments that allow the performer to create simultaneous audio and visuals. This is one area where software is well-suited as this would be awkward to achieve with other means. Enrique Franco's Miró system (2004) follows in this tradition. It allows the performance of audiovisual works driven by gestural input from the performer using a graphic tablet. The input gestures are mapped to audiovisual output according to specific processing treatments that are selected by the user at the outset. Franco et al. (2004) have also developed a systematic comparison of existing audiovisual instruments, and have developed a framework theory for designing effective instruments of this type.

The author's *Different Strokes* project (Zadel and Scavone, 2006a,b) is another example of a novel performance interface that is made possible by software. The premise is that the performer uses a graphic tablet to draw strokes on the screen, and these strokes create looping and cascading animated figures. The temporal dynamics of the drawing gestures are preserved in the animation, giving it an organic, non-quantized character. Strokes are associated with preloaded sound samples, which are scrubbed through as strokes are activated in the animation. The intention is to have a novel, graphical sequencing platform that a solo performer can use to build up a piece of music interactively, on-stage. Further, the interface screen can be projected so the audience can see what the performer is doing.

Belgian composer Thierry de Mey's piece *Light Music* (SMCQ, 2009; Szpirglas, 2011) is reminiscent of some of the other works discussed above. Created in 2004 in collaboration with percussionist Jean Geoffroy and engineer Christophe Le Breton, *Light Music* is an audiovisual piece for "conductor/soloist," projection and interactive systems. It features graphical traces on a large overhead screen that match the soloist's hand movements. As the performer's hands are thrust from darkness into light, their image is captured by a video camera and projected. The images subsequently fade out after a short time, producing ephemeral traces of the movements on the screen.

Inertial sensors, namely digital gyroscopes and accelerometers, are attached to the performer's fingers and are used to capture aspects of the motion (Szpirglas, 2011, p. 36). Movement analysis is also performed on the video signal, and specific movements are identified. These data are used to simultaneously control the generation and manipulation of musical material.

While this is not strictly a graphical interface where the interaction happens at a two-dimensional display, this piece bears mentioning because of its use of gestural line in audiovisual performance, as in Levin's work, DS and *d_verse* (as we shall see in Chapter 4), and because of its audiovisual nature.

2.2.4 Applications for Smartphones and Tablet Computers

In recent years, we have seen small-format, multi-touch screens become commonplace, following from the commercial success of Apple's iPod Touch. Their use is quite widespread, and are included in so-called "smartphones" (Internet-enabled cellular phones) and tablet computers, such as Apple's iPad. The commercial success and affordability of these devices has created a rich market for multi-touch music applications.

Many of these music applications mimic the interfaces of existing instruments and music hardware: drum machines, piano keyboards, synthesizers. Other applications attempt to make use of the input hardware in a new way that takes advantage of its multi-touch capabilities and its touch interaction paradigm. Applications such as Gliss (von Falkenstein, 2011) and Thicket (Ott and Packard, 2010) show that these devices offer new interaction-design potential to explore. Most applications are not necessarily intended to be used on-stage (except for perhaps TouchOSC (hexler.net, 2012)), and are rather intended for composition, or as interactive musical toys. (Some interfaces that predate the introduction of ubiquitous touch surfaces, such as the Lemur (JazzMutant, 2008), are designed to be used in performance.)

Going forward, the relative affordability of touch-screen devices, their novelty, and the enthusiasm around their potential will provide a rich basis for innovation in music performance software.

2.3 Interface Design Models

There are various common paradigms underlying graphical performance interface designs. The most common is the control-panel layout, but other models exist. Graphical software interfaces for music can be loosely categorized as follows (Levin, 2000; Franco et al., 2004):

Scores, timelines and diagrams Musical material is represented in a plot with respect to time using markings that represent events. Their positions and shapes are meaningful, representing both the events and their parametrization. Examples include traditional score representations, contemporary graphical scores, step-sequencing interfaces, individual tracks in digital audio workstations (e.g., ProTools), waveform displays, and spectrograms.

Control-Panel Displays These interfaces are designed to mimic real-world hardware device controls, as in mixers and analogue rack-mount units. Virtual knobs, faders, buttons, and meters are arrayed into control panels and are manipulated with the mouse using a direct manipulation metaphor. This is arguably the easiest interface style for new users to understand: the controls mimic their physical counterparts; their operation is culturally ingrained and well understood due to our experience with physical mixers and other similar devices; and the direct manipulation interface metaphor is common in nearly all graphical software (i.e., the ubiquitous WIMP paradigm: windows, icons, menus, and pointer (Wanderley and Orio, 2002)). Examples include most VST modules, Reason, Ableton Live, and Max/MSP.

Dynamic/Interactive Objects These interfaces consist of on-screen objects that act on their own. The user interacts with these objects to manipulate and parameterize them, directing their interaction with each other. Examples include FMOL, Different Strokes, and Electroplankton (Iwai, 2005). Couturier (2006) writes "[i]f graphical interfaces are a good way to control virtual copies of real objects, they can also represent interaction situation that are unrealizable in the real world, like navigation in trees, control of dynamic objects...".

Levin (2000) and Franco et al. (2004) make a distinction between interactive objects and "painterly interfaces" or "drawings and free-form images." They see

interactive objects as dynamic widgets, whereas the material in free-form images is abstract and does not have well-defined object-like boundaries. The visuals and the sound in these painterly interfaces should be tightly synchronized, direct reflections of each other.

Couturier (2006) proposes augmenting this notion of dynamic/interactive objects from an "instrumental interaction" perspective (Beaudouin-Lafon, 2000). This can include adding appropriately-chosen external hardware devices that offer more control bandwidth than a mouse. A mouse offers a single point of contact with a graphical interface, whereas other devices may allow multiple simultaneous control streams.

2.4 Performance Practice

A number of trends and commonalities can be seen in graphical software performance practice, centering around how to manage a complex sound-producing system in real time. Artists are also employing external elements, such as video, to heighten the visible performativity of their stage show.

2.4.1 High-level Process Control

Joel Chadabe's ideas (1984) about "interactive composing" are relevant to graphical software performance. He describes the process as directing musical processes in real-time. The processes can be generative, or they could transform existing musical material. This could be described as computer-aided, human-directed, real-time composition. The computer is essential in this process; by taking on the duties of actually producing the physical sound, this computer-aided approach frees the composer/performer to think more about the high-level structure and direction of the piece. This makes complex music possible for a solo performer where it would not be otherwise.

This approach is reflected in almost all of today's graphical performance software. The performer is not required to play every note; there is always a certain control distance between the performer and the ultimate sound. The processes involved can be note- or DSP-level, generative or transformational. The basic theme of the algorithmic stages can vary somewhat, from the interactive sequencing style of Ableton Live, to a

more "generative processes" style that is often used with Max/MSP.

This strategy obviates the need for the performer to have specific musical physical performance skills, as the computer itself can keep everything synchronized and harmonic (if that is desired). The computer mediation of the sound production can also serve as a safety net, ensuring that no audible mistakes are made on-stage.

2.4.2 Laptop Orchestras and Ensembles

Typical laptop performances consist of one or two performers controlling all of the sound. This is reflective of the fact that computer music composers and producers, both academic and non-academic, tend to create their pieces entirely by themselves. Calling on a solo performer to reproduce these intricate, multi-layered works on-stage is one of the central challenges in graphical software performance. As we have seen, the most common strategy is to use generative and transformative procedures to offload the responsibility of physically producing the live sound to the computer, leaving the performer free to concentrate on the piece's higher-level direction.

A different approach is to do away with the solo performer model and mimic traditional orchestras by involving many computers and performers. The Princeton Laptop Orchestra ("PLOrk") (Trueman et al., 2006; Trueman, 2007; Smallwood et al., 2008; Wang et al., 2008) follows this model, with fifteen performers on stage at once. Each musician has his or her own laptop, operating software as required by the piece. The ensemble is coordinated by one or more conductors. One of the most interesting aspects of the PLOrk model is that each performer has his or her own hemispherical speaker and amplification rig. This is meant to mimic the acoustic properties of traditional instruments where omnidirectional sound sources are distributed throughout the performance space, instead of mixing down all of the laptops' audio to a small number of loudspeakers. The computers may or may not communicate with each other via network messages, depending on the composition. The software systems are often controlled by the standard laptop keyboard-and-trackpad interface, but sometimes external input devices are also used. The PLOrk model has been imported to other places, like Stanford University ("SLOrk") and elsewhere (Harker and Atmadjaja, 2008).

Smaller laptop ensembles exist as well, using a few performers instead of recruiting a whole orchestra's worth of musicians. One of the earliest examples is the Hub, formed

in San Francisco in the mid-eighties (EMF Institute, 2006). Other examples include the European Bridges Ensemble (2012), PowerBooks_UnPlugged (2012) and the Milwaukee Laptop Orchestra (2008). The author was also involved in the sole performance of the Montreal Genetic Laptop Orchestra (Van Nort, 2006). This ensemble consisted of fifteen improvisers, but no conductor.

The ensemble approach to graphical software performance is positive since it frees individuals from excessive physical and musical burden, allowing each performer to manage a more focused aspect of the overall sound. Also, musical performance has always been largely a group activity, and the ensemble configuration helps recapture this aspect. While we generally expect a solo acoustic performer to have a much more minimal sound than a group, we do not necessarily have the same expectation for solo computer performers.

2.4.3 Audience Engagement

The practice of live performance with graphical software became common around the late nineties when technology began to make it possible. Once this performance style had established itself and started to occur more regularly, some audiences and critics began to react negatively (Cascone, 2002, 2003). Its lack of visible physical causality and performativity was challenging to some audiences. "[A]udiences experience the laptop's use as a musical instrument as a violation of the codes of musical performance" (Cascone, 2003, p. 101). Similarly, Stuart writes that, "[w]ith the development of the laptop as a tool for performance has come a cry for the loss of performativity. This loss is perceived in the lack of action and visual and tactile interaction between the performer and their instrument" (Stuart, 2003, p. 59). Indeed, complaining that a laptop performer could just be playing a CD and checking his or her email has become clichéed, but that fact is a testament to how challenging laptop performance was (and is) for some audiences.

The critics' main objection is essentially that operating software as performance is ultimately non-performative. The sound is not created through physical means, and the standard computer input devices that are used tend to minimize visible, physical gesture. Schloss (2003) and Croft (2007) argue that musical performance requires a visible cause and effect relationship between performer action and sound, and that physicality

is an essential component: a perceivable notion of effort is required by audiences. Finally, Schloss believes that performativity is important: "[p]eople who perform should be performers. A computer music concert is not an excuse/opportunity for a computer programmer to finally be on stage" (Schloss, 2003, p. 242).

In response to this criticism, artists have been adding extra elements to their performances to make them more visually interesting and conform more closely to the audience's apparent expectations. In some cases, live instrumental performers are added to augment the computer-generated sound. We see in some of these performances a move back toward the classic "composition for performer and tape" approach. Alternatively, some performers add novel hardware controllers to their setups that require more visually appreciable gestures.

Perhaps the most popular strategy in this vein is for artists to add a simultaneous video projection to augment their performance (Jaeger, 2003, p. 53). This is typically done via software and computer projection, so laptop artists can stay within their preferred medium for visuals as well. In this performance context the visuals can be abstract to match much of what happens in the music. The author's Different Strokes interface also proposes that the software interface itself is projected on-stage. On this point, Stuart writes that"[i]t has been the case that some performers have projected their desktop onto a screen during the performance, allowing the audience the chance to watch how the music is manipulated. While this is informative, it detracts from the sound as the audience shifts from listening to viewing" (Stuart, 2003, p. 65). The proponents of live coding disagree, however, and some make exposing their desktops standard practice in their laptop performances (Collins, 2003, p. 69).

Some laptop performers believe that a change in audience expectations and concert culture is what is needed. d'Escriván (2006) argues that the negative reaction to laptop performance is perhaps a cultural and generational phenomenon, and that new listeners raised on a diet of video games are more open to nearly invisible electronic control. Stuart (2003) suggests that audiences need to shift their focus from visual performativity to aural performativity, to concentrate more on the sound itself. In accordance with this perspective, audiences do seem to be acclimatising progressively to seeing laptops on-stage.

2.5 Survey of Tablet and Touch Surface Input Devices

Complementary to our examination of the software aspect of a performance system, we must also consider the corresponding hardware aspect. In this section, we survey alternatives to computer mice that are designed for interaction on a 2D surface, such as graphic tablets and touch screens. In particular, we examine projects that use drawing tablets in novel ways for musical control. Finally, we consider the differences between standard computer interfaces from a musical control perspective.

2.5.1 Graphic Tablets

Graphic tablets are well-known in the digital arts community for affording natural control in drawing tasks. Graphic tablets typically allow three degrees of freedom (x position, y position, and pen pressure) along with a handful of discrete variables (whether the pen's tip or "eraser" end is being used, and a momentary switch). More expensive designs also sense pen tilt along the x and y dimensions.

Various optional components and capabilities are also available. The devices sometimes include a "puck," which resembles a cordless mouse that is used on the tablet surface. It can be used as a mouse, and some tablet models can also sense the puck's rotation on the surface (Wacom, 2000, p. 41). This can be used, for example, to rotate a virtual drawing page with one's non-dominant hand, while drawing using the stylus with the dominant one. While consumer graphic tablets do not typically include an integrated display, some professional models do (Wacom, 2007).

One advantage of using a graphic tablet is in how it leverages fine motor control; it is operated with the fingers, versus the comparatively coarse control of a mouse. Humans have a long history of using styluses for writing, so their use is well-established. Target acquisition can also be much easier using a pen. Finally, the pen and tablet provide natural, passive haptic feedback to the user (Miranda and Wanderley, 2006, p. 31).

2.5.1.1 Graphic Tablet Application Examples

Numerous works have been proposed that use a graphic tablet for musical control. The dexterity with which a musician can manipulate the pen and the tablet's low cost and availability make it attractive for musical applications. Though this section focuses on

hardware input devices, standard graphic tablets are used to control software in a variety of ways, fundamentally changing the user's experience. In this section we survey some system designs downstream from the hardware controller itself.

Matthew Wright has used graphic tablets to control his musical works for many years, using a variety of mapping strategies (Zbyszyński et al., 2007). For example, he sometimes subdivides the surface of the tablet into regions. When he places his pen on a region, it starts a musical event associated with that region, and the rest of the pen gesture parameterizes the event. The subsequent pen gesture can leave the bounds of the initial region. He also uses scrubbing techniques, where the pen position is mapped to point to locations in a sound file. Other interaction techniques include drag and drop, and dipping (Wessel and Wright, 2002).

Arfib and Dudon (2002) use the tablet for bimanual control in two dimensions. They control a software emulation of a "photosonic instrument." The two-dimensional position of the stylus maps to the instrument's filter parameters, and the puck controls the amplitude and source sound. Each of these is done through a one-to-one mapping from x and y position to parameter values.

Kessous (2002) uses the graphic tablet along with a joystick to control a formant-synthesis singing voice synthesizer. The tablet controls the fundamental frequency of the model arranged in a spiral, similar to the visualization of Shepard tones. This is an interesting way to map from the two-dimensional tablet space onto a one-dimensional (radial) axis. The musician can locate a given pitch within an octave absolutely, but he or she can still increase or decrease the frequency boundlessly by spiralling multiple times.

Doug Van Nort uses the tablet to navigate through a high-dimensional parameter space using an interpolation model (Van Nort et al., 2004; Van Nort and Wanderley, 2007). He maps from 5D space (2D position, 2D tilt, pen pressure) to a high-dimensional space of granular synthesis control parameters. The synthesis parameter presets are embedded in the low-dimensional control space, and various strategies are used to interpolate between them at the intermediary points. Other authors have also proposed similar preset interpolation mappings from a low-dimensional space to a high-dimensional one (Momeni and Wessel, 2003; Bencina, 2005).

The pen and tablet are sometimes used as an improved replacement for a standard computer mouse. The author's *Different Strokes* project (Zadel and Scavone, 2006b) uses the tablet for a direct manipulation task by tying the tablet surface to an interactive

visualization. Strokes are drawn with the tablet, creating periodic animated figures that reflect the timing of the performer's drawing gestures. The animation state is then mapped to synthesis and processing parameters to control sound production. This direct manipulation design differs from the above examples by using the tablet surface as a direct analogue to the two-dimensional screen space and emphasizing interaction with virtual objects. Direct manipulation is a common interaction strategy; computer users are used to interacting with virtual on-screen widgets via the mouse.

Zbyszyński (2008) has proposed an elementary method for musical performance on the graphic tablet, much like standard exercises on traditional musical instruments. This speaks to the usefulness of the tablet as a musical controller.

2.5.2 Touch Surfaces

Another class of surface-like controllers are touch surfaces. Touch surfaces are flat tables or tablets that are used by touching them with one's fingers. These are classified as alternate controllers by Miranda and Wanderley (2006, pp. 30–32). The surfaces may have markings on them to partition the surface into different active areas, or could just be blank. The surface could even have an integrated display, making it a touch screen. The screen display can either be physically integrated into the surface, or could be projected onto the surface. Many touch surfaces can only support a single point of contact at a time (single-touch), but contemporary devices typically support multiple points of contact. These are called multi-touch surfaces.

The advantages of touch surfaces are that the performer can typically use his or her hand as the primary input method instead of using a tool, as with a graphic tablet or mouse, and that they can be multi-touch. Still, some touch devices require that the player's hands are augmented in some way (e.g., Couturier's "Pointing Fingers" interface, described below).

2.5.2.1 Touch Surface Examples

Jazzmutant's Lemur controller (2008) is a multi-touch interface that includes an integrated display screen. It can display a selection of widgets such as faders and bouncing balls, configured by the user depending on his or her application. The unit operates

autonomously, sending OpenSound Control (OSC) messages over ethernet. The advantages of this unit are that it is multi-touch, configurable, and uses OSC.

A class of touch surface examples work via frustrated total internal reflection (FTIR). These surfaces consist of a sheet of transparent material through which a beam of light—usually infrared—is shone from the sheet's edge. The light propagates inside of the plate, but does not exit it due to the material's optical characteristics. When a user touches the surface, its refractive properties at that point are altered and some of the light exits from the surface. This light can then be detected from the back face of the plate using video techniques. Points of contact appear as blobs on the video and can be sensed with simple image processing algorithms. Other properties of the contact points, like pressure, are also senseable (Miranda and Wanderley, 2006, p. 32). This method of implementing multi-touch surfaces has the advantage of being scalable: very large displays can be built cheaply since the surface itself is often only plexiglass. Such large multi-touch surfaces can be used by multiple individuals simultaneously, opening the door for collaborative interaction in the same software space.

The first musical example of the FTIR technique is Eric Johnstone's Rolkey Aspyroyd (Miranda and Wanderley, 2006, p. 32). The light source is a high-power halogen bulb, and the surface consists of plexiglass. The surface display is created from a CRT mounted above the plexiglass, which is reflected through a half-silvered mirror. Similar work was done by Davidson and Han (2006). Infrared light is used, and both the image projector and the detector camera are situated behind the surface. Davidson and Han have demonstrated that a number of novel bimanual interaction techniques are possible using this method, especially because of the device's large size.

Another multi-touch surface is the Tactex MTC pad. It consists of fiber optic sensors distributed throughout a compressible rubber pad. The pad itself measures roughly 3" by 5" and does not have any markings. It has been used in musical controllers by Huott (2002) in his Ski project, where three Tactex pads are mounted to a single wooden frame. Two pads are on the front of the surface and one is on the back, allowing all three pads to be manipulated simultaneously by using both hands.

Jean-Michel Couturier's Pointing Fingers interface (2003) takes another approach to multi-touch sensing with an integrated display. He attaches four Flock of Birds (Ascension, 2007) magnetic 3D position sensors to the user's fingers and translates their positions in space to 2D plane positions near the display screen. The system can sense

when the fingers are touching (i.e., are near to) the screen, and can thus implement multi-touch interaction.

Various methods exist for augmenting regular surfaces to make them touch-sensitive. Crevoisier and Kellum (2008) use a plane of laser light just above a regular table surface. Fingers that come in contact with the surface reflect the light, which is picked up by an overhead camera. This technique is somewhat similar to FTIR, but the advantage is that ordinary surfaces can be augmented in this way. The "time delay of arrival" method uses piezoelectric elements attached to a surface and triangulates acoustic signals generated from tapping on the surface (Crevoisier and Polotti, 2005). This strategy has the advantage of being usable with objects that are not planar.

Standard laptop trackpads are touch surfaces that can be used for musical control by mapping from various input gestures (Fiebrink et al., 2007). Some trackpads are also multi-touch, allowing multi-finger gestures and increased control potential. Trackpads have the advantage of being available on all laptops and being well understood by users. The trackpad surfaces are very small, however, making it impossible to perform larger gestures.

Korg's Kaoss Pad (2007) is an example of a commercial single-touch surface controller that is popular with electronic musicians. It consists of a small, dedicated hardware device whose surface is principally devoted to a 2D touch area. It is used to send two parameters simultaneously from a single finger via MIDI. The device also contains some on-board sampling and synthesis functions.

Bill Buxton and colleagues created a touch-sensitive tablet that was originally demonstrated in a musical application for controlling FM synthesis (Sasaki et al., 1981). It worked via capacitive sensing over a mesh of wires, and could sense the position and pressure of a single point of contact. Further work (Lee et al., 1985) made the surface multi-touch. Cardboard templates were used to partition the surface into regions, each corresponding to a given control parameter (Brown et al., 1990). One advantage of this scheme is that the templates provide tactile feedback to the user and thus the surface can be used without having to look at it.

2.5.2.2 Sensor Surfaces

Some touch surfaces are implemented by embedding discrete sensors in a planar surface that can be manipulated by the user's fingers. Linear FSRs and other flat sensors can be used for this purpose. A number of works exist that use this implementation strategy.

The Buchla Thunder device (Freed, 2008) includes 36 touch- and pressure-sensitive strips on one flat control surface. It is multi-touch, as the sensors are independent from one another. Control data are transmitted via MIDI. The Thunder interface is laid out in such a way that it is easy to manipulate many continuous parameters at once via each hand, one parameter per finger. Another such controller is the Midiman Surface One (Miranda and Wanderley, 2006, p. 33), similar to the Thunder in spirit and design. It extends the existing touch-sensitive controls with an additional matrix of potentiometers. The Surface One controller only reached the prototype stage, and did not make it to market.

While not a digital input device, Michel Waisvisz's Crackle Box (2004) is similar to these control surfaces. Metal contacts are arrayed on the surface of the box, which are wired directly into the box's analogue audio circuits. The capacitance of the strips is changed when a performer touches and manipulates them, affecting the sound. This induces an interesting level of direct control over the device's electrical workings.

2.5.3 PDAs

Another class of devices that support tablet-based interaction are personal digital assistants (PDAs). Originally conceived for organizational uses, they have been co-opted for musical purposes, with various musical applications available for all different platforms. The small form factor of these devices that makes them portable also serves as an interesting challenge to GUI design. PDAs typically use styluses for input, so they could be seen as a subclass of graphic tablets with integrated displays.

Geiger (2006) has showcased various interaction designs for PDAs, using a variety of dynamic gestures for input. He lists four interaction methods: region-based triggering, gesture recognition, border crossing, and continuous parameter control. For example, he presents a virtual guitar application where the strings are drawn vertically on the screen and are plucked by crossing them with the stylus. The pitch is controlled by the stylus's vertical crossing position along the string. He also shows a drum mode where

the screen is divided into regions corresponding to different percussion sounds. The sounds are triggered by touching inside the region. A third application uses the 2D stylus position with respect to the entire screen to control a theremin model.

While not strictly a PDA, the Nintendo DS also uses a stylus for input and has a similar form factor. Toshio Iwai's Electroplankton game (2005) allows the user to interact with animated musical agents that move across the screen through different scenarios. The animation events, such as collision, have corresponding sounds, and the different scenarios each have their own sound design. The user interacts with the on-screen creatures by placing them and directing their paths in the environment. The application is meant to be a fun musical diversion for non-experts and suggests an interesting class of music games that could appeal to a large audience.

Many other software applications for mobile music-making are available for the iPhone and other PDAs. For example, MooCowMusic:Band (2008) is a set of applications that simulate musical instruments that the user can play. The Chocopoolp (2006) application is a miniaturised digital audio workstation for making music on the Palm Pilot.

2.5.4 Smartphones and Tablet Computers

In recent years, multi-touch display surfaces have become nearly ubiquitous, beginning with the introduction of Apple's iPod Touch. The mass-market proliferation of this hardware lowered the prices of touch-screen components, allowing other manufacturers to make their own competing devices. Now many touch-screen devices of varying form factors are available.

With this new glut of relatively affordable touch-screen devices comes new software that takes advantage of its affordances, as was discussed in Section 2.2.4. This development effort also leads to new innovation in terms of interaction design and input gestures.

2.5.5 Tangible Interfaces

"Tangible interfaces" are interactive surfaces that use physical props as interface widgets. Each prop's position and other properties are sensed and communicated to the computer. The surfaces often include a video-projected overlay display that is synchro-

nized with the state of the physical elements. The core idea behind tangible interfaces is that they leverage the user's existing real-world object manipulation skills (Fitzmaurice et al., 1995). Various sensing technologies are used, including RFID and video techniques. An advantage of using a tangible interface is that multiple users can work with the interface simultaneously since the surfaces tend to be large and since many props can be sensed at once. Tangible interfaces fall slightly outside of the realm of tablets and touch surfaces, but they are similar in spirit.

The Audiopad interface employs RFID tags embedded in physical pucks on a tabletop (Patten et al., 2002). The pucks are used to select audio tracks, modify effects settings and select items out of tree-like menus. The reactable (Kaltenbrunner et al., 2004; Jordà et al., 2007) works in a similar way, but the pucks are sensed from below by a video camera. Specially designed markings on bottom of each puck allow them to be identified and their orientation to be recovered. The pucks correspond to audio synthesis and processing nodes that are patched together by proximity. The Jam-O-Drum interface (Blaine and Perkis, 2000) is a large table with six embedded drum pads and an overhead video projector providing the graphical content. A number of collaborative musical games have been developed for the platform. The Music Table project (Berry et al., 2003) uses coloured cards that are sensed by a video camera, each representing different musical phrases and operations. The cards are used to physicalize the various data in the system and to make the interaction more intuitive. An augmented version of the input video signal is shown on a separate computer screen as feedback.

2.5.6 Offline Tablet Use

Graphic tablets have found use in offline applications in addition to real-time ones since they allow natural drawing and gesture input. The SSSP score editing tools, created at the University of Toronto in the 1970s, contains various software and hardware components for scoring music, including a digitizing tablet (Buxton et al., 1979). The pen is used to input musical notes gesturally and efficiently: the stroke's initial vertical staff position determines the pitch of the note, and the shape of the stroke's tail indicates the note's duration. The pen was also used for selecting notes by encircling them. The selection could be further refined by deselecting a sub-region of the selected notes with a second circling gesture.

Iannis Xenakis's UPIC system (Marino et al., 1993) allows composers to draw various 2D graphs that define envelopes, wave tables, and pitch trajectories. The user can further draw pitch and time warping functions to modify previously defined curves. The system renders the output sound immediately, allowing the composer to work interactively and iteratively.

2.5.7 Augmented Graphic Tablets

Some projects use additional sensors in conjunction with a drawing tablet to provide extra degrees of control, in the same way that acoustic instruments can be augmented with sensors. The HandSketch controller is an interesting use of a graphic tablet that follows the augmented instrument theme (D'Alessandro and Dutoit, 2007). It is a standard Wacom graphic tablet augmented with eight FSRs, all reachable by one hand. The instrument is held vertically like an accordion, with one hand controlling the pen and the other manipulating the FSRs. The two-dimensional parameter space on the tablet's surface is curved, with one of the axis following the arc of the user's hand as it pivots at the elbow, and the second axis radiating linearly out along the user's forearm. The three tablet dimensions (two spatial axes and pressure) are mapped to the pitch, loudness and timbre of a vocal synthesizer, while the FSRs are used for discrete control. The force sensors on the back of the instrument select operating modes, and the ones on the front control semitone transposition of the instrument's pitch (or other discrete parameters, depending on the mode).

2.6 Conclusion

This chapter has surveyed the developments in graphical performance software and the corresponding hardware that can be used with these applications. We have also examined practices that have evolved in laptop performance to respond to some of its challenges. This discussion has included a wide variety of software interfaces and input hardware. This underscores the fact that graphical software interfaces for music performance can indeed be hosted on a wide variety of hardware platforms, each with its own properties. In the next chapter, we examine topics in software performance interfaces from the designer's perspective.

Chapter 3

Interface Design Considerations

This chapter outlines various considerations that performance interface designers should take into account when developing their systems. In particular, we begin by characterizing drawing tablets and touch surfaces in the context of musical performance applications. Some perceptual issues in designing interactive software interfaces are then explored, along with lessons that we can learn from the field of Information Visualization. We finally discuss the use of direct manipulation in our interface designs, and of non-visual sensory modalities.

As in the previous chapter, here the focus is on the design of interactive graphical interfaces for performance in general. Case studies that explore the use of varied hardware contexts in design begin in the next chapter.

3.1 Graphic Tablets as Musical Input Devices

Two-dimensional interfaces are common in computer systems. The desktop metaphor mimics the real-world version, computer screens are two-dimensional, and computer mice operate on flat surfaces. In electronic audio hardware we see control-panel interfaces, such as mixers and fader boxes, with arrays of physical controls laid out on flat panels. These layouts give the user a vantage point from which to see the state of the whole system at a glance. We have also recently seen the emergence of many similarly two-dimensional input devices for digital music applications, such as USB MIDI controllers and button grids. And recently, portable, general-purpose touch screens and

tablet computers have become ubiquitous. This section examines two-dimensional input devices for music, focusing in particular on graphic digitizing tablets and touch surfaces. These are related in that the user works with respect to a fixed physical plane, but the device's properties and workings can vary significantly.

The section is structured as follows: first, we discuss some of the interaction design considerations one must make when using these input devices. We then enumerate the mapping patterns that are commonly driven by these devices: spatial mapping, direct manipulation and gestural interaction. Finally, we compare these 2D input devices to standard desktop computer input devices for musical control, as these alternate controllers often aim to improve on the capabilities offered by mice and keyboards.

3.1.1 Interaction Design Considerations

When designing music systems that use graphic tablets or touch surfaces for input, we must take care to understand the capabilities of both the device and the human user. Here we discuss some of the design considerations we must bear in mind when making use of these input devices.

First, we should consider the notions of integrality and separability of dimensions (Jacob et al., 1994). Integral dimensions are linked and cannot be changed independently of one another, such as the x and y components of the position of a mouse. Alternately, separable dimensions must be manipulated independently, like two separate faders on a control panel. We apply this concept to both the physical device's dimensions as well as the dimensions of the perceptual structure of the task; for example, the perceived structure of a 3D space consists of three integral dimensions since they are taken together and not separately. Jacob et al. argue that the input device's structure should match the target task's structure with respect to integrality and separability. Couturier (2006) also writes that control tasks must be matched with appropriate input devices.

When designing our interfaces, we should also bear in mind the notions of chunking and phrasing. Buxton (1995) explains how physically performing an action can reveal a natural "phrasing" of the movement. For example, a "drag and drop" action with a mouse forms a phrase of actions: click on the object, drag while holding the button, and finally release the mouse button. The muscle tension the user employs to hold down the mouse button naturally brackets the parts of the gesture as a single connected "phrase."

We can also note how, similarly, experts think about tasks in chunks and phrases, abstracting away the low-level details and consciously thinking only of the higher-level aggregate action. Buxton argues that we can facilitate the acquisition of high-level task chunking through our interaction design, aligning the physical task phrases with experts' logical phrases.

In working with pen-based input devices, it is important to understand how the brain and motor systems perform handwriting and drawing tasks. Viviani and Terzuolo (1982) describe how handwriting movements are divided into segments ("units of action") that exhibit constant angular velocity. This suggests, they argue, that the brain controls these movements in two stages: first, it plans the trajectory of the next segment, and second, it supervises the trajectory's execution. Basic facts about our motor functioning such as these are important to take into consideration to ensure that our instruments can be used by performers as we intend.

3.1.2 Mapping and Control Strategies

A number of established strategies exist for mapping from the natural 2D space of the tablet's surface to the space of continuous synthesis parameters, as well as other non-spatial mappings. Understanding these fundamental patterns helps us reason clearly about our mapping choices.

Spatial Mappings a) 2D to 2D: We can map from the two-dimensional surface of the tablet to another two-dimensional space. We can do this linearly, by associating each axis with a synthesis parameter (potentially performing a change of basis), or non-linearly, applying some transform to the surface coordinates (e.g., mapping from polar coordinates on the tablet surface to synthesis parameters). Other non-linear mappings are also possible, like clamping or exponential functions. If we use pressure, we can map from the 3D tablet space to a 3D parameter space in this way as well.

b) 2D to nD: We can map from two or three dimensions on the tablet to n-dimensional spaces by defining appropriate mathematical functions, or by interpolating between n-dimensional parameter preset vectors (see Section 2.5.1.1 for examples).

Direct Manipulation The tablet or touch screen can be used to directly manipulate on-screen elements, taking advantage of the user's real-world manipulation skills. This is the standard way of designing desktop interfaces. A GUI on a screen is inherently two-dimensional, so this corresponds well to a tablet or table. This method works both when the screen is integrated with the manipulation surface and when it is separate from it.

Gestural Interaction The movement of the fingers or stylus can be interpreted as a gesture that has a specific meaning, and the gesture's properties (e.g., extents, speed) can parameterize the operation. Wessel and Wright (2002) have described various gestures that are suitable for use with a graphic tablet, such as drag and drop, scrubbing, and dipping. Fiebrink et al. (2007) propose circular gestures for controlling musical processes.

We can use combinations of these strategies as well. For example, Apple's iPod touch (2012) combines direct manipulation with some gestural control, where a "flicking" gesture can be used to slide through a long list of items.

Finally, we can add other top-level logical layers to our interaction. For example, recall Wright's partitioning of the tablet surface (Zbyszyński et al., 2007): the surface is divided into discrete regions, each associated with a different operation or parameter set. By starting a movement in a region, the user selects an operation, and then the gesture can leave the region's extents while the pen tip is still depressed.

3.1.3 Comparison: Characteristics for Musical Control

Computer mice, graphic tablets and touch surfaces are all essentially pointing devices. How should we choose between them for a given control context? Here we examine the relevant differences between these devices for musical applications.

3.1.3.1 Graphic Tablets Versus Mice

Though graphic tablets and mice are both pointing devices, each has its own properties that we must take into account when choosing one for musical control. We can begin by considering the physical properties of each device. With a graphic tablet, the user manipulates the pointer via a stylus or pen, which is much smaller and lighter than a

mouse. This permits much more precise movement and targeting using the natural fine motor skills of our fingers. The size and weight of a typical mouse may prohibit quick, fine movements.

The degrees of freedom offered by each device are a significant consideration. The tablet offers three continuous degrees of freedom (two spatial dimensions and one pressure dimension), while the mouse only offers two spatial dimensions. These continuous degrees of freedom are integral (Jacob et al., 1994) for both devices. Mice and styluses can both offer some discrete control as well; they are often augmented with momentary switches and buttons.

We can consider the positioning characteristics of each device. With a mouse, the positioning is relative: the user can lift the mouse to displace it on the table without moving its associated on-screen pointer. In contrast, graphic tablets use absolute positioning with respect to the tablet's surface. Absolute positioning seems particularly intuitive for certain tasks (e.g., drawing, or exploring a timbre space). Of course, if the tablet itself is moved, so is the spatial frame of reference. One consequence of relative positioning is that when the user lifts his or her hand from the mouse to type, the on-screen pointer stays in position. With the tablet, the pen must be set down altogether to type. The pointer stays in position when the pen is lifted away from the surface, but it jumps discontinuously when the pen is used again. This can be significant if the on-screen pointer position is mapped to continuous synthesis parameters.

Both the mouse and the graphic tablet offer the potential of bimanual control (Kessous and Arfib, 2003). Each occupies only one of the operator's hands, so the other hand is free to use a keyboard or other input device. Such a combination could allow simultaneous discrete (keyboard) and continuous (tablet/mouse) control.

The feedback offered by these two devices is largely equivalent. Both offer passive haptic feedback, in that the user can feel the device and the friction in its movement, along with visual feedback on the computer screen. The haptic feedback gives no positional clues, as it is the same regardless of the pointer's position. The graphic tablet offers the option of putting a printed paper template on the tablet's surface to give further visual feedback about the pointer position.

3.1.3.2 Graphic Tablets Versus Computer Keyboards

Computer keyboards are often used for musical control with live performance software, so it is appropriate to compare them to the use of graphic tablets. The devices are obviously quite different, as the tablet is a pointing device and the keyboard is for text entry. The tablet offers continuous control along three integral dimensions, while the keyboard offers discrete control along many separable dimensions. The tablet can be used discretely, however, by partitioning its surface into regions.

The haptic and tactile feedback offered by the keyboard is more useful for absolute positioning and target acquisition; consider, for example, the prevalence of touch typing. Discrete selection in this manner with a graphic tablet would require constant visual feedback about the current pointer position. The keyboard also allows the user to input discrete commands very quickly by using both hands.

3.1.3.3 Touch Surfaces

Many of the observations we have made for graphic tablets also hold for touch surface input devices. They offer relatively fine control over the pointer since the operator uses his or her fingers directly instead of via a heavy tool. Positioning is continuous and absolute with respect to the touch surface. The surface offers two integral spatial dimensions, and may also integrate pressure. Touch surfaces can feature integrated visual feedback, as in a touch screen, though some graphic tablet models offer this as well. Finally, the key distinguishing factor is that they can offer multi-touch operation. This is also possible with a discrete keyboard, but not with mice or graphic tablets. Such multi-touch operation allows bimanual interaction on a single device.

One potential consideration is the device's accessibility: mice and keyboards are inexpensive and ubiquitous, and graphic tablets are relatively easy to obtain. Touch surfaces are also becoming very common with the rapid popularization of smartphones and tablet computers, but are costly compared to mice and graphic tablets.

3.1.3.4 Selecting an Input Device

Given all of these observations, which device should we select for musical control? This question ultimately depends on our musical task and the mapping we intend to use.

Jacob et al. (1994) note that it is important to match the input device to the task, so that integral tasks are done with appropriately matched integral input devices, and similarly with separable tasks.

Generally, graphic tablets would seem more appropriate for musical control when compared with computer mice, since they can be controlled more finely and can offer more continuous degrees of freedom if that is required. However, the tablet's absolute positioning and limited physical extents might not be appropriate for a given musical control scenario. Also, if discrete on/off control is required, it may be beneficial to use a keyboard alongside a pointing device. If multi-touch or multi-performer use is necessary, then a touch surface input device should be considered.

3.1.4 Summary

We discussed some general interaction design considerations that must be taken into account when designing for tablet and touch surface controllers in the context of musical applications. We enumerated the basic mapping themes that are commonly used, and compared tablets and touch surfaces to mice and keyboards for musical control. Graphic tablets and touch surfaces can be versatile and effective musical controllers when applied to appropriate control tasks.

3.2 Perceptual Considerations

In designing graphical software interfaces for music performance, we must also be aware of the perceptual and cognitive characteristics of the human performer. Here we discuss the perception of the performer from the perspective of the interface designer. These insights about performer perception should inform the design choices that graphical instrument developers make.

Some graphical interface designs for music performance will include active on-screen elements that are intended to be perceived as sound-making objects. A given graphical element needs to be perceived by the brain as a single visual object, and its associated sound needs to be perceived as a single sound object. Further, these two percepts need to be fused through the process of multimodal integration so that the sound is perceived as being produced *by* the on-screen visual object.

In this section, we first examine the way auditory stimuli are partitioned into perceptual groups in terms of auditory scene analysis, one of the most significant theories in the field. Next, we examine relevant aspects of multimodal integration. We then explore the use of animation in graphical software. Finally, we comment on how these facts can be brought to bear on our performance interface designs.

3.2.1 Auditory Scene Analysis

We can describe how the brain segments environmental sound using the concept of auditory scene analysis (Bregman, 1990). When physical entities make sounds, these travel through the air and are compounded at our ears. The auditory scene analysis process describes how the brain partitions this mixed audio signal to recover the original sound events and the sources behind them. Each of these parts corresponds to a basic perceptual unit, the mental structures that are operated upon by higher-level perceptual functions. This partitioning process is involuntary and primitive, in the sense that it is the lowest level of auditory perception. The theory describes the perception of sound functionally; that is to say, it describes the perceptual effects of the auditory system but not how it works at a physiological level.

Auditory scene analysis is divided into two components: sequential integration and simultaneous integration. The former deals with how the perceptual system determines whether successive auditory events come from the same sound-producing event; the latter deals with how we separate out multiple sources that sound simultaneously.

The basic idea underlying all of auditory scene analysis is that the auditory perception system has evolved in the context of real-world, environmental sound sources. Because the real world obeys consistent physical laws, environmental sounds will exhibit acoustic regularities. The auditory system is tuned based on the assumption that sounds will exhibit these regularities and will use them to help make sense of its sensory input (Bregman, 1990, p. 221).

Bregman (1993) describes four basic regularities that sounds in the world tend to exhibit:

- unrelated sounds do not tend to start or stop simultaneously;
- sounds tend to change slowly, in both single sound events or sequences of sounds from a single source;

- vibrating bodies tend to exhibit harmonically-related partials;
- and modulations of the sound tend to happen coherently across its different parts.

These regularities exist due to acoustics and the physical laws of sound production, and are used in both sequential and simultaneous integration.

In auditory scene analysis terminology, when a sequence of sounds is bound together into a perceptual unit in this way, it is called a stream. The process of stream segregation occurs when a succession of sounds splits perceptually into two or more streams. As sounds tend to change slowly, similarity is the primary determinant of sequential grouping in the auditory system (Bregman, 1990, p. 221).

The most basic factors influencing sequential integration are the frequency separation between successive sound events and their spacing in time (pp. 58–65). The frequency separation required is dependent on the repetition rate: when the tones are repeated more quickly, the same segregation can be achieved with a smaller frequency separation. This phenomenon was first demonstrated in the early seventies through experiments by van Noorden, and then by Bregman and Campbell (Bregman, 1990, pp. 58–61). Weaker cues for streaming include having a consistent spatial origin between events, as well as a consistent sound intensity.

We can similarly consider which aspects of the sound help promote simultaneous integration of parallel components. One consequence of the fact that real-world sounds often come from one-piece resonating structures is that when there is some modulation of the sound, either in frequency or in amplitude, it tends to occur coherently across all of the sound's partials. For example, when the pitch of a sound raises slightly, say from tightening the vocal cords in a vocal sound, the frequency of each partial in the mixture is raised by the same relative amount. Sounds often contain small modulations of frequency and amplitude, and these modulations happen in parallel across all of its frequency components. This coherent modulation is a strong clue to the perceptual system that the partials emanate from the same sound source (Bregman, 1990, p. 292). This is reminiscent of the Gestalt principle of "common fate," where the visual perceptual system takes advantage of the fact that the parts of an object tend to move in space coherently (p. 249).

Strings, columns of air, and other vibrating structures tend to create harmonically-related partials due to their resonant modes. This fact leads the auditory system to tend

to fuse harmonically-related partials into a single percept (Bregman, 1990, pp. 232–233). Inharmonically related spectra do not tend to fuse well (p. 237).

Another cue for simultaneous integration is the spatial origin of each partial. Interaural timing, differences in intensity, and filtering due to the ears all combine to give the auditory system clues about the spatial location of a stimulus. The auditory system is able to perform this same operation on the individual frequency components of a sound (Bregman, 1990, p. 297).

Since sounds tend to be stable over short time frames, the perceptual system looks for parts of a mixture that are plausible continuations of sounds from a moment before and tends to preserve those existing groupings. Bregman calls this the "old-plus-new" heuristic (1990, pp. 222–224). The brain then subtracts out the components of the mixture that it assumes are a continuation, and examines the residual components left over separately.

Consistent cues for sequential and simultaneous integration interact to improve the robustness of the streaming process. Our perceptual processes often deal with only partial information, and often the process of perceiving is subject to some amount of error. The perceptual system uses multiple cues to reinforce its percept in light of this error and partial information (Handel, 2006, p. 15). Also, natural sound events give complimentary cues that reinforce each other because they exist in the physical world with its consistent physical laws.

3.2.2 Multimodal Integration: Factors Promoting Fusion

Given that multimodal integration involves signals from separate perceptual systems, we should ask how those signals are combined to form a single fused percept. For audition and vision, the two main factors are correlation in time and in space (Driver and Spence, 2000, p. R732; Bregman, 1990, p. 181; Handel, 2006, pp. 406–414). If a sound and a visual event occur simultaneously and originate in the same location, the brain typically assumes that the two are related and come from the same external object. Handel (2006, p. 404) describes the "unity assumption," where the perceptual system assumes each sound comes from some object in the visual field. He also adds a third integration factor to the two above, which he calls "compellingness," the "a priori belief in the connection between the auditory and visual stimuli" (p. 414). Observers that have a

preexisting expectation of a connection between the stimuli are more likely to perceive one.

A certain amount of discrepancy in the correlation between the events is tolerated. The perceptual system uses three general strategies to resolve these conflicts (Handel, 2006, p. 404):

- disregard one of the stimuli as an error;
- combine the stimuli into a compromise percept; or
- decide that the stimuli come from two separate objects.

Integrating multimodal signals in this way can be understood as a strategy for improving the robustness of our perceptual estimates (Bregman, 1990, p. 307).

The influence of temporal synchrony that we see here for multimodal integration is reminiscent of the influence of onset synchrony in simultaneous integration. Bregman notes, though, that there is not much influence of visual stimulation on simultaneous integration (pp. 290–291).

3.2.3 Cognitive Demands of Animated Musical Interfaces

One important feature of interactive software systems is their use of animation. We might reasonably expect that animated interfaces place more cognitive load on a musician than static printed scores. Since animations change from moment to moment, we can expect that more and more information is being conveyed to the performer at each instant. If this is true, could these increased cognitive demands overwhelm the performer?

Living in the real world, we take in stimuli from moving sources constantly. The human perceptual system is well tuned to moving elements. We might then argue that animated sensory input could be easier to parse than static input (Bartram, 1998). With a static score, the performer needs to be able to imagine the encoded musical gestures; in an animated interface, he or she can simply experience them. Robertson et al. (1993, p. 61) argue that having smooth, continuous animation reduces users' cognitive load since they need not re-parse the scene and reestablish object correspondences from moment to moment.

Humans perceive the natural world multimodally (Ernst and Bülthoff, 2004). The task of perceiving the animated scene might thus become still easier if there is some amount of coordination and redundant information between perceptual streams. Natural events give rise to coordinated sensory signals, so similar coordination between the audio and visual parts of an animated representation should help us to understand it. Also, interacting with interfaces causes kinesthetic sensations in the user. If these are also coherent with the interface's multimodal feedback in a natural way, it may further reinforce the resulting percept and help the user make sense of the interaction.

We have to be careful about our conclusions on animated representations, however. In the fields of HCI and education, it is not completely clear whether animation in and of itself helps or hinders the user (Scaife and Rogers, 1996; Tversky et al., 2002). The animation must be designed with respect to the user's perceptual and cognitive traits. If coordinated multimodal stimuli should help the user, the inverse is also true: uncoordinated multimodal stimuli might serve to confuse the user. Researchers seem to agree that much more work needs to be done on this topic to ensure that animations support the intended use instead of distracting from the task at hand.

3.2.4 Implications for Interface Design

So what can these observations and research tell us about how to design graphical software interfaces for musical performance? In particular, we would like to consider direct-manipulation interfaces that feature on-screen elements that act like active sound-producing objects. These interfaces can feature many simultaneously active elements that the performer must manage. What heuristics should we follow to arrive at a successful interface design?

The fundamental rule that we should observe is that the interface must respect the user's perceptual facilities and cognitive constraints. As we have seen, the human perceptual system is tuned to be sensitive to real-world objects in an environment and to all of the physical regularities that these exhibit. We should use these environmental objects as a model. If we intend for an interface element to be perceived as an object, we must use the cues that indicate to our perceptual system that the element is an object. More precisely, the visual and auditory stimuli should be perceived as objects both individually and multimodally.

3.2.4.1 Objecthood Cues

Each on-screen interface element will have both a visual and an auditory manifestation. First, the audio component must appear to come from a single source. We can ensure this by appealing to the factors used in the auditory scene analysis process, as described above. Overall, this means that an audio signal intended to appear as a single source should behave coherently overall, and it should change continuously and not too rapidly.

If the sound is made of a series of events, it should observe the appropriate sequential grouping factors: overall similarity between events, appropriately small frequency separation, and appropriately small time separation between events. The spectral components of each event should be integrated as well, with synchronized partial onsets, coherent modulation, coherent spatial location, and consistent spectral relations. Generally, synthesis algorithms will ensure that all of these spectral requirements are met, but we must keep these factors in mind when using more exotic synthesis techniques.

Similarly, the visual elements of the interface should provide the appropriate perceptual cues to indicate their objecthood.

Recalling the unity assumption (Section 3.2.2), we should also ensure that each sound object should correspond to one (visible) interface element.

Given our appropriately-designed auditory and visual stimuli that can be individually segregated into objects, we must further ensure that they are fused together into a single percept through multimodal integration. First, they must be tightly synchronized in time so that visual events (movements, onsets) are accompanied by corresponding activity in the audible event. Discontinuities in the audio should be correlated with discontinuities in the visuals, and vice versa. These facts suggest that we should pay special attention to accurate timing and low latency in our program code. Second, the auditory and visual stimuli must be perceived to be originating from the same location in space. For example, we could design the interface so that the left and right screen positions could control the panning of the sound event.

3.2.4.2 Multiple Objects in Motion

One distinguishing detail of the graphical interface style that we are considering here is that it can feature many on-screen elements that are all moving at once. The use of

motion in the interface design would be a key element, as motion gives important information about the world around us. There are integration processes for dynamic stimuli over and above those for static stimuli, and the integration of dynamic information happens early on in the perceptual process (Soto-Faraco and Kingstone, 2004, p. 64).

One practical consideration is that the user will have to be able to track many simultaneous movements in order to control the elements effectively. Scholl (2001) gives evidence that bears on this issue. He describes results concerning "multiple object tracking," where a subject is required to visually track "independently and unpredictably moving identical items" (pp. 9–10). The results show that "subjects can successfully perform this task (with over 85% accuracy) when tracking up to five targets in a field of ten identical items" (p. 10). This suggests an approximate number of objects that we can reasonably expect users to be able to manage. An important point that Scholl makes is that only discrete objects with respect to visual grouping can be tracked in this way. This suggests that perceptual objecthood is an important factor to consider in designing our graphical interfaces.

These results hold in the visual modality, but they say nothing about the auditory modality. We would expect that multimodal, fused percepts would only help to strengthen the sense of each interface element's overall objecthood.

Another detail to consider is that in our direct-manipulation interface, the objects must not be moving too fast to be acquired and controlled. The user must be able to grab the elements with the mouse, or an alternate input scheme may be necessary. Also, continuous motions hold together better in audition and in vision (Bregman, 1990, p. 179), so we should take care to ensure continuity.

Using many concurrent, moving interface elements may tax the user's attentional capabilities. There is a relatively small limit to the number of objects that can be attended to at once (Bregman, 1990, p. 667). This limit will have to be respected in the interface design, and not all objects should require constant attention. Attention is still useful in this context, however: the use of attention in the multiple object tracking task described above sped up the response time to the attended objects (Scholl, 2001, p. 10).

3.2.4.3 Schemas and Object Identification

Prior knowledge of an object affects the segregation process. Handel describes how the "compellingness" of multimodal stimuli can help fuse them into a single percept (2006, pp. 414–415). Bregman also indicates that schema-based perceptual processes can affect lower-level ones (1990, pp. 666–667). These high-level processes "prime" the lower-level ones to tell them what to expect.

We can exploit this effect in our interface design to help to fuse the visual and auditory signals we use. We could use stimuli that tap into existing object knowledge that the user is likely to have; for example, we could use an animation of a hammer in sync with its associated sound. Alternatively, we could have the user learn particular sound and visual associations over time. Practice with the software and reinforcement learning of its interface elements will help to fuse their audio and visual aspects.

3.2.5 Summary

We have explored perceptual issues in the context of graphical performance software design. In particular, we examined some perceptual processes related to the segmentation of changing sensory information: the process of object formation from an auditory scene analysis perspective, the integration of visual and auditory information, issues related to animated interfaces, and the human limits of object tracking.

A fundamental part of designing graphical performance software is a solid understanding of human perception. The basic lesson we can learn is that we must design our systems to respect the perceptual mechanisms and cognitive limits of our users. By supplying users with the kinds of cues they have evolved to use, we can make our performance systems more effective.

3.3 Potential of Applying Information Visualization Techniques to Graphical Instruments

In designing our graphical software instruments, we would benefit from investigating how other fields solve similar problems. The field of information visualization studies how to convey information effectively through graphics, and therefore serves as a fertile area from which to draw ideas and inspiration. Information visualization researchers

think carefully about how to convey data visually, making use of spatial relations, shape and colour. Their careful consideration of how to depict information visually for human consumption and interaction offers instrument designers clear guidelines on conveying information effectively. We outline here some of the ways in which techniques from information visualization can benefit our instrument designs.

The foundations of information visualization research are human-centred and are substantiated by user testing. The ultimate goal is to facilitate human understanding of large datasets by amplifying our cognitive capabilities to make sense of the information (Card et al., 1999, pp. 1–6). Visual representations are externalizations of information, used as cognitive aids. They help us reason about data and discover patterns in it in ways that are not possible using only our minds.

One way to amplify our cognitive capabilities is through perceptual inference, which means that the visualizations are designed to make use of the cognitive and preattentive cues that our visual perceptual systems are sensitive to (p. 16). These systems pick out specific features of the visual field that tend to be meaningful in a natural environment—edges, continuity of visual textures, coherent movement—and can parse those features automatically. When we design visualizations to take advantage of these features, we tune them to the abilities of the users that we are designing them for.

A common problem in information visualization is conveying high-dimensional data with a spatially two-dimensional graphic display. Various visual structures can be used: making the markings' retinal properties meaningful, rendering the topology of the data, or using 3D graphics (pp. 23–31). This can be useful for our purposes since graphic performance interfaces often drive complex systems that contain a large amount of state information.

Another issue that the information visualization literature covers is the use of animated representations. Humans are naturally tuned to seeing objects in motion, so this can be a useful and powerful strategy to exploit. As we have seen, using continuous animation can lower users' cognitive load (Robertson et al., 1993; Bartram, 1998).

Various specific techniques could be applicable to interfaces for music. One is to employ a "focus plus context" strategy to be able to explore a large amount of data (for example, see Lamping and Rao, 1996). Other techniques have been developed for navigation around data spaces (Card et al., 1999, p. 31). This could be useful in a graphical instrument for navigating through a timbre space. For example, we could add such a

visualization to Van Nort's interfaces (2004).

3.3.1 Differing Perspectives

The information visualization community has much to offer to designers of graphical instruments. However, the application domains are different. In traditional information visualization, there is a scientific approach to conveying information. In the case of graphical musical instruments, user creativity, emotional response and aesthetics are also important considerations. Further, interaction with the visualization is as important as how it conveys information, so we need to bear in mind HCI and user experience.

In information visualization, the approach seems to be to isolate the various dimensions of the data, then isolate the dimensions of visual structures, and then associate the two in a one-to-one manner (Card et al., 1999, pp. 17–31). Tweedie (1997, p. 376) notes that it is also instructive to visualize some derived metadata as well. For example, we could plot the mean of a given set of data points to glean some more information about them. It is well known in digital musical instrument design that complex mappings from control to audio synthesis can lead to more engaging and expressive instruments (Hunt et al., 2002); perhaps the same might be true of mapping data to a visual representation in the context of musical control.

Finally, graphical instruments are designed to be played, or at least interacted with. The dominant paradigm in information visualization is navigation though and viewing of large data sets. This is not necessarily well suited to musical interaction (except perhaps in the case of timbre space navigation).

3.4 Direct Manipulation versus Indirect Control

The main goal in dealing with graphical instruments is to find interesting ways to use software that are appropriate and effective for performance. Traditional WIMP interface designs (see Section 2.3) are common and well understood, but we should be careful to evaluate them from the perspective of musical performance.

The dominant paradigm in UI design is direct manipulation (Shneiderman, 1983). Users move and manipulate on-screen elements with physical gestures that resemble their real-world analogues. The graphics update smoothly and dynamically to maintain

the illusion of real manipulation. Indirect manipulation—the use of typed commands or menu items to manipulate objects—is also used in today's interfaces quite often. For example, Microsoft Office's context menus are a typical example of indirect manipulation. To make text italic, the user does not just grab it and physically slant it; he or she selects a menu item, issuing a command to make the text italic.

The advantages of direct manipulation are obvious for graphical interfaces for music. At a high level of expertise, musicians are able to express themselves directly and spontaneously without necessarily needing to consciously attend to all the physical details of their movements. Interfaces that are cognitively or physically unnatural could impede this expert mode of playing. Direct manipulation designs are intended to be relatively transparent by employing natural manipulation gestures, ostensibly lowering the cognitive demands on the user. Here we see interface designers tuning the interaction to match our human physical and cognitive capabilities (Shneiderman, 1983; Shneiderman et al., 1992; Tweedie, 1997).

However, direct manipulation is not always appropriate. We can imagine situations in performance where we could achieve something with a command more effectively than with direct manipulation. We need to design the instrument to be *effective* in performance, and that may mean resisting a dogmatic adherence to the direct manipulation paradigm.

For example, when an operation has to be done globally on all of the screen elements, it can be tedious to perform the operation on each element in turn. In the Different Strokes environment, for example, we can imagine wanting to delete all of the green strokes at once. Also, when the user has to perform an operation on many objects in succession, it can be helpful to be able to define a template (macro) operation or script and have the system apply it to all the candidate objects on the screen. These kinds of operations are better suited to indirect manipulation.

Indirect manipulation is also useful for operations that do not immediately suggest an appropriate physical metaphor. For example, the user might want to switch the program into a different mode of operation.

Another situation where indirect manipulation could be beneficial is when the user would need to perform some manipulation precisely, like speeding up the musical tempo by a factor of two exactly. That would be suited to numerical entry or some similar method.

Finally, in the case of interacting with moving objects on a screen, it might be difficult to manually target and manipulate an object due to its movement. Indirect manipulation could make it possible to modify the objects even though they cannot be acquired by the mouse or pen.

Other interaction philosophies are being explored that depart from the traditional indirect manipulation and currently dominant direct manipulation techniques. Beaudouin-Lafon (2000) proposes instrumental interaction as a new model for designing interfaces that extends the direct manipulation approach. It positions itself somewhere between direct and indirect manipulation, and may be potentially interesting for graphical musical instrument design (Couturier, 2006).

3.5 Employing Modalities Other Than Visuals

In Information Visualization, there are two main possibilities for mapping information to modalities other than vision: sound and haptics. These are the only other practical modalities for input/output devices. (Smell and taste are not commonly used in computer interfaces.) The use of a multimodal interface could be beneficial as each modality is well suited to convey particular kinds of perceptual information, and we can leverage these advantages. For example, visuals have better spatial resolution than audio, and audio has better temporal resolution than visuals (Handel, 2006, p. 416).

There are some good reasons for taking advantage of extra modalities to complement our visual interface. First, and most important, each modality represents an extra channel to carry information to our performer. Humans constantly receive information via all senses in parallel in the natural environment, so we are well adapted to this input. Second, as noted in Section 3.2.3, we can design our system so the signals to each modality reinforce each other to build a stronger percept for the user.

The critical point to address in considering modalities other than visuals is that for each information stream that we would like to convey, we must choose to represent it in a modality that is appropriate for it. For example, temporal information is best conveyed aurally, as noted above. We must equally ensure that we provide an appropriate mapping to the target modality. An inappropriate mapping could compromise the intended feedback to the performer.

3.5.1 Auditory Modality

We can use auditory display (also known as sonification) to transmit information to the user. In our case, using sonification for informational purposes presents a slight challenge since the ultimate aesthetic output of our instrument is itself sound. Kramer (1994, pp. 6–13) outlines multiple advantages of auditory display, on its own and in conjunction with other display modalities.

First, sound is not very directional; sound waves radiate outward in all directions. This can be an advantage in that the performer need not physically look at the display at all times to be getting feedback from the instrument. We can also use sonification for alerting purposes. Some kind of sound event can be emitted when the instrument goes into a state that the performer needs to know about. This can happen ambiently, without requiring the full attention of the performer. For example, the sonification might be a quiet, filtered noise signal, signifying that the instrument is in a given state. The performer might not need to attend to the sound while it is stable, but it would grab his or her attention when the timbre changes to indicate a new state. Finally, since sound has more acute temporal resolution than visuals, it could be used in the interface for time-accurate feedback.

The use of sonification in our case requires some careful thought since the main product of our graphical instrument will also be sound. The instrument's audio output is itself a stream of feedback that the performer will rely on, and any additional sonification would have to remain separate from it by monitoring via headphones or with on-stage sound monitors. The sonification should also not clash spectrally with the instrument's output; the signals should be made to occupy different parts of the frequency spectrum or could be acoustically separated in some other way.

3.5.2 Haptic Modality

Haptics is an extremely important, natural modality to use in a musical interaction since tactile and kinesthetic feedback is present in all acoustic musical instruments (Jordà Puig, 2005, pp. 139–141). Performers rely on the tactile feedback of their resonating instruments to help isolate their sound in an ensemble. Off-the-shelf haptic devices are also readily available for standard computer systems. These can accomplish standard pointing tasks just as a mouse or a graphic tablet, but can simultaneously output forces

back out to the performer.

Haptic devices have the important properly of simultaneously supporting input and output through a single device. This could be an important factor in creating the illusion of interacting with real objects and inducing perceptual fusion (Hayward et al., 2004, pp. 17–18).

Haptic renderings can simulate many sensations, including pressure, temperature, friction, softness, wetness, vibrotactile signals, and proprioception (p. 21). Haptic displays are also capable of rendering small details of a scene that would otherwise clutter a graphical display (p. 20). This could be an advantage when used in conjunction with graphics.

Finally, Hayward et al. argue that "creators often prefer to use their hands directly" (p. 20), making haptics a good fit for creative instruments.

We explore the use of a haptically-enabled input device to control *Different Strokes* in Chapter 5.

3.6 Conclusion

We have explored various topics that have an important bearing on our performance software designs. The use of graphic tablets in a musical context was discussed, along with perceptual and cognitive considerations, potential lessons from with the Information Visualization field, the appropriateness of direct manipulation designs, and the use of sonification and haptic feedback in performance software. In the next chapter, we begin to explore the use of a given software application in varied hardware contexts to bring insights about its design to light.

Chapter 4

Different Strokes Application Development

A number of feature extensions were made to the *Different Strokes* (DS) application over the course of this work, in the context of various research projects. Two of these projects are described in this chapter: first, the *d_verse* project, a multi-disciplinary artistic work; and second, the adaptation of DS to operate on a multi-touch table. This chapter describes the various features that were developed to accommodate the needs of these projects, the challenges they posed, and the lessons learned about the interface's design through adaptation to these new usage contexts. We begin by describing the original *Different Strokes* project, in order to contextualize the extensions presented in this and subsequent chapters.

4.1 *Different Strokes* Main Characteristics

Different Strokes is an experimental software interface for computer music performance based on drawing gestures (Zadel and Scavone, 2006a,b, 2008). It is intended for use in a solo graphical software performance[1] context. A fundamental design goal of the software is to improve performer control and transparency in graphical software performance, which is traditionally dominated by computerized automation and indis-

[1] As described in Section 2.1, we define graphical software performance to be musical performance through generic computer hardware, where the performer plays music by operating an on-screen user interface. This is also known as laptop performance.

cernible performance gestures. The impetus behind the project was to explore interfaces amenable to solo improvisation that provide a demystifying visualization to present to the audience.

The *Different Strokes* software starts as a black, full-screen canvas. The user draws strokes on the screen using a digitizing tablet, as one would in a freehand drawing application. The strokes become tracks along which *particles* can move.

Particles form the program's main animated element. Particles appear as white point lights that slide along the on-screen strokes, moving according to the temporal evolution of the original drawing gesture. Quickly drawn strokes produce fast-moving particles, and similarly for slowly drawn strokes. Particles always move in the drawing direction of a stroke. A screenshot is shown in Figure 4.1.

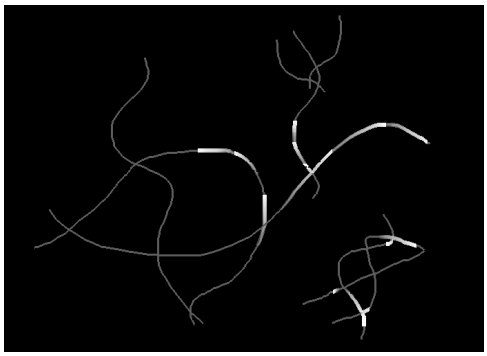


Figure 4.1 A screenshot of the *Different Strokes* interface.

The intersections between the strokes are meaningful: when a moving particle arrives at an intersection, it is duplicated and the two particles continue outward along the two outgoing tracks. See Figure 4.2 for an illustration of this behaviour. This mechanism allows particles to orbit continuously around closed stroke loops (i.e., self-intersecting strokes), bringing about periodic motion (Figure 4.3).

In order to introduce at least one particle into the system, there is always a particle attached to the drawing head. The performer can introduce a particle into an existing network of strokes by crossing one of the existing strokes. This will cause a new particle to be made immediately, starting from the intersection and flowing down into the existing network (Figure 4.4).

By drawing networks of strokes, the performer creates cascading and looping patterns of animated particles. The movements of these particles are in turn mapped to

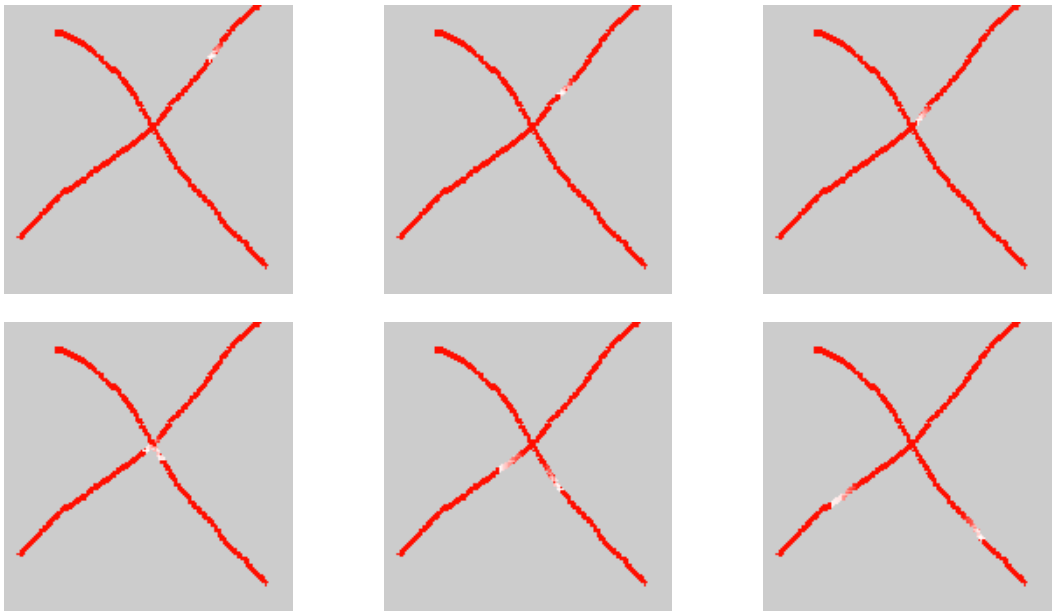


Figure 4.2 Particle behaviour at stroke intersections (Zadel, 2006).

sound sample playback: each stroke may be associated with a sound file and each particle on the stroke represents a playback head. The motion of the particles, and thus the corresponding patterns of sounds, is controlled by creating and modifying stroke topologies in real time. This novel, visual approach to sequencing is intended to be non-analytical and intuitive to the performer, as well as appreciable by the audience (when projected). See Zadel and Scavone (2006a,b) for more information about common stroke shapes and system behaviours.

The original software design used variable speed sample playback to generate sound. A pre-loaded sound file is selected via the keyboard and associated with subsequently drawn strokes. The entire sound file is mapped to the stroke such that the sound file's beginning and end correspond to the beginning and end of the stroke. As a particle moves along the stroke, it acts as a playhead that scrubs through the wavetable according to its movement. A consequence of this mapping is that playback speed is dictated by drawing speed: quickly drawn strokes may play back at a high pitch and slowly drawn strokes may play back at a low pitch, depending on the length of the stroke and the length of the associated sound file. Changes in drawing speed result in correspondingly changing playback speeds.

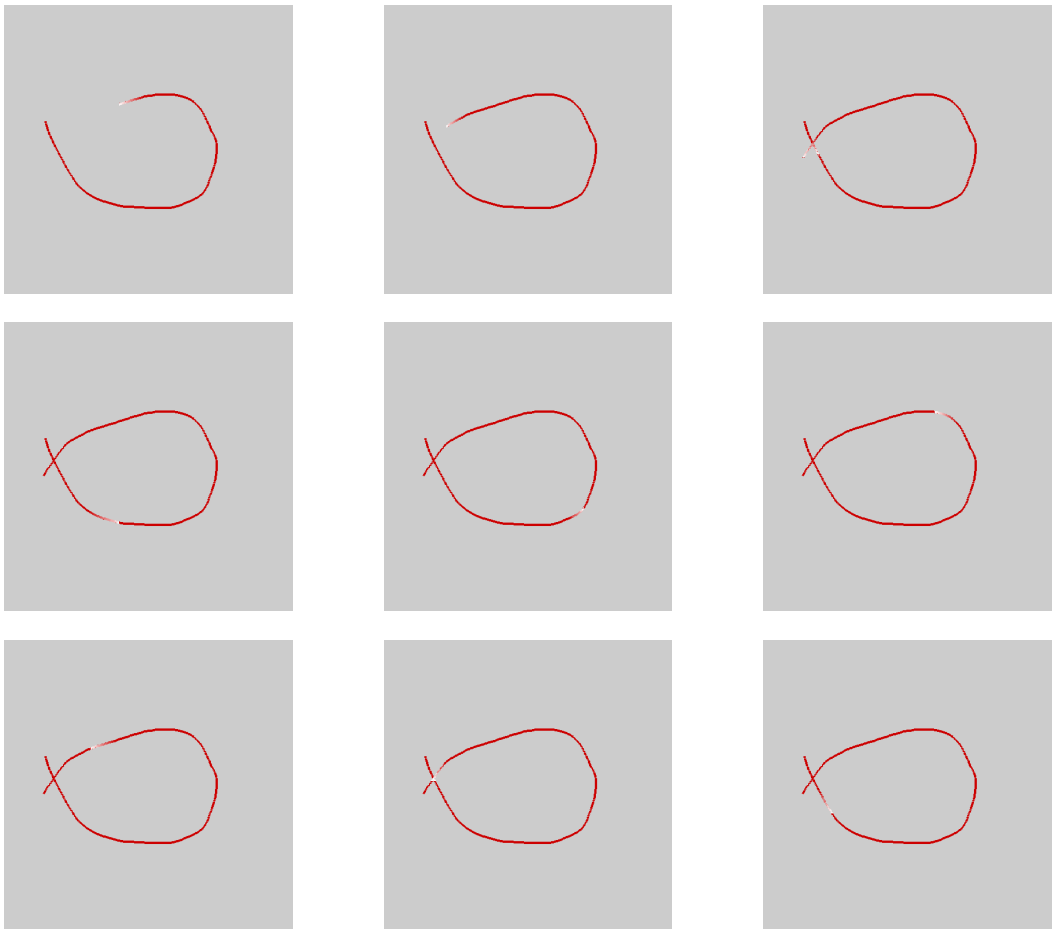


Figure 4.3 An example of a self-intersecting stroke, which creates a loop.

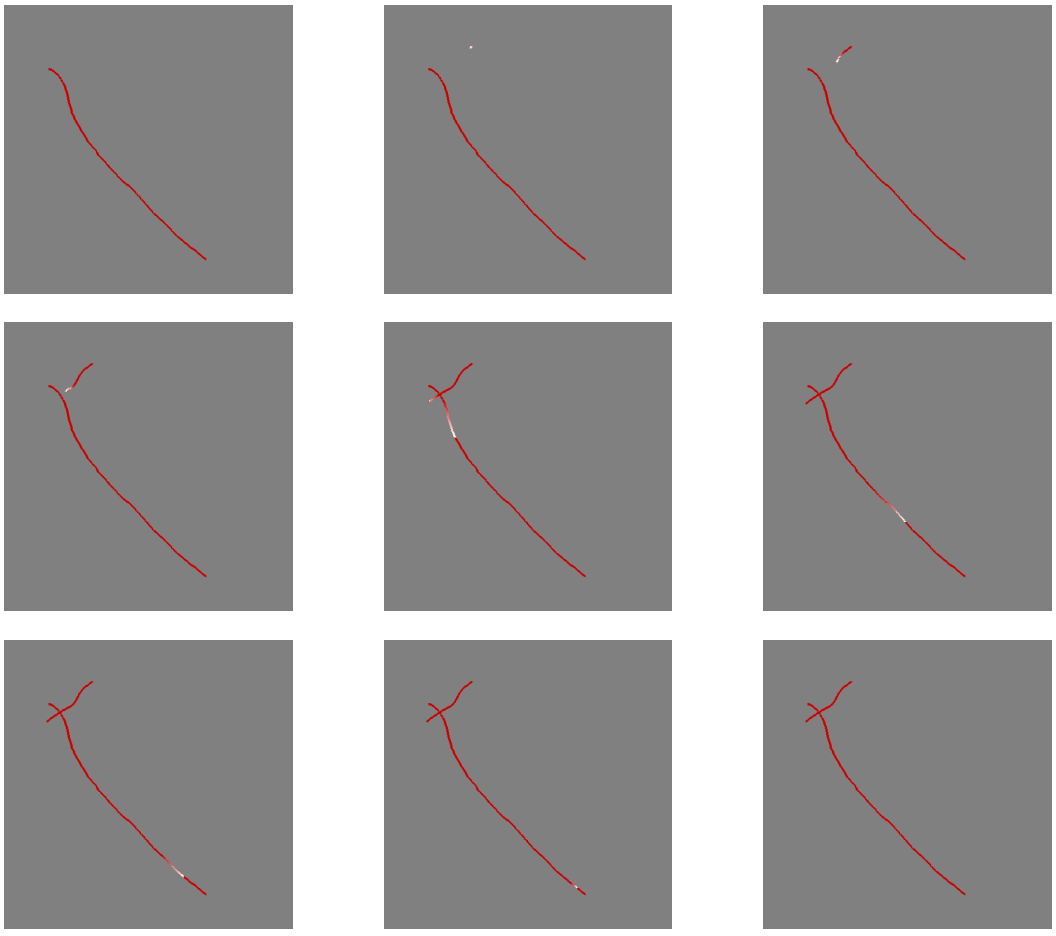


Figure 4.4 Introducing particles by crossing an existing stroke.

DS takes advantage of the typical user's long experience with stylus tools for creating directed markings on paper. It allows the performer to transfer this deeply ingrained expertise to the control of a sound-making process by having these markings relate directly to parameters of sound. The use of drawing gestures to drive the system also leads to some other attractive properties. For example, the natural variability of hand movement induces organic, non-quantized timing in the sound patterns.

Controlling DS is a bimanual task: one hand draws with the graphic tablet, and the other hand presses keystrokes on the keyboard. These actions are often done quickly and in a coordinated fashion. For example, the performer might quickly introduce a stroke into a network to create a sound pattern, and then quickly delete it to silence the sound pattern. This performative drawing is at the heart of a *Different Strokes* performance.

The crossing gesture (crossing an existing on-screen stroke with the cursor) is also used for removing strokes. The user hits a key to enter the removal mode, and then the user crosses the strokes he or she would like to have deleted. The deleted stroke disappears immediately and the associated sound is stopped.

The colour of a stroke indicates which sound file it is associated with, or that it is silent. A given sound file will always produce strokes of a consistent colour, so that the user can tell at a glance how each stroke will sound. Silent strokes are coloured blue.

4.1.1 DS Software Design

The software was designed to be modular. There are five main components, each containing its own set of classes. These subsystems are illustrated in Figure 4.5.

The workspace The workspace serves as a container for all of the strokes. It contains a list of stroke objects, each of which is a list of points in a 2D plane. It is not dependent on any of the other subsystems in the software. This is the core of the simulation.

The scheduler This set of classes has a notion of the current time, and can schedule *events* that are to be dispatched at particular times in the future. The simulation is animated by stepping the scheduler forward by small increments of time. When the scheduler is stepped forward to a given time, pending events scheduled to

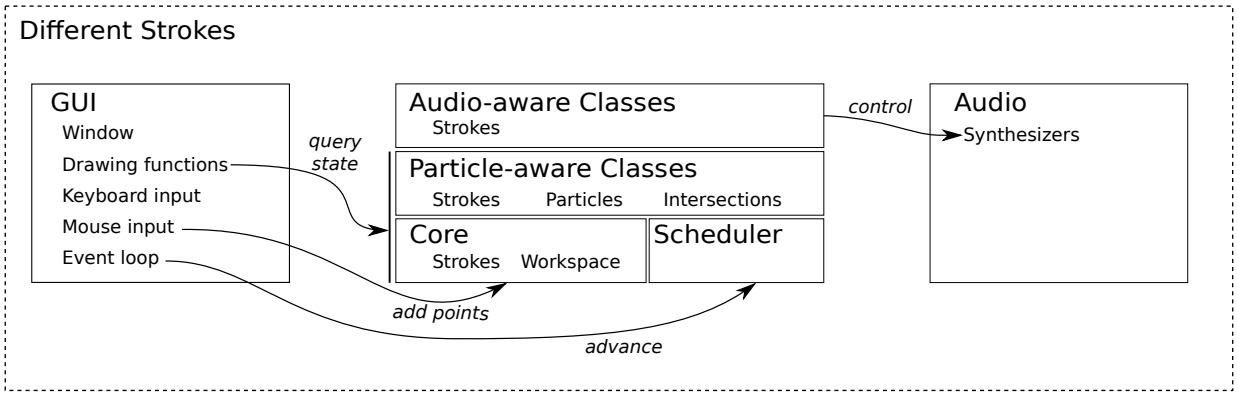


Figure 4.5 The main components of the DS code are the graphics subsystem (GUI), the simulation subsystem, and the audio subsystem. Arrows indicate the interactions between the components. The simulation is made up of layers of classes that each add extra functionality to the one below.

execute before that time are dispatched and then cleared from the scheduler. The scheduler is stepped forward to the current time at each animation frame.

The Particle-Aware Classes A *stroke set* is a set of classes that implement the specific behaviours present in the animation. This stroke set is aware of particles, their movement, and what to do when they reach intersections between strokes. It extends (subclasses) the basic strokes in the workspace to implement this behaviour. The particle-aware stroke set depends on the scheduler as well in order to implement animation. These classes do not depend on the graphics or audio subsystems in any way.

The audio subsystem The above classes are subclassed to provide audio functionality. These audio-aware stroke classes decorate the existing stroke methods by adding audio calls that are invoked when certain occurrences happen in the simulation. For example, as a particle moves along a stroke, it computes its speed; the audio-aware subclass of the particle class uses that speed in turn to drive the speed of an associated sample playback synthesizer.

This subsystem contains synthesizer classes that generate streams of samples to send out to the audio hardware. These synthesizers are controlled by real-time occurrences in the simulation as it is animated.

The audio layer also contains an interface to the audio hardware; this is imple-

mented through `RtAudio` (Scavone and Cook, 2005).

The graphics subsystem The graphics subsystem contains all of the drawing code. At each frame, the state of the simulation is advanced to the current time, and the drawing functions render out its state as a 2D drawing. The drawing procedure is read-only in the simulation state.

Various software extensions to the original *Different Strokes* project were implemented to bring the system into novel hardware contexts, and to address the needs of the research projects that made use of the system. The first of these was the *d_verse* project.

4.2 The *d_verse* Project

Between 2007–2009, the *Different Strokes* environment was used in a collaborative art project entitled *d_verse: transitional algorhythms of gesture* (langshaw, 2011). The project was comprised of a series of performance and video works exploring the themes of communication and gesture. It was directed by poet pk langshaw of Concordia University, in Montreal. It brings together a number of different disciplines—dance, film, sound art, and poetry—and explores the permeability of the barriers between them. The works combine live performers and various real-time, digital processing systems to create an active, responsive whole. The abstract notion of *line* is the common aesthetic element shared between the different disciplines: a hand-drawn line in cursive writing, a line traced out by a dancer's movements, projected wireframe images. This theme is used as the medium of artistic communication between the disciplines.

Visually, the works involve dancers onstage in a black box environment. The dancers move in response to the other performance elements or according to prearranged choreography. A commonly explored aspect of these interactions has involved the projection of video onto or around the dancers, with accompanying real-time spatialized audio. The projection elements include real-time processing of captured video, interactive Flash animations, and the *Different Strokes* interface. Photos from the project production sessions and rehearsals are shown in Figures 4.6–4.10

An important element of the *d_verse* project is the close interrelation between the individual disciplines. At various points in the works, each artistic element is expected to

Figure 4.6 An image from *d_verse* filming, which took place during the summer of 2008. Note the projection of the *Different Strokes* interface. (Photos courtesy pk langshaw.)

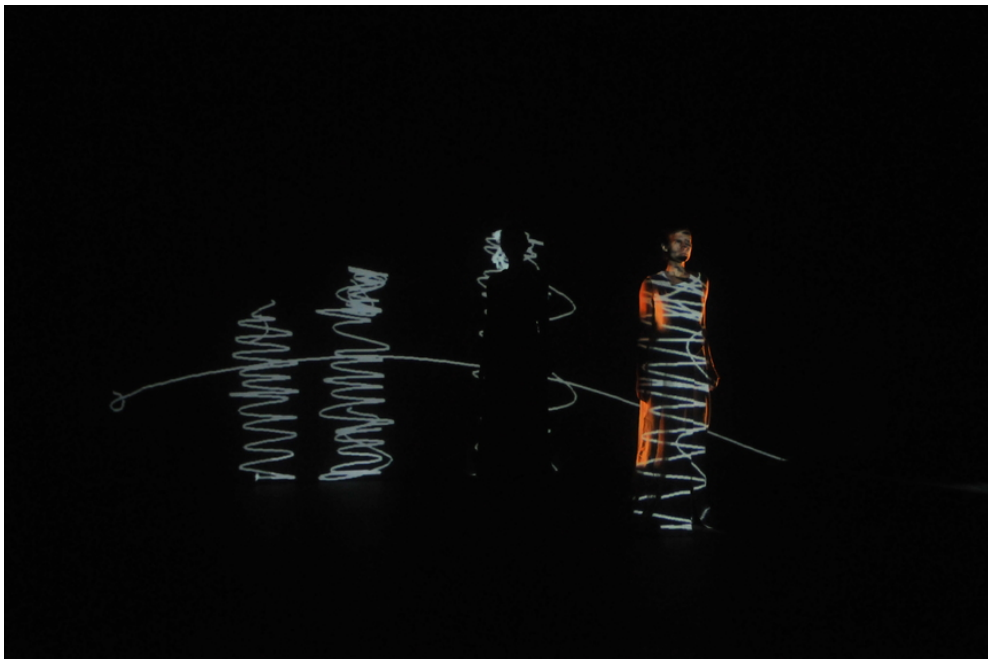


Figure 4.7 An image from *d_verse* filming.

Figure 4.8 An image from *d_verse* filming.

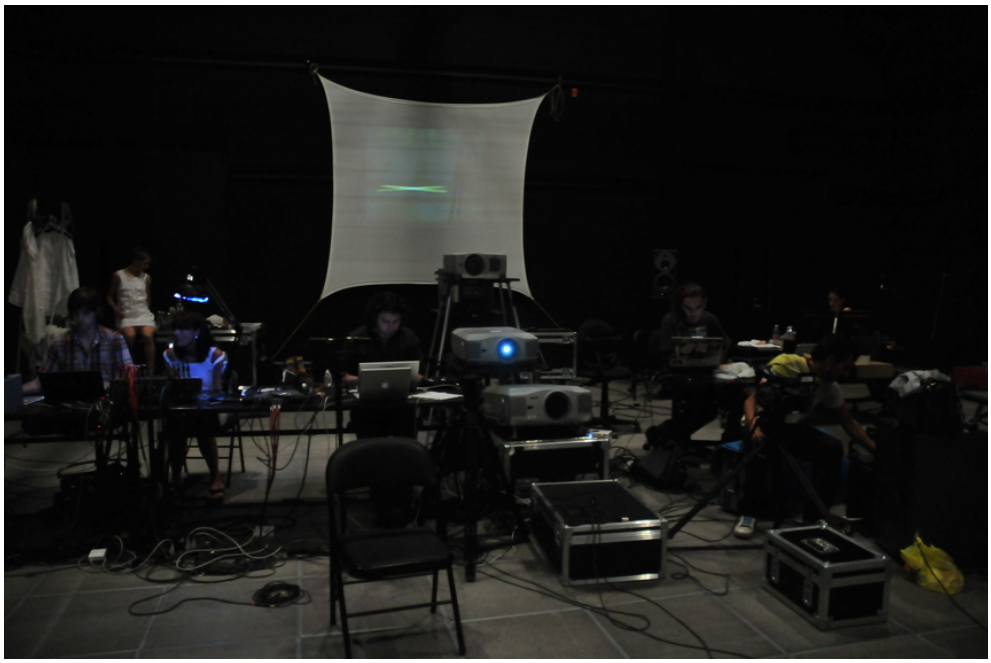


Figure 4.9 The off-camera computer operators with projection equipment.

Figure 4.10 An image from an early *d_verse* development session, summer 2007. The *Different Strokes* interface was projected onto a scrim in front of the dancer.

take the lead, while the others follow, reinterpret and react. Each discipline is responsible in part for creating the narrative, and they are all engaged in a dialogue with each other.

Different Strokes was chosen as a major audio-generating component of the *d_verse* project because of its tight coupling of audio and visual media, as well as its use of gestural line. Work has focused on extensions to the *Different Strokes* interface to accommodate the needs of the project, and in modifying its sound generation algorithms to meet the group's aesthetic goals. Timothy Sutton, a Concordia University graduate, acted as composer and was responsible for coordinating the overall musical aesthetic.

Different Strokes was used to control audio processes and to simultaneously give a visual representation of that control. A significant part of the audio in *d_verse* came from *Different Strokes*, treated with outboard processing (simultaneously through Max and external hardware). Recordings of recited poetry formed a fundamental component of the soundscape, either played back directly or through a granular synthesis algorithm. The DS system was also modified to allow recited poetry to be recorded in real-time for subsequent granulation.

The *d_verse* project took place as a series of development workshops and a final filming session. Multiple participants took part in these sessions, each representing their own discipline. Most of these participants acted as performers, either on- or off-stage, all working simultaneously and improvising in concert with the others. The various performers and their roles are detailed in Table 4.1. In this chapter, the word "performer" is taken to mean the DS performer unless otherwise specified.

Performer Role	Description
Different Strokes performer	Operated DS
Dancer(s)	Movement on-stage
Filmmaker(s)	Operating the camera
Music composer	Overall sound aesthetic, operated Max/MSP
Interactive media performer(s)	Operated the projected elements
Lighting designer	Dynamically operated the stage lighting

Table 4.1 The various performers involved during the *d_verse* filming and development sessions.

4.2.1 d_verse and *Different Strokes*

The *d_verse* project's focus on interrelated disciplines mirrors the integration of audio and visuals in Different Strokes. As *d_verse* was realized through real-time performance and dynamic gesture, *Different Strokes* was a natural fit in this context.

One of the main themes of the *d_verse* project was the exploration of the idea and aesthetics of gestural lines. *Different Strokes* evokes gestures in two ways: the visual amplification of the performer's physical drawing gesture, and the flowing, gestural movements of the animated particles. The particles' movements along the strokes resonate with the dancers' movements. Since the project is based around words, both spoken and written, the software was also used as an activated writing surface for handwritten text. The input device most typically used with DS is a digital drawing tablet (e.g., a Wacom tablet), and these devices are well suited to capturing flowing drawing gestures.

While gestural and real-time qualities of *Different Strokes* lent themselves well to the *d_verse* project, certain aspects needed to be updated. These are elaborated upon in the following section.

4.2.2 Feature Updates

Certain features were added to the *Different Strokes* software over the development phase of the project. These included multi-channel support, granular synthesis functionality, mappings from the simulation features to synthesis parameters, and updates for robustness. The extensions to the DS architecture that were made for the *d_verse* project are illustrated in Figure 4.11.

4.2.2.1 Multi-Channel Audio and Sound Placement

The application was updated with a multi-channel signal path to allow sound spatialization. The Synthesis ToolKit's (STK) StkFrames structure (Scavone, 2012) was introduced into the code to easily manage blocks of multi-channel audio.

The spatial placement of mono synthesis voices was one of the particular design challenges in the development efforts. There is one synthesis voice per stroke, associated with a single "sounding" particle. There are many features associated with this particle that could be used in the mapping: its position in space, its normalized position along

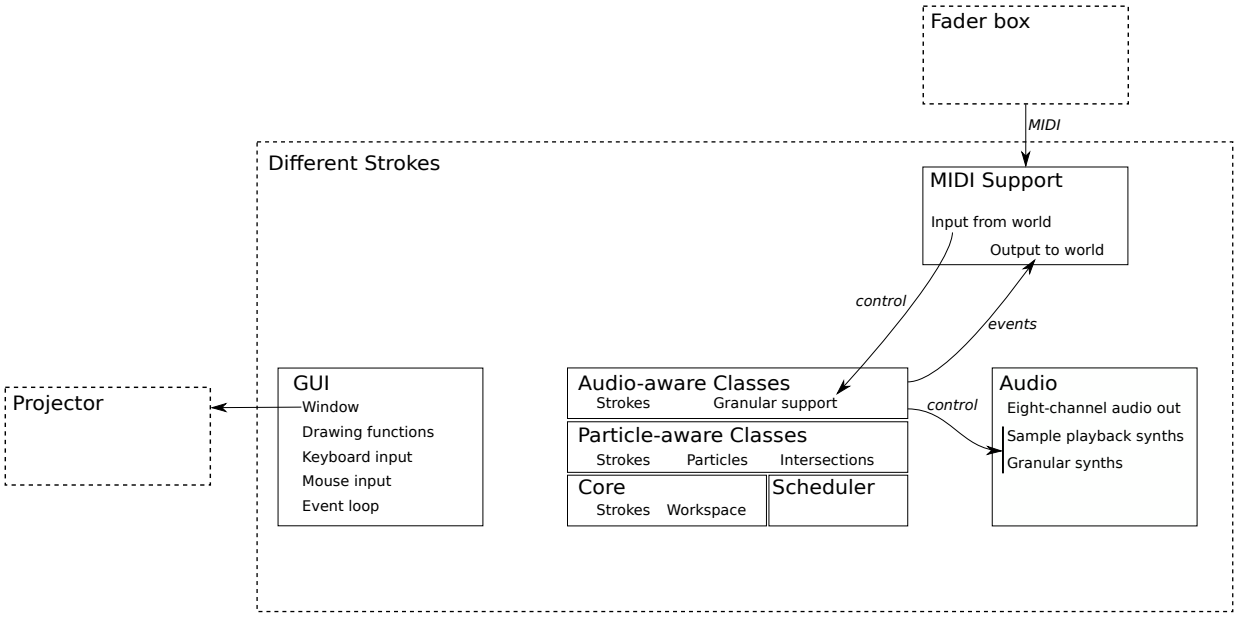


Figure 4.11 System diagram illustrating the new features added to *Different Strokes* for the *d_verse* project.

a stroke, its speed, and stroke curvature at its position being just a few examples. The panning algorithm implemented uses the particle position in the global 2D plane to determine the sound's position in space. Thus, a large loop containing a particle that orbits the whole screen will produce a sound that circles the central listening position.

Panning was implemented using vector base amplitude panning (VBAP) (Pulkki, 1997). A non-general implementation of VBAP specific to a circular speaker array was done, located in the `MPI_AudioPanner` class. The speakers were assumed to be lying on a circle and equally spaced along its circumference. This simplified the implementation by only requiring a single azimuth value to place a given monophonic sound in the speaker array. This assumption could obviously not be made in general, but was appropriate for the *d_verse* piece.

In `MPI_AudioPlayheadStroke::updateState()`, the panning value is set according to the sounding particle's position in the 2D plane. The azimuth value for a sounding particle is computed with respect to the centre of the 2D drawing canvas.

Code is included in `MPI_AudioPanner` to configure the rotation and direction of the speaker circle to suit the physical placement of the speakers in the performance space. The speakers are assumed to be numbered consecutively in space from one through n,

where n is the number of speakers, in either a clockwise or counter-clockwise direction.

The algorithm works by first determining which pair of speakers straddle the panning position, and the two signal gains are then computed using the "tangent law" method and normalized so that $g_1^2 + g_2^2 = 1$ (i.e., to ensure equal power between the two active speakers) (Pulkki, 1997).

Each stroke's synthesis voice is panned independently according to its sounding particle's current (x, y) position. Thus, many spatially separated sources can be heard moving around the sound field simultaneously, one for each stroke making sound.

One consequence of associating a specific location of the drawing canvas with a location in the sound space was that the *d_verse* project often called for drawing in a particular part of the visual space. For example, the performer might be directed to draw on a dancer onstage in a particular way, but this would also bind the sound to the corresponding location in the speaker array. This coupling could sometimes be problematic during performance.

The `MPI_AudioPannerStereo` class was implemented for use on standard stereo speaker systems outside of the context of *d_verse*. It works similarly to the `MPI_AudioPanner` class, but pans the audio from a sounding particle between the left and right speakers depending on the horizontal position of the particle (with a particle at the left of the screen being panned to the left, and similarly for the right). The algorithm does simple equal-power panning as a function of the linear x position of the sounding particle. The $x = 0$ position is taken to be the centre of the canvas along the horizontal axis.

4.2.2.2 Granulation

The *d_verse* project called for real-time processing of spoken poetry. Granulation of an audio input buffer was added to the application using the `Granulate` object from STK, with sound input via the appropriate facility in `RtAudio` (Scavone and Cook, 2005). A new "granular" stroke type was added to the software for this functionality, selectable via a predefined keystroke. The idea behind the granulation subsystem was to add an extra option for performance-time transformation of the sound.

A challenge in integrating granular synthesis was to define the mapping from the simulation state to the granular parameters. There are many individual parameters in a

granular synthesis model that can be controlled, and many parameters in the simulation that can be queried.

One option that was discussed was to associate vectors of predefined granular synthesis parameters with various points in the 2D drawing plane. The 2D position of a sounding particle would be used to determine a granular parameter mapping by linearly interpolating the spatial settings, allowing the parameters to evolve in time as a particle moves along a stroke. This could be done using methods described by Van Nort (2009), Choi et al. (1995) or Bencina (2005). This scheme would suffer, however, from the same issues mentioned in the previous section: having to draw on particular parts of the 2D drawing surface could conflict with having to follow the onstage movement of the physical dancers.

The solution that was ultimately used was to control the granular synthesis parameters externally via a MIDI fader box. This decoupled the synthesis control from the simulation, and made it possible to effect various timbres independent of the drawing position. While this deviated from the notion of using only the interface to control the sound output, the added flexibility was needed for the *d_verse* project. One disadvantage presented by this scheme, however, was the increased number of physical controls that the performer was required to manipulate during performance (see Section 4.2.3.4). The granular parameters are set globally, and affect all of the granular synthesis processes identically.

Two methods were implemented for associating granular synthesizers to DS's animated elements. In the first, each granular stroke owned its own input buffer and synthesis voice. As the pen touched the tablet, recording was turned on for the current stroke, and its input buffer was filled. Recording stopped when the pen was lifted. With this approach, the number of synthesis voices could grow quickly, as there is one voice per drawn stroke. Consequently, the computational load could easily reach the capacity of a given CPU. Also, the aural space became cluttered very quickly as the performer drew. This method was consequently only used during the development workshop sessions.

In the second approach, which was used during production, one global granular synthesis process was used. The software was set up to capture audio into a buffer in the software on a key command, which would then be granulated immediately. There was a mapping from the amount of activity in the on-screen animation (the number

of active particles on granular-typed strokes) and the number of voices in the global granular synthesizer.

This scheme featured one global audio buffer stored in an `MPI_AudioGranularWorkspace` instance. The buffer was written circularly; when the write pointer reaches the end of the buffer, it jumps back to the beginning. The size of the buffer is set in the code at compile time. (We used six seconds at 44kHz sampling rate.)

The `MPI_AudioGranularWorkspace` instance contains a single granular synthesizer with multiple voices. There is a linear mapping between the number of particles that are currently moving along granular strokes and the number of voices, such that more particles mean more voices are sounding. For example, for one active particle on a granular stroke, one granular voice sounds; on twenty granular particles, eight voices sound; and for six active particles, two granular voices sound. The maximum that can be sounding at once is limited to eight voices. This was done to reduce the amount of aural clutter and CPU load created by the granular synthesis setup.

A challenge with *Different Strokes* in the *d_verse* project was the fact that it is a relatively inertial system, where a persistent visual structure is set in motion and then allowed to run. This runs contrary to the mapping in ostensibly similar systems, where the drawn gestures correspond directly to commensurate aural gestures. In designing the mapping, it would have been most elegant to derive output parameters solely from on-screen elements (e.g. particle position along the stroke, or the number of particles on a stroke). The system's inertial quality was however at odds with the needs of the piece, which sometimes called for quick, gestural sound changes in response to the dancer's movements (e.g. bringing the sound up to a crescendo, coordinated with the dancing). In production and workshopping, the directors of the piece needed more immediate control of the sound.

To address this issue, we decided to control the different granular parameters individually using an external hardware control surface, as mentioned above. This allowed us to control the audio parameters in a gestural and practical manner, but extended the mapping outside of the established DS hardware setup (i.e., computer keyboard and drawing tablet). This compromise was necessary for the aesthetic constraints of the *d_verse* project. This belies a central tension, however, between inertial control, as seen in DS, and immediate, low-level instrumental control. *Different Strokes* is performative

with respect to setting up and modifying a symbolic system (as defined by Malloch et al. (2006)), but then that system runs autonomously, and its output can be difficult to change gesturally.

4.2.2.3 MIDI input and output

MIDI input and output was added to *Different Strokes* to help communicate with external software, and to receive control signals from external hardware. Various events in the interface caused MIDI messages to be sent to the composer's Max/MSP setup. MIDI input was received from the external hardware fader box, and was used to drive parameter changes in the granular synthesis process.

Since the audio code in the application was done at a low level in C++ using STK, the implementation effort required to add various audio effects was high compared to other prototyping languages such as Pd or Max. This fact led to the decision to use external software processing and plugins to help vary the sound instead of reimplementing existing audio effects. This had the added benefit of distributing the computational load for audio processing across multiple computers. The external system received the *Different Strokes* audio, as well as control data in the form of MIDI messages.

For output, events such as "pen up," "pen down," particle creation, and other information were broadcast as Control Change messages. Timothy Sutton, the composer on the *d_verse* project, generated impulsive audio events when particle creation messages were received by his Max/MSP setup, and these sounds were mixed into the other audio material he had composed.

MIDI capabilities were added via the `RtMidi` class (Scavone and Cook, 2005). One global `MPI_Midi` instance encapsulates the application-wide MIDI functionality, which essentially wraps `RtMidi`. The `MPI_MidiMapping` class defines the correspondence between the input MIDI CC message values and the granular synthesis parameters. In the *d_verse* project, knobs and faders were linearly mapped to the following parameters:

- read rate (for pitch-shifting);
- the number of overlapping granular voices;
- the time-stretching factor;

- the ramp percentage (affecting the shape of the trapezoidal window applied to each grain);
- the starting offset into the wavetable;
- the delay between grains;
- the degree of randomization;
- and the gain.

Also included was a reset button to reset the offset into the wavetable.

4.2.2.4 Robustness Improvements

The original *Different Strokes* application exhibited a feedback problem where certain stroke topologies (double loops) would result in an ever-increasing number of particles (Zadel, 2006). The *d_verse* project required the used of cursive handwriting gestures, so the program had to be modified to be robust with respect to these topologies.

A simple solution was implemented to address this issue, where a fixed upper limit is enforced on the number of particles allowed per stroke. If a stroke already contains its maximum quota of particles, no new particles will be introduced.

In the current scheme, only three particles per stroke are allowed. When a given stroke already contains three particles, any new particles that would be created at intersections are ignored and are not created. The number of particles per stroke is global and is set at compile time. The value of three particles per stroke is arbitrary, though the number should not be too high for performance reasons. See Figure 4.12 for an illustration.

This leads to some interesting effects that can be exploited by the performer. Since the strokes are drawn manually, there is natural variability in the timing of the particle propagation, and variability in the on-screen positions of the intersections. This means that particles can be produced at an intersection in an irregular way leading to pseudo-random behaviours, even as DS is an entirely deterministic system. This can be interesting from a sequencing perspective, where a given topology or phrase can be triggered irregularly.

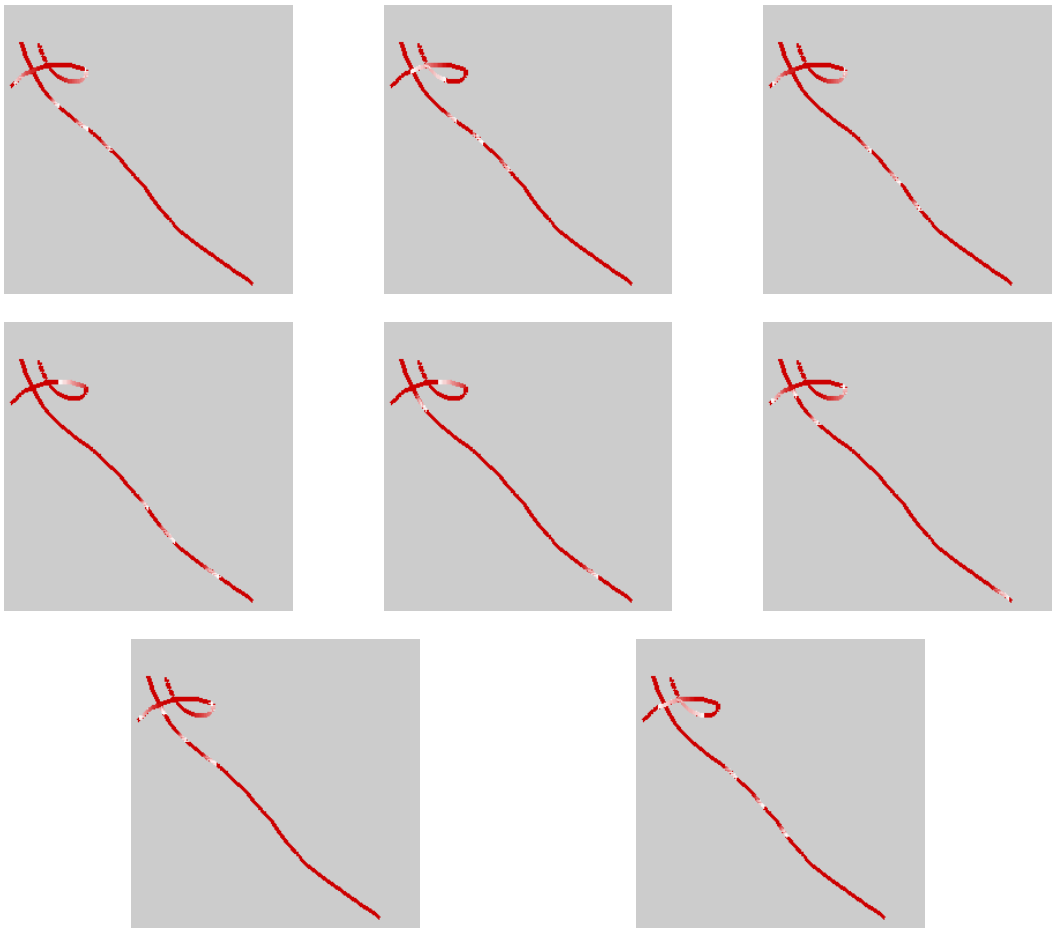


Figure 4.12 To prevent an ever-increasing number of particles in some looping topologies, a limit of three particles per stroke is imposed. Particles flowing into an intersection will only create new particles on the perpendicular stroke if there are fewer than three particles already on it. In the above illustration, the looping stroke continuously delivers particles to the straight stroke, but only three particles can exist on the straight stroke at any moment.

Also, one can "steal" particles from a stroke by scribbling backwards at its end. This effectively introduces more particles downstream from any other intersections, suppressing the introduction of new particles and thus removing them from the structure. In this way, particles can be removed from a shape where it would have previously been impossible.

4.2.2.5 Garbage Collection

Garbage collection was implemented to remove inactive particles, improving the performance of the system overall.

Particles slide along each stroke and stop at the very end, where they will have no subsequent effect on sound playback since they are no longer active. Garbage collection for these particles is done globally. At each frame of animation, in the `MPI_Workspace update()` call, each stroke is updated; in `MPI_ParticleStroke::updateState()`, all of the particles that are at the last position on the stroke are removed.

4.2.2.6 Smoothing Playback Discontinuities

In *Different Strokes*, moving particles scrub through their associated wavetable according to their speed and position. As new particles are added, the playhead jumps to the new position, creating interesting stuttering effects. Unfortunately, this also introduces discontinuities and clicks in the audio output.

A crossfading feature was created to help alleviate this problem. Two wavetable playback voices are associated with each stroke, with only one sounding at a time. When a new particle is added, it obtains the currently silent voice, and the old voice is crossfaded to the new one, producing a smooth transition.

One drawback of this scheme is that there are only two wavetable playback voices per stroke that operate at a fixed, global crossfade time. If new particles are introduced while in the middle of an existing crossfade, the playback discontinuity will be audible. The crossfade time is short enough (50ms) that this is not usually a problem.

4.2.3 Unexpected Challenges

There were a number of interesting, unexpected issues that arose over the course of the group development sessions for the *d_verse* project.

4.2.3.1 Key Commands in Low Lighting

The *Different Strokes* interface design has used key commands for selecting the stroke drawing mode. For example, the user hits the "2" key to select wavetable number two, and subsequently drawn strokes are associated with this wavetable until the mode is changed again. In adding extra features, new keystrokes were assigned, and there are now a fair number of meaningful keys for use in this project. Relatively precise targeting is required in the performer's non-dominant hand to find the desired key.

d_verse rehearsals have occurred in dark black box environments, making it very difficult to see the keyboard. One drawback of using many individual keyboard commands is the fact that it is difficult to find individual keys without adequate lighting. While using a small light or a back-lit keyboard is a quick solution, simplified keyboard commands or an alternate method of selecting modes that does not require visual attention may be preferable to allow effective performance in these conditions.

4.2.3.2 Parallax and Drawing on Dancers

One of the more aesthetically appealing effects in the *d_verse* project was when the interface strokes were drawn in order to make their projection fall on the dancers and the set in an interesting way. This is visually attractive, but can be difficult due to the parallax from the differing positions of the projector and the performer's point of view. If these are not lined up closely, it can be challenging to target a particular part of the physical space. A certain amount of occlusion from the dancers' bodies also occurs that can make targeting difficult.

As discussed above, *Different Strokes* is an inertial system, in the sense that drawn strokes remain on the canvas until they are explicitly deleted by the performer. If the performer draws on the dancer and he or she moves away, the drawn stroke remains in place until it is deleted. It is extra work for the audio performer to clear strokes that are no longer needed visually. One possible future solution would be to remove strokes

automatically after a predefined amount of time, or to slowly fade out the stroke and its corresponding audio over time.

4.2.3.3 Visual Versus Audio Precedence

In developing the project, there was a difference of perspectives on how to use the software. The system was designed as a controller for audio performance. The visual aspect of *Different Strokes* serves as an interface and a visualization, but not necessarily an aesthetic end in itself. In *d_verse*, artistic visuals were central to the project, and the desire on the part of the artistic director was to use the software in a primarily visual way with the audio as a sonification of the graphics.

4.2.3.4 Multiple Points of Focus

Another interesting challenge that was encountered in applying *Different Strokes* in the *d_verse* project came from projecting the interface. In the original software design, the performer must focus his or her attention on two things: the computer screen and the computer keyboard. In *d_verse*, the projection of the interface onto the stage is used to draw on the dancers and interact with other stage elements. As such, it requires the performer's attention, and it can be difficult to alternate between the various contexts quickly (keyboard, screen, projection). This situation is challenging to accommodate by the original software design model.

One solution could be to add a dedicated hardware device for use in the DS performer's non-dominant hand that would not need visual attention (as can sometimes be the case with the computer keyboard). For example, a specially-designed sensor device could be used, or a commercial device, such as a SpaceNavigator (3Dconnexion, 2012).

Interestingly, the drawing tablet itself was not found to require visual attention since the pen stays in the performer's hand and the tablet surface is stationary. There is a one-to-one relationship between the hand motion and the motion of the cursor on the screen, and this is enough for the performer to track the pen's location. This is similar to the way a computer user does not need to attend to a mouse to use it, so long as they do not let go of it.

4.3 Multi-touch port

Another extension of *Different Strokes* relates to its use on multi-touch hardware. This section describes modifications to the software for it to be used on two touch surfaces: a large-scale table and a portable tablet. We describe the implementation details and lessons learned in each case.

4.3.1 Motivation

There has been a lot of research energy expended recently on multi-touch devices, to bring these ideas to new physical contexts using new sensing techniques. This comes in tandem with the commercial proliferation of these devices (smartphones, tablet computers, large-format touchscreens). New interaction techniques are being developed on this hardware as well (e.g., pinching, swiping). This climate makes it easier than before to port DS to this hardware. Also, since DS is screen-oriented and works with drawing, it is a good candidate for these multi-touch surfaces.

The potential benefits include more intuitive interaction, because the user draws directly on the visual representation of the system. Also, new kinds of drawing and performance are possible when the software can sense multiple points of contact simultaneously, such as bimanual schemes or multi-user use.

Different Strokes was ported to a prototype multi-touch surface in the Input Devices and Music Interaction Laboratory (IDMIL) at McGill University. This allowed us to experiment with the expanded potential of the software in this hardware context.

4.3.2 System Implementation

The original *Different Strokes* code assumed a single drawing point, i.e. a mouse pointer or the tip of a digitizing pen. For porting the application to multi-touch interfaces, such as the iPad and smartphones, the code was updated to allow multiple concurrent drawing points.

The port was originally done in the context of a multi-touch table that was designed in the IDMIL by Master's student Bruno Angeles. The *Different Strokes* code was updated to allow multiple drawing points, and Bruno interfaced this code to work on his *TactoSonix* multi-touch hardware. This involved replacing the DS graphics and input

handling with an equivalent version written on top of openFrameworks (2012), since this was the environment used with the table. openFrameworks is a set of C++ libraries for artistic applications, which features multi-touch support. A system diagram for the extensions to *Different Strokes* to work on multi-touch hardware is given in Figure 4.13.

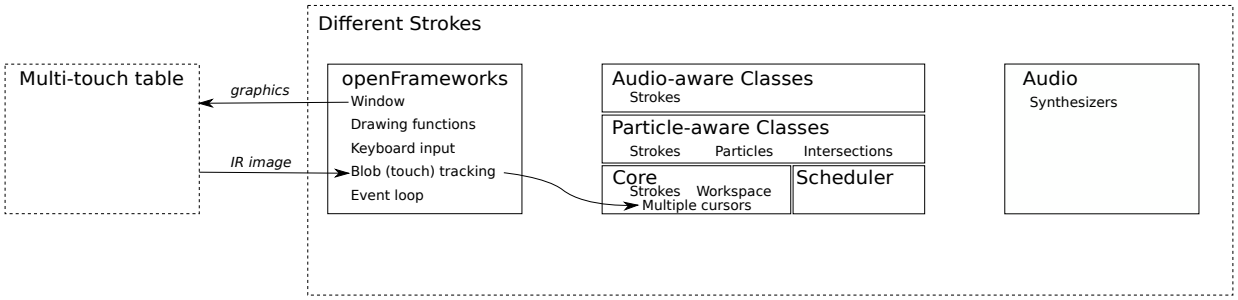


Figure 4.13 System diagram illustrating the new features added to *Different Strokes* to support multi-touch interaction.

4.3.2.1 Multi-touch Table Hardware

As mentioned, the multi-touch implementation of DS was done using Bruno Angeles' *TactoSonix* project. *TactoSonix* is a hardware and software ensemble for multi-touch musical interaction, consisting of a custom designed multi-touch table and an associated software suite.

The table uses the frustrated total internal reflection (FTIR) approach to multi-touch sensing (Miranda and Wanderley, 2006, p. 32; Davidson and Han, 2006) described in Section 2.5.2.1. The surface of the multi-touch table is made of plexiglass, and contains infrared (IR) LEDs embedded internally along its edge. These LEDs allow IR light to be shone inside the plexiglass surface. Because of the light's angle of incidence and the refractive properties at the top and bottom surfaces of the plexiglass, the IR light reflects completely off these surfaces and stays within the plexiglass. When a finger touches the surface, however, it changes the refractive properties at that point, and some of the light escapes near the point of contact through the bottom surface of the plexiglass.

This reflected light can then be captured with a camera positioned below the surface. The points of finger contact appear as bright blobs in the image. These blobs are then tracked from frame to frame using image processing methods to determine when a contact point exists and what its position is. Images of the table and its internals are shown

in Figure 4.14.

The *TactoSonix* system complements this custom hardware component with a corresponding software system. The software system uses two parts, one associated with the camera and one associated with the projection. The video frames from the infrared camera, which contain the visible images of the points of contact, are processed to isolate these points. This is done using Community Core Vision (2012), a software package designed for multi-touch tables. Community Core Vision (CCV) provides tools for calibrating the video processing and for blob tracking. The IR image frames are converted into isolated touch events, equivalent to "touch detected at position $(0.3, -0.1)$."

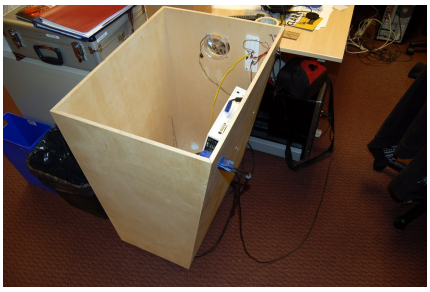
These events are then transmitted to a separate application using the TUIO (Tangible User Interface Objects) protocol (Kaltenbrunner et al., 2005), which is based on OSC. This protocol was developed for use in multi-touch interaction, and communicates messages that indicate which blobs are present from frame to frame, and what their positions and orientations are. The purpose of TUIO is to decouple the sensing part of a multi-touch system from the rest of the internal logic to make it possible for artists to concentrate on the graphics and interaction design. In our case, the blob creation and movement data is passed to the openFrameworks version of DS, and the touch positions are used as cursor movements to drive the system.

The second half of the *TactoSonix* system consists of applications that receive TUIO, and then produce the associated graphics and sound. These are written in openFrameworks, which contains APIs for handling each of these subparts. These applications also contain the program logic that defines the behaviour of the system. The generated images are projected back through the projector onto the plexiglass surface from below, where they can be seen by the user. This setup allows the graphics to be "touched" directly. By colocating the sensing and the generated graphics at the plexiglass surface, a heightened sense of immersion is possible than in systems where there is a spatial disconnect between the sensing and the graphic output (e.g., a mouse and computer screen).

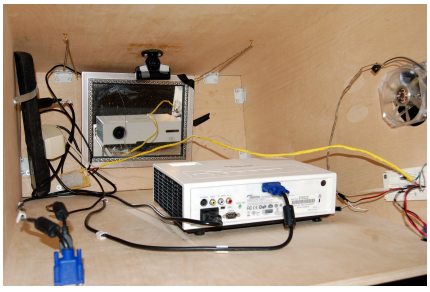
For the projection, the upper surface of the plexiglass panel is covered with a sheet of thin paper as a layer to receive the projection. The projector and the IR camera are situated inside the table's enclosure, a few feet underneath the plexiglass surface, as can be seen in Figure 4.14(c).

An alternative implementation approach would have been to integrate a TUIO li-

(a)



(b)



(c)

Figure 4.14 Images of the *TactoSonix* multi-touch table. (a) The whole unit. (b) The table without the plexiglass top, showing the projector. (c) An interior view, showing the IR camera, projector and mirror. (Photos courtesy Bruno Angeles.)

brary into DS itself, and retain the existing graphics layer instead of porting it to openFrameworks. This would have been able to directly receive the communication from the CCV component. This was not chosen, however, because of the reusable body of code that Angeles had already built up as part of *TactoSonix*.

4.3.2.2 *Different Strokes* updates

Originally, *Different Strokes* was intended for use with a graphic tablet, which only has one point of contact, the pen tip. This assumption was carried through the code, which relied on a single C++ STL iterator (`MPI_Workspace::currentPolyLine_`) to indicate the current stroke that was being appended to.

DS was updated to maintain a hash table of current (x, y) positions, indexed by a parameter called `cursorid`. `MPI_Workspace::beginPolyLine()`, `MPI_Workspace::endPolyLine()`, and `MPI_Workspace::appendPolyLinePoint()` were updated to accept this `cursorid` parameter, which was used to index into a hash table of active cursors. Each cursor is a pointer to the `MPI_PolyLine` to which it appends points.

When the user touches and removes their finger from the table, the blob tracking algorithm in the CCV component detects this, and calls the `ofDifferentStrokes::blobDown()` and `ofDifferentStrokes::blobUp()` functions in the openFrameworks wrapper for multi-touch DS. These, in turn, call the `MPI_Workspace::beginPolyLine()` and `MPI_Workspace::endPolyLine()` functions in DS.

The values for `cursorid` are allocated and managed by the *TactoSonix* system. CCV abstracts the notion of a multi-touch interface, so each new movement event simply receives `(x,y,touchid)` triples. When a user touches a finger to the glass and moves it, its `touchid` remains consistent from frame to frame and does not change while the finger is in contact. This ID number is used directly by *Different Strokes* as the `cursorid` parameter.

Photos of DS running in its multi-touch incarnation are shown in Figure 4.15.

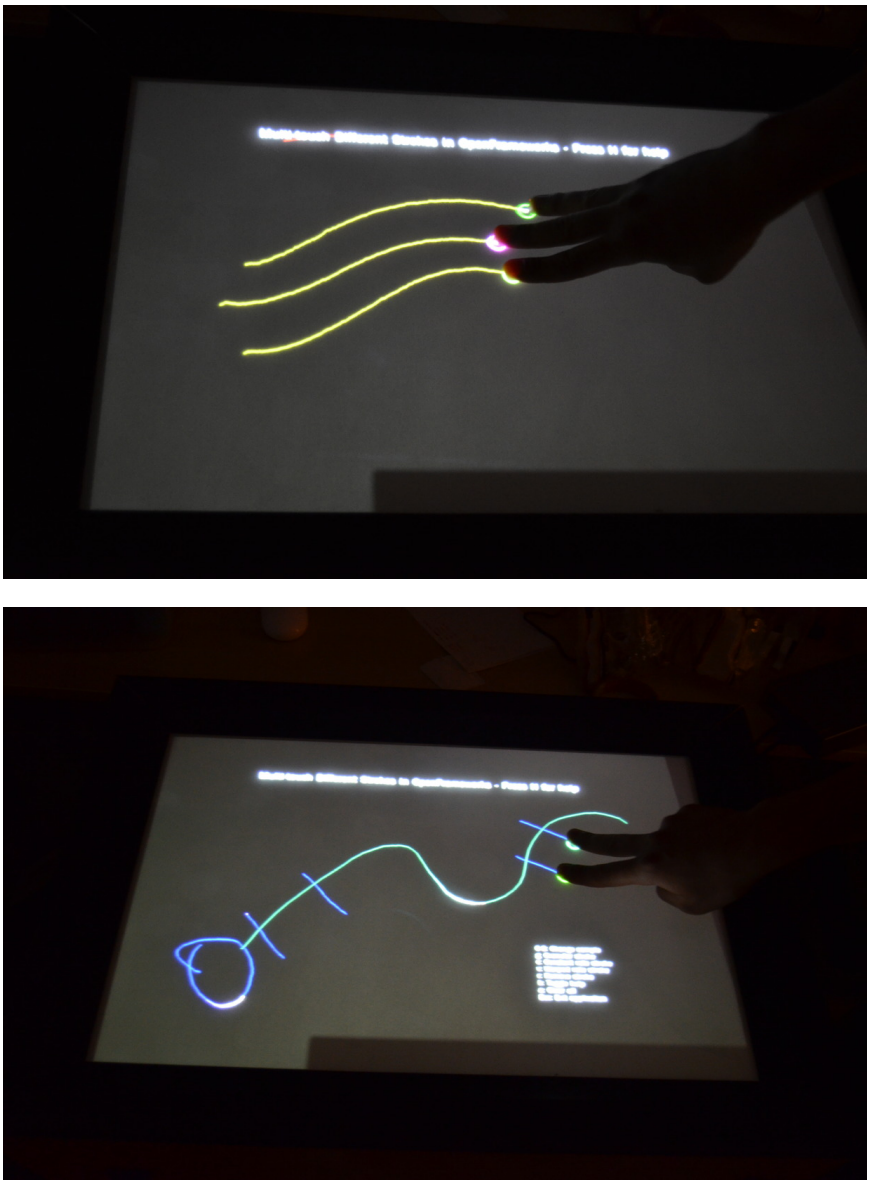


Figure 4.15 Photos of *Different Strokes* running on the *TactoSonix* multi-touch table.

4.3.3 Interaction Design

4.3.3.1 Benefits

Running *Different Strokes* in a multi-touch environment opens up the potential for contemporary styles of touch-oriented gestural interaction. For example, we could use two-finger gestures like pinching to zoom and orient the drawing. These gestures are well understood by users due to their prominence on smartphones and tablet computers.

Also, because of the large format of the table and the fact that it is multi-touch, multiple performers could work on the table at the same time. This allows new interaction dynamics since performers can work in synchrony, making sound in a coordinated way on a single physical interface. Players can also edit and alter each other's structures once they have been drawn, leading to a playful interaction between the performers. Further, the table's larger format also lends itself to being seen directly by an audience as well without the need for an external projection screen.

4.3.3.2 Challenges

With the multi-touch implementation, it is easier to generate more strokes in a given time—multiple fingers can be in contact with the surface, dragging across it, instead of just one point of contact. The large number of strokes can quickly run up against the performance constraints of the system. At each moment when drawing, DS needs to determine if an intersection was created and take appropriate action if one was. This is an $O(n^2)$ algorithm, where each new linear segment needs to be intersected with all others. Optimizations are used to prune unnecessary calculations (like checking the bounding boxes of the two line segments to see if they overlap), but the main computation remains $O(n^2)$. The software can be seen to slow down and become somewhat unresponsive after many strokes have been created.

Also, in this multi-touch implementation, the drawing state is still selected using the keyboard. This poses two problems for the multi-touch implementation. One, the performer must move back and forth between the table and the keyboard, breaking the flow of performance. Two, since multiple points of contact can be drawing at once, and since there is only one drawing state at a time, all fingers will be drawing with the same kind (colour) of stroke. It may be desirable to allow the performer to draw with different

kinds of strokes at the same time. One option might be cycling through different stroke types or dividing the screen into regions, or associating a stroke type with each region.

4.3.4 Alternate Touch Screen Implementation

In addition to the multi-touch version described above, another touch-screen version exists for Android tablets and smartphones, ported by Stephen Sinclair of the IDMIL. Photos of the running application can be seen in Figure 4.16.

Though this version runs on multi-touch tablets, it currently only supports single-touch interaction since it was based on the single-touch DS codebase. The code to allow multiple concurrent drawing cursors described above could be integrated into this version to allow multi-touch interaction.

In terms of the programming changes that were necessary, the drawing code in the application had to be converted from the *OpenGL* API to *OpenGL ES*, which is the version for embedded systems that is used on Android. Also, the audio code needed to be interfaced with Android's native audio subsystem instead of the `RtAudio` system used in the standard version of DS. Finally, the system needed to be packaged for Android by including the appropriate XML metadata for the application and interfacing with its core Java components.

An interesting interface addition that Stephen included was a simple palette for selecting the drawing mode. This is necessary on tablet computers since they do not have a built-in keyboard, rendering the DS keyboard commands unusable. The silent stroke can be selected by tapping on the blue square (top); the removal state can be selected by tapping on the grey square (middle); and the sounding stroke can be selected by tapping on the red square (bottom). (Only one sound is available in this prototype version.) This is a partial solution to the mode selection issues noted in Section 4.3.3.2; while the user does not need to move to a separate keyboard, it does not address being able to draw with only one stroke colour at a time.

4.3.5 Discussion and Future Developments

As mentioned above, the default method of finding intersections is computationally expensive, and can slow down the system in a multi-touch context since many strokes can

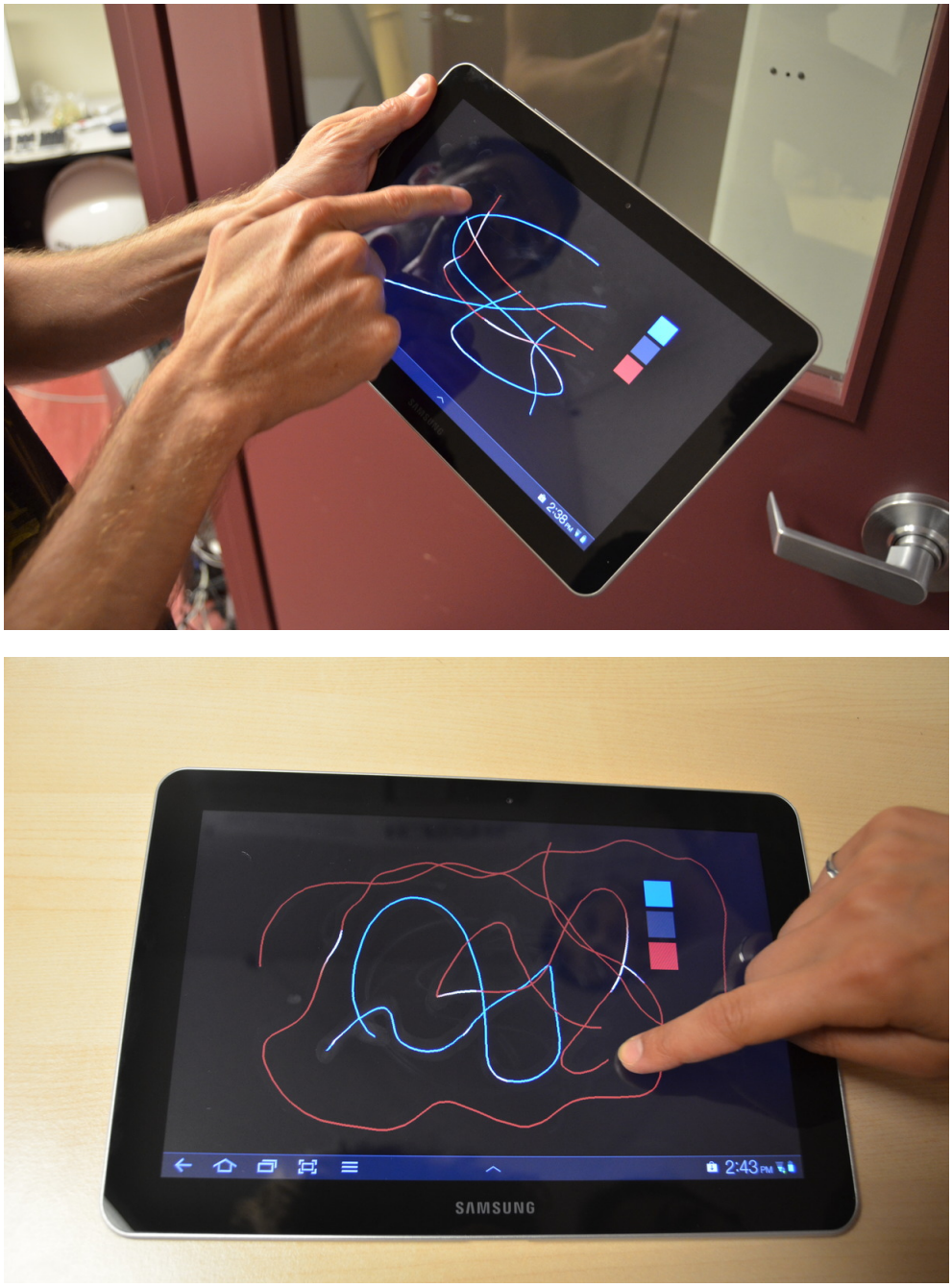


Figure 4.16 *Different Strokes* running on an Android tablet.

be drawn quickly. A more efficient intersection-detection algorithm could be investigated to alleviate the computational load.

New multi-touch gestures beyond "painting" could be investigated. For example, pinching to reorient the image may be interesting. A specific challenge will be to distinguish between these gestures and gestures intended for drawing.

In this vein, user testing could be done to validate the usefulness of the system and to characterize its properties in comparison with the default *Different Strokes* setup.

4.4 Summary

In this chapter, we have presented the design of the original *Different Strokes* application, as well as new extensions to DS. We described features designed for use in the context of the *d_verse* project, and the extension of DS for use on a custom multi-touch table. We have also described some new observations that we can make about the design of DS that were revealed by adapting it to be used in these contexts. In the following chapter, we describe another extension where *Different Strokes* is controlled with a haptic input device, allowing the physical sensation of the interaction to be manipulated.

Chapter 5

Exploratory Study Using Haptic Feedback

One goal in porting *Different Strokes* to various hardware platforms and usage contexts is to examine how input hardware differences may change the way in which the system is used. This information could be useful to performance software designers when they design their mappings and decide which input gestures they will use to drive their own systems.

A force-feedback device was integrated into DS to be able to experiment with varying haptic contexts and examine their influence. This serves as a convenient testbed for changing the physical parameters of the interaction. This chapter describes the implementation of this project, along with an exploratory user test that examined some of these effects.

5.1 DIMPLE integration

Haptic technologies provide an excellent opportunity to experiment with various possible physical characteristics of an input device. A force-feedback device can be made to feel different to the user by making changes in software, without necessitating any physical changes to the device itself. This provides us with fertile ground on which to experiment with consequences of the physical characteristics of the input hardware.

In order to conduct the experiment presented in Section 5.2, a haptics controller was

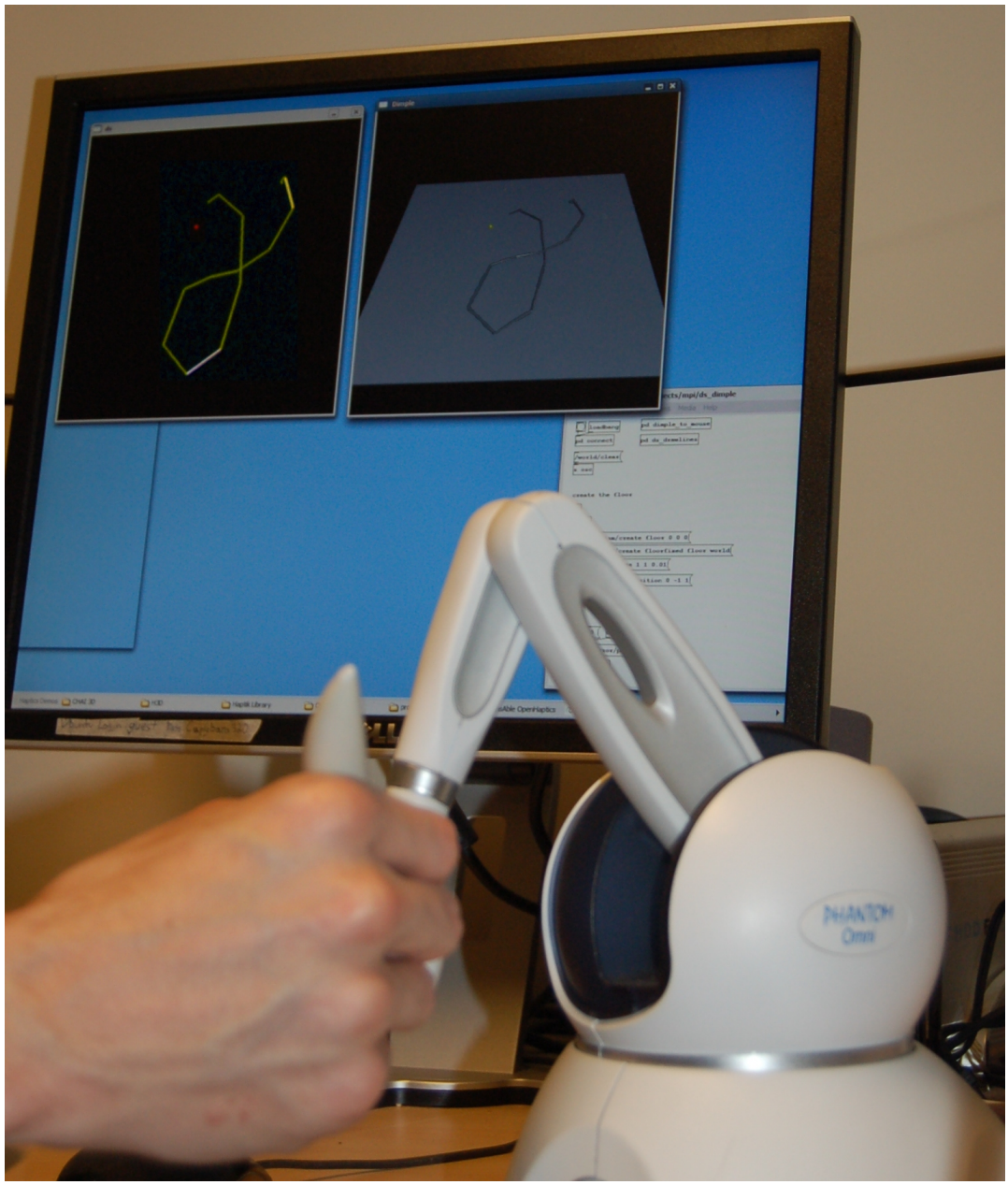


Figure 5.1 The prototype's physical setup (Zadel et al., 2009). The user holds a SensAble Phantom Omni device and manipulates it, feeling the forces controlled by the software. The image on the screen shows the three software applications used in the implementation: *Different Strokes* (at the left), DIMPLE (in the middle), and Pure Data (lower right). Note the flat square polygon in the DIMPLE window: this acts as a writing surface that the user can feel through the haptic device.

integrated into the *Different Strokes* environment, necessitating various software changes (Zadel et al., 2009). Our setup (Figure 5.1) includes three pieces of software working in tandem—*Different Strokes*, DIMPLE, and Pure Data—and a SensAble Phantom Omni for haptic display. This implementation was done in close collaboration with Stephen Sinclair, of the IDMIL. A system diagram illustrating the integration of DIMPLE into *Different Strokes* is given in Figure 5.2.

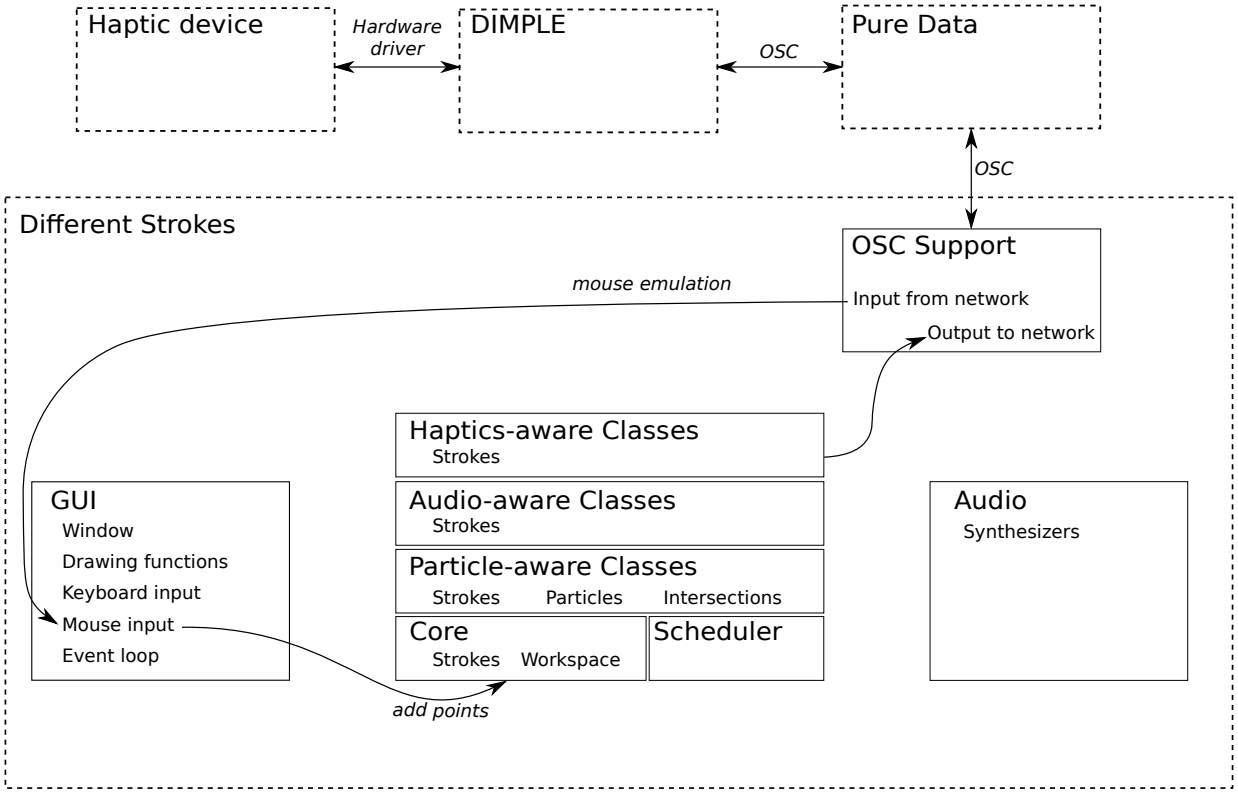


Figure 5.2 A system diagram for DIMPLE integration into *Different Strokes* to support haptic input devices.

The ultimate goal of integrating force feedback into *Different Strokes* was to determine whether particular types of haptic feedback can affect the experience of performing. Could certain forces and textures promote expressive, musical playing?

5.1.1 Previous Work

Haptic devices have been used previously in musical contexts. For instance, O'Modhrain (2000) showed that the presence of force feedback can improve performance in electronic

musical interaction, using a 2-DOF haptic device to interact with a virtual Theremin and violin model. Most relevant to our project, Rodet et al. (2005) implemented a musical 'game' using a 6-DOF haptic device simulating a virtual drawing surface. During development they performed an informal comparison between the use of a graphic tablet and the haptic device, allowing users to draw freeform representations of presented stimuli. This study made some interesting observations on the relationship between curvature and drawing speed during expressive unconstrained gestures: in particular, that, for expressive purposes, subjects would sometimes perform gestures that were contrary to the laws governing the relationship between velocity and curvature that the authors expected (e.g., as discussed by Viviani and Terzuolo (1982)). In general they found agreement between usage of the graphic tablet and the haptic device, though not many details in this regard were presented.

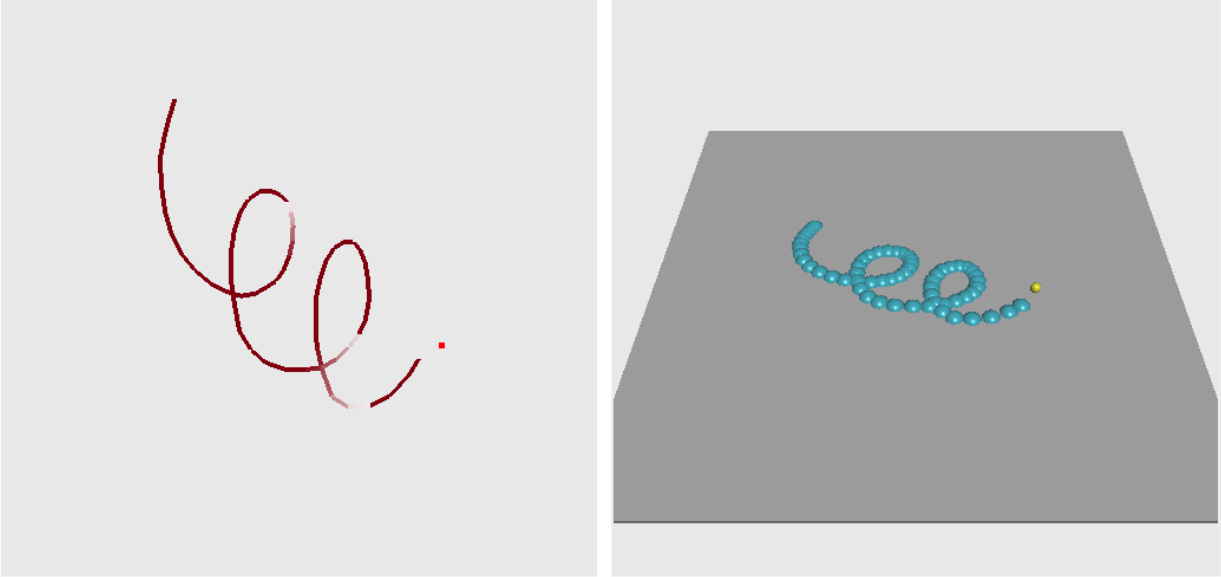


Figure 5.3 The synchronized *Different Strokes* (left) and DIMPLE (right) applications.

5.1.2 DIMPLE description

For prototyping this system, we made use of DIMPLE (Sinclair and Wanderley, 2009), a haptic simulation environment that can be controlled through Open Sound Control (OSC) messaging. The purpose of DIMPLE (the Dynamic Interactive Musically Physi-

caL Environment) is to allow experimentation with physical dynamics in a 3D simulated environment that includes haptic feedback. The system is designed to ultimately drive sound synthesis, and can be considered musically-oriented due to its OSC control interface.

DIMPLE provides a scene graph that consists of simple 3D objects of different shapes and sizes, and a spherical cursor that represents the location of the haptic device. These 3D objects can be subject to various physical forces (such as collisions, gravity and friction). These basic components are combined to make complex composites. Further, the objects can be related by various kinds of spatial constraints like joints and springs. DIMPLE's rigid body dynamics functionality allows these objects to interact.[1] For haptic rendering, calculations of collisions and penalty forces are performed at a rate of 1 kHz.

The environment can be probed using a haptic device. The cursor is shown in the graphic rendering of the environment, and the user can control its position in 3D via the device. The cursor is used to "touch" the objects in the environment: when it contacts an object, the user can feel physically accurate haptic feedback through the device, as though she was touching real objects in space.

DIMPLE brings together a number of libraries to help implement these features: CHAI 3D (Conti et al., 2005; haptics device handling and force effects), ODE[2] (rigid body dynamics), LibLo (Harris, 2012; OSC communication), and GLUT and FreeGLUT[3] (3D graphics via OpenGL)

5.1.3 Synchronizing the workspaces

The task was to create a DIMPLE environment that matched the DS workspace, and to use the haptic cursor as the DS pen. This implied bidirectional real-time communication between the two programs, which was mediated using a simple Pure Data patch.

The patch was designed to interface between DS and DIMPLE, constructing virtual objects on the fly to match the strokes in DS. This created lines which could be felt haptically on a virtual drawing surface. This surface also featured programmable friction coefficients and other modifiable properties.

[1] Since the DS workspace is static, however, this feature was disabled.
[2] http://www.ode.org/
[3] http://freeglut.sourceforge.net/

5.1.3.1 DIMPLE to DS

In DIMPLE, a large, thin prism object was created to represent the drawing plane, oriented horizontally to mimic a physical drawing tablet. The haptic cursor position was requested from DIMPLE at regular intervals, and this 3D position information was converted to messages for DS representing pen movements, with positions converted to a normalized 2D coordinate system. The vertical axis was used to determine pen up/down status: if the cursor was very close to the plane, it was considered to be in contact with it, and therefore the user was understood to be drawing.

DS was configured to receive the following messages:

```
/pen/up
/pen/down <float:x> <float:y>
/pen/move <float:x> <float:y>
```

DS then interpreted these messages exactly as though they originated from a mouse or drawing tablet.

OSC support in DS was implemented using the `liblo` C library (Harris, 2012). As mentioned in Section 6.1, *libmapper* could also have been used to send and receive messages between the applications, but the present DS/DIMPLE integration work was done prior to adding *libmapper* support to the DS code.

5.1.3.2 DS to DIMPLE

On pen-down, DS initiates the construction of a *polyline,* a data structure consisting of several line segments appended one after the other. To represent each line segment in DIMPLE, both a cylindrical mesh and a series of spheres were experimented with. These were placed in the drawing surface, protruding by some variable amount. A cylinder was created for each line segment, centred between two points and with a rotation calculated to match the segment's angle, resulting in a snake-like appearance in the DIMPLE 3D view. In the case of using spheres, they were constructed to overlap at intervals along the length of the line segment. Since the objects were created dynamically while drawing, as the haptic cursor looped back on its own trajectory, it would temporarily produce a bump as it collided with previously constructed objects. A visualization of this interaction for the case using spheres can be seen in Figure 5.3.

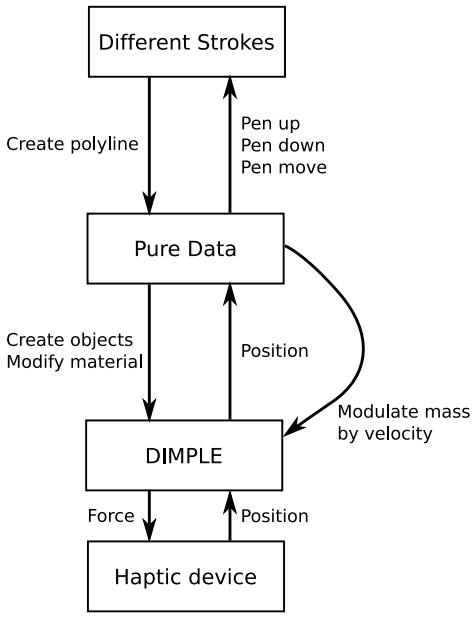


Figure 5.4 Information flow between the DS and DIMPLE applications.

DS was configured to send the following messages, which Pure Data used to generate corresponding OSC instructions for DIMPLE:

```
/ds/stroke/start
/ds/stroke/end
/ds/stroke/add <float:x> <float:y>
```

5.1.3.3 Pure Data Patch

DIMPLE reports the haptic device's current 3D cursor position with OSC messages such as

```
/world/cursor/position 0.2 -1.0 0.77
```

Different Strokes works in 2D, and expects OSC messages of a form that has a one-to-one correspondence with pen/mouse signals:

```
/pen/down 0.1 -0.31
/pen/move 0.12 -0.35
/pen/up
```

The current haptic device position needed to be converted to these pen-oriented messages.

We decided to use Pure Data (Pd) to do this conversion and to act as an intermediary between DIMPLE and DS. Pd is a good prototyping environment, and has good support for text list manipulation and Open Sound Control. Also, its GUI capabilities make it useful for quickly creating interfaces to control DIMPLE's overall state. These transformations could have been done in the C++ source code of the two applications, but adding this middle layer made it easier to prototype the system.

The Pd layer receives OSC messages from DIMPLE, manipulates them, and sends the result out to DS as OSC. It works similarly in the opposite direction, from DS to DIMPLE.

The messages from DIMPLE to DS were transformed by changing the coordinates from the 3D coordinate system to the 2D coordinate system. The patch also determines when the pen is considered to be drawing by detecting when the cursor comes within a certain small distance of the vertical drawing plane in the 3D environment, and then transmits "pen up" and "pen down" messages to DS.

The Pd patch was used also to control the overall state of the DIMPLE environment for use during the experiment. For example, the Pd patch provides buttons and number boxes for changing parameters and for initialization. It is used to set up the scene's initial geometry (i.e., the horizontal drawing plane), as well as change the haptic effects and their parameters. The buttons dispatch OSC messages to DIMPLE that modify the corresponding parameters.

The patch takes care of creating the trails of spheres and cylinders in the DIMPLE environment that act as a 3D extrusion of the drawn strokes. These strokes can then be felt with the haptic device when that geometry is touched.

Finally, the Pure Data patch contains logic used in the "modulated inertia" condition in the experiment, described below: it computes the velocity of the cursor from its changing 3D position, and sends out messages such as `/world/cursor/mass 0.0123` to update the cursor's mass as appropriate.

5.1.4 Description of Haptic Effects Used

For the purposes of the experiment described in Section 5.2, the experience of drawing with a graphic tablet was emulated. The DS drawing plane is modelled in DIMPLE as a virtual wall that is oriented horizontally in space, mimicking a standard graphic tablet. The pen was considered to be drawing when it came into contact with this horizontal plane.

The haptic properties of the DIMPLE cursor and the virtual surface were experimented with. Specifically, properties and effects investigated were static friction, dynamic friction, cursor inertia, and viscosity. As mentioned previously, virtual objects were also used to mirror the DS line segment data structures, so that the line segments themselves could be felt. The haptic effects that were used are summarized in Table 5.1.

Static friction Static friction is the component of the friction force that is present when the cursor is in contact with the drawing surface, but is not yet moving. It impedes the drawing tip from starting to move. Once overcome, there is no other friction present so the tip slides smoothly. High values of static friction lead the drawing surface to feel "sticky."

Dynamic friction Dynamic friction (also known as kinetic friction) is the component of the friction force that is present when the cursor is in motion along the drawing surface. It is proportional to the force causing the movement. High levels of dynamic friction cause the drawing surface to feel as though it is made of soft rubber.

Inertia Inertia is the tendency for an object to keep moving once it is in motion. We associated a variable mass to the drawing cursor. Small cursor masses (such as 0) make it easy to change the cursor's direction quickly; larger cursor masses (such as 0.015) make it harder to change the cursor's direction once it is moving since it has more momentum. The cursor masses were determined arbitrarily, by manually trying different parameters such that the inertia was perceivable, but not exaggerated.

We do not apply gravity to the cursor in the simulation. This means that the inertia felt is different than what would be felt moving a real physical object with a comparable mass, which would be subject to gravity. With noticeable inertia, the cursor feels as though it is "floating."

Modulated Inertia We implemented a non-physical model of cursor mass as well, where the mass of the cursor was updated continuously as a function of the drawing speed. High speeds would reduce the mass, reducing the inertia felt, and low speeds would raise the mass.

Viscosity Viscosity in the environment is a general damping factor on the cursor movement. It feels as though the cursor is moving in a bath of thick liquid.

Virtual wall (drawing plane) The drawing plane is implemented as a virtual wall, and feels like a firm surface. Its hardness is related to the frequency of the haptics simulation and the physical properties of the haptic controller. Controllers with less physical inertia and simulations with higher sample rates lead to stiffer walls.

The CHAI 3D library (Conti et al., 2005) is used for rendering most of the haptic effects. CHAI uses the "god object" method (Zilles and Salisbury, 1995) to implement collision detection and penalty forces, and the "friction cone" method for implementing friction effects (Harwin and Melder, 2002). The inertia effect is implemented as a spring-coupled mass, and was written by Stephen Sinclair. Virtual walls are implemented using standard techniques in the field (Salisbury et al., 2004). Modulated inertia was implemented in the Pd patch by computing the speed of motion and sending messages to DIMPLE to change the cursor's mass accordingly.

5.1.5 Effect on DS interaction: Qualitative Observations

While an exploratory usability study is presented in Section 5.2, this section describes the general setup of the force simulation and our qualitative impressions of the corresponding gestural implications.

Being able to feel the drawing plane as a flat virtual surface was comparable to the sensation of using a normal graphic tablet, but with a completely frictionless character. Simulating this surface virtually freed us to easily alter tactile and force properties of the plane in ways that are not possible with a physical tablet.

Static and dynamic friction seemed to have different effects. Dynamic friction allowed creation of sharp corners in the stroke and made it easier to stop the pen suddenly. In contrast, static friction only affected the beginning of the stroke, but made it easier to immediately start a stroke at a high speed.

Overly large friction coefficients interfered with drawing, but moderate settings enabled improved control as compared to a completely frictionless surface. Viscosity had similar effects to dynamic friction, both seeming to increase control over drawing speed. They also helped in aiming the stroke towards a target, making it easier to create an intersection at the desired location.

Static friction	Affected beginning of stroke; easy to start at high speed.
Dynamic friction	Sharp corners; easy to stop suddenly.
Cursor inertia	Enforced large curvature.
Variable cursor mass from drawing speed	Good scribbling, but also good control at high speeds.
Viscosity	Similar to dynamic friction; increased control over drawing speed.
Virtual drawing plane	Same as physical plane, but frictionless.
Feeling intersections	Cylinders or spheres; feel the intersections as they're touched.

Table 5.1 The haptic effects implemented and our impressions of their gestural implications.

Adding inertia to the haptic cursor had, in a sense, the opposite effect to dynamic friction: it reduced control accuracy, enforcing large curvature of strokes. For example, it made it difficult to perform a "scribbling" action, instead promoting the creation of round shapes and making it easier to maintain a constant drawing speed. This implies a low-pass filter effect: high-frequency shaking in the cursor position will be impeded and thus smoothed out.

As a compromise between these two observations, the cursor mass was modulated to be inversely proportional to the drawing speed; moving faster would result in constant, controlled speed, but slowing down would still allow sharp corners. This seemed more successful, allowing quick scribbling but promoting accuracy at low speeds.

Finally, an effect afforded by DIMPLE's dynamic haptic scene construction was allowing the user to feel the shape of the strokes themselves. Creating stroke intersections is an important action in DS, so physically feeling when intersections are made could be valuable. When the line intersects with itself or another line, a small vertical bump is felt since the pen tip contacts the corresponding chain of objects (cylinders or spheres) embedded in the drawing surface.

A challenge with this setup was to make the haptic sensation useful but not distracting to the performer. When the cylinders protruded too high above the surface such that their walls were perpendicular, it impeded movement instead of giving a slight indication. By embedding the spheres and cylinders further down into the surface, a more subtle slope was created. This highlights the fact that it is possible for the haptic effects to detract from the drawing experience if they are not well parameterized.

5.2 Exploratory Force-Feedback Study

We performed an experiment where participants used *Different Strokes* via a force-feedback device. The haptic device allowed us to control the physical feeling experienced while drawing, and we collected participants' preference ratings and recorded each drawing gesture's position versus time. Data collection took place February–April 2011.

5.2.1 Goals and Hypotheses

The main goal of the study was to determine if and how differences in the physical input device affect a user's experience of the system. There were two main hypotheses:

- That the existence of haptic feedback would increase the participant's enjoyment of and engagement with the system, but that it would not matter which haptic condition was used. O'Modhrain (2000) indicated that the presence of haptic feedback can increase users' performance with a musical system, but did not find a difference between the various haptic effects used.
- That the dynamic friction condition would help participants maintain a steady speed as they drew. This is relevant to *Different Strokes* due to its mapping design, linking drawing speed and sample playback speed.

One factor we wanted to understand in particular was if the presence of a certain haptic effect would make it easier to have a steadier drawing speed. The drawing speed is one of the main determinants in the sound playback in DS (the others being audio sample selection and the starting position in the wavetable). The drawing speed is generally fairly noisy, and perhaps certain haptic effects would smooth out the gesture velocities.

5.2.2 Experimental Design

The participants were given a set of exemplary tasks to do with the system. These tasks were performed under a series of haptic conditions in order to see how the haptic feedback affected the participant's performance with a given task. There were three tasks and five haptic conditions, giving 15 trials in total. These were presented in randomized order to each participant.

In each of the tasks, the participants were not asked to complete a certain number of gestures or work for a fixed amount of time. They were allowed to perform the requested gesture in any manner until they were satisfied that they had formulated an impression of the combination of task and haptic effect. They were allowed to include ancillary strokes as well, like adding loops to feed particles into the structures they were drawing in order to hear them. While this approach fostered a generally playful and exploratory engagement with the tasks, the presence of spurious gesture traces made it more difficult to clean up the data during analysis.

The tasks were chosen to reflect common shapes and operations in DS, along with a more musically-relevant task to help maintain the ultimate focus on musical performance.

5.2.2.1 Tasks

Straight lines with constant speed: Participants drew as many straight lines as they wanted, in any direction, at any overall speed. The requirement was that the participants tried to maintain a constant drawing speed over the length of a single stroke. As mentioned, some participants made extra loops to play back the strokes they were drawing, which necessitated manually tagging which lines were relevant to the gesture analysis in post-processing (see Section 5.2.5.1).

Drawing loops: Participants simply drew loops in any manner they chose. This is a basic gestural primitive in the DS system.

Replicating a pre-recorded sound pattern: Prior to the experiment, a simple sound pattern was created with the DS system and the resulting audio was recorded. The sound example was a simple loop that crossed a second straight stroke, triggering both sounds periodically. Participants listened to this audio example and then

tried to recreate it in *Different Strokes*. The same example was played to all participants. They listened to the audio example first, then it was stopped, and then they tried to recreate the sound that they had heard from memory. Participants could listen to the sound example as many times as they desired, but at no point were they shown its visual representation.

Examples of the gesture trajectories for each task can be seen in Figure 5.5.

5.2.2.2 Haptic Conditions

The haptic effects that were used in the experiment are a subset of those presented in Section 5.1.4. As described above, all conditions included a haptically simulated horizontal surface for the participants to draw on.

"No haptics" For this condition, only the plane was used without any other haptic effects. We considered this to be the neutral condition.

Dynamic friction A dynamic friction model across the surface: friction could be felt while the cursor (pen tip) was in contact with the virtual surface and was moving. In this condition, there was no haptic effect felt when the cursor was not in contact with the surface.

Cursor inertia The cursor has a relatively high simulated mass, leading to inertia: the pen continued to move in the direction it was already moving in. This effect was present when the cursor was in contact with the surface, and also when it was in free space.

Modulated cursor inertia Pen mass was simulated as in the previous condition, but the mass was modulated as a function of the pen's speed: as the velocity went up, the mass went down. This effect was still present when the pen was not in contact with the surface.

Static friction Static friction was present on the surface. There was no effect when the pen was not in contact with the surface.

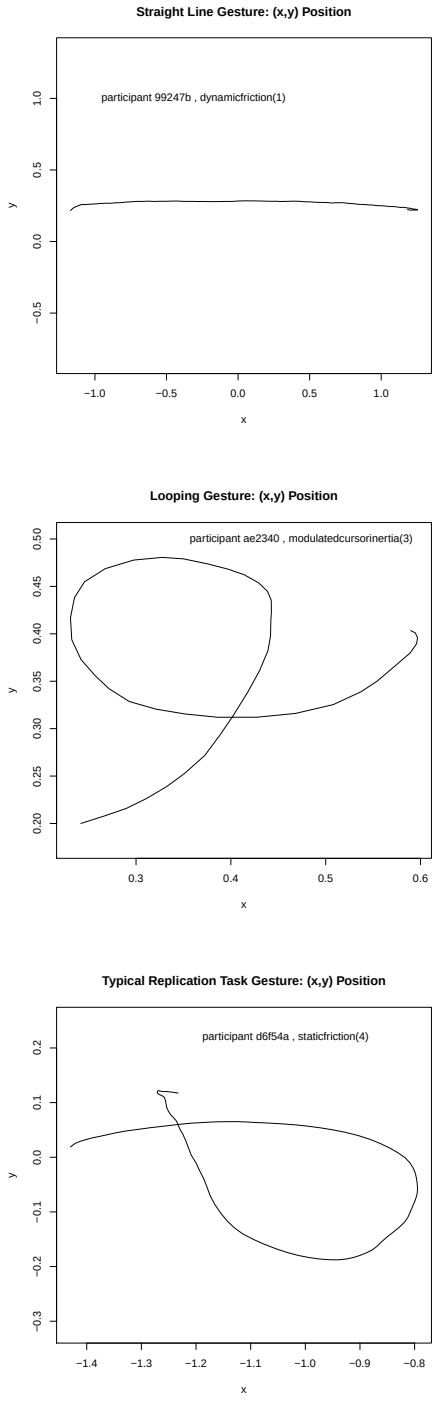


Figure 5.5 Typical stroke traces for each of the three experiment tasks in (x, y) screen space.

5.2.2.3 Questionnaire

We used a questionnaire to obtain preference ratings for each combination of task and haptic effect. Ultimately, the goal was to determine if there was a preference for a particular haptic effect in the context of using the software for performance, and if the participants felt that the haptic effect impeded their ability to perform the task. The questionnaire posed three questions for each task:

1. How easy was it to perform the task?

   1 2 3 4 5 6 7

   Very easy Very difficult

2. How enjoyable was it to perform the task?

   1 2 3 4 5 6 7

   Very enjoyable Not enjoyable

3. Did the haptic feedback influence your performance?

   1 2 3 4 5 6 7

   Improved performance Detracted from performance

These three questions were completed immediately at the end of each trial, so the performing trials and rating them were interleaved.

5.2.2.4 Gesture Recording

During each trial, all drawing gestures were recorded by *Different Strokes* and logged to text files. Incoming messages (pen down, pen up, (x, y) position) were time-tagged and logged as they were received by DS. Also, the start time of each trial was recorded, along with the times at which a new haptic effect was enabled. The motion of the cursor when it was not in contact with the drawing plane was not recorded. Details about the log file format are presented in Section 5.2.5.1.

5.2.2.5 Other Details

As mentioned above, the pen needed to be in contact with the virtual surface to draw. This was not completely obvious to all participants at the beginning of the experiment.

Two sounds were used: a percussive "factory" clip (2.27 seconds duration), and a recording of birds chirping (2.63 seconds duration).

5.2.2.6 Procedure

The procedure followed by the experimenter was as follows:

- The participant read and signed the ethics form;
- the three tasks were explained;
- the participant was reminded that they could take a break at any time and that they could opt out at any time;
- the participant was given as much free play time as they wanted to familiarize themselves with the system;
- the participant would perform 15 trials (five haptic conditions for each of the three tasks);
- as the participant was working, they would freely verbalize any comments they had about their experience and these would be noted by the experimenter;
- after completing each task, the participant would fill out the three questions rating their experience; and
- at the end of the experiment, the participant would be debriefed, and the experimenter would reveal the details about how the experiment worked and what was being tested (e.g., what the haptic effects were, what the actual shape of the sound pattern replication task was, etc.).

5.2.3 Participants

We had 17 participants (11 male, 6 female) with mean age 29.12 ($\sigma = 4.157$, min $= 22$, max $= 40$). At the end of the experiment they were required to fill out a one-page demographic questionnaire detailing their experience with music, computer use, video games, and laptop performance. The demographic breakdown was as follows:

- 6 classically trained musicians, 10 non-classical musicians, 1 non-musician;
- 10 software developers, 7 experienced computer users;
- 1 gamer, 8 casual gamers, 8 non-gamers;
- 4 people with laptop performance experience, 13 without.

All of the participants except one were right-handed.

5.2.4 Analysis of Questionnaire Data

In total, the participants performed 765 ratings (one for each of the 17 participants, 3 questions, 3 tasks and 5 haptic conditions). Overall, the ratings varied across the whole range of possible values (1 to 7). Looking at histograms of the data, they are not normally distributed. (They do not resemble a Gaussian distribution, nor are they unimodal.) Box plots of the data are shown in Figures 5.6–5.8. Each of these figures is divided into three parts, one for each task. Within each task, the box plots for each of the five haptic conditions are shown. In these plots, vertically higher ratings are more positive ("very easy," "very enjoyable," "haptics improved performance"). Each box summarizes 17 ratings, one for each participant.

An ANOVA is typically used for normal data to determine if there is a significant effect of the main variables in an experiment. Since the ratings are not normally distributed in our case, we must use non-parametric statistics. The non-parametric equivalent to an ANOVA is the Friedman test (Conover, 1980, pp. 299–308). The results of the Friedman tests are summarized in Table 5.2.

Question	Factor	Friedman test		
how easy	haptic cond	$\chi^2 = 3.1724$,	df = 4,	p = 0.5294
how easy	task	$\chi^2 = 10$,	df = 2,	p = 0.0067
how enjoyable	haptic cond	$\chi^2 = 2$,	df = 4,	p = 0.7358
how enjoyable	task	$\chi^2 = 6$,	df = 2,	p = 0.04979
haptic influence	haptic cond	$\chi^2 = 4.1404$,	df = 4,	p = 0.3873
haptic influence	task	$\chi^2 = 0.7778$,	df = 2,	p = 0.6778

Table 5.2 Friedman test results on the questionnaire data.

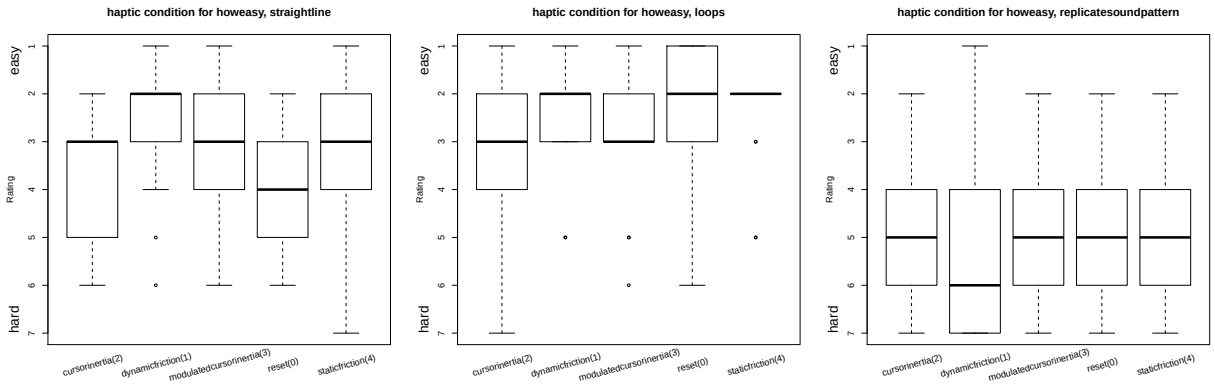


Figure 5.6 Box plots for the ratings for "How easy was it to perform the task?"

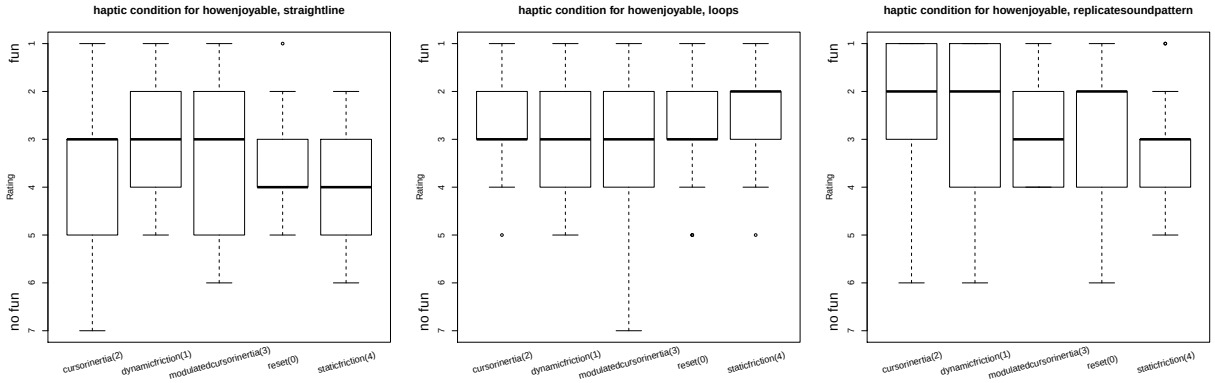


Figure 5.7 Box plots for the ratings for "How enjoyable was it to perform the task?"

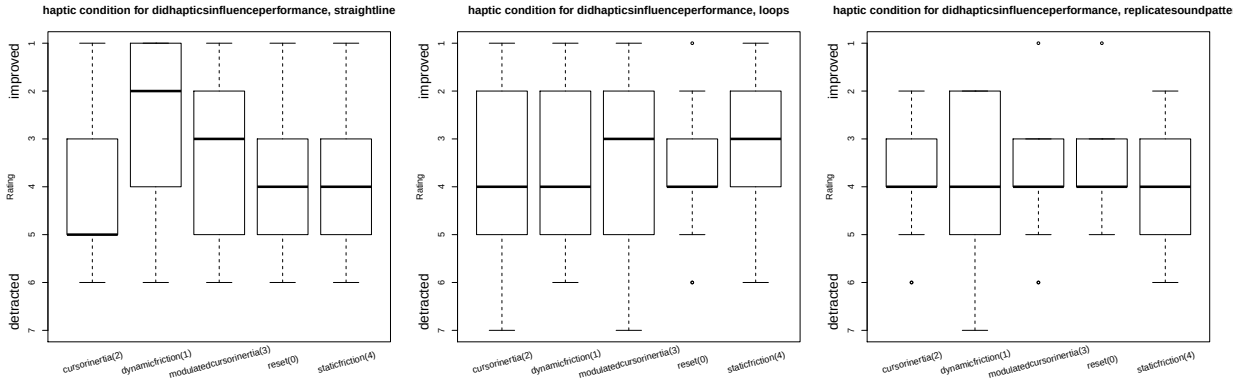


Figure 5.8 Box plots for the ratings for "Did the haptic feedback influence your performance?"

We see that for two of the questions ("How easy was it to perform the task?" and "How enjoyable was it to perform the task?"), the task had a significant effect. (We consider the p value to be statistically significant for $p < 0.05$.) In no other condition can we conclude that there is an effect. In particular, the haptic condition seemed to have no significant effect on the participants' ratings. This does not confirm our first hypothesis, that including haptics in the interaction would make the system more enjoyable and engaging.

If we look at the box plots in Figures 5.6–5.8, we can see that there are generally small differences between the distributions of the ratings over the haptic conditions. In Figure 5.6, we see that the scores for the replication task are generally lower (i.e., the task was more difficult). In Figure 5.7, the scores for the straight line task indicate that it was generally less enjoyable, and that the replication task was (perhaps surprisingly) slightly more enjoyable.

5.2.5 Analysis of Gestural Data

Data analysis was done on the gestural data recorded during the experiment. Only the data from the straight line task was used in these analyses since these data corresponded with our second hypothesis, that dynamic friction would improve participants' capacity to maintain a steady speed when drawing a straight line. We wanted to see if any of the haptic conditions would lead to a steadier stroke speed, as this is an important parameter that is used in *Different Strokes'* mapping to sound.

5.2.5.1 Data Preparation

During the straight line tasks, participants were allowed to make other kinds of strokes as well, such as loops. In *Different Strokes*, strokes do not make sound as they are drawn, but rather when a particle moves along their length. In order to be able to hear the strokes, some participants elected to add loops and other structures to generate particles that would then flow along the straight strokes. These participants would then have a better idea as to how even their drawing speed was. Other participants opted to approach the task as a purely gestural one without sound. Neither approach was imposed on the participants.

All of the strokes that were drawn during the straight line tasks were manually reviewed and were tagged as being straight (i.e. to be included in the analysis of straight strokes) or not. A small Python application was written that would show one stroke at a time, and was used to manually tag which ones were straight. We judged straightness by visually inspecting the 2D shape of each curve. We inspected 2189 strokes in the straight line trials and deemed 1271 of them to be straight (58.1%). In future experiments, allowing only straight strokes and using some other mechanism to introduce particles may be preferable to save this manual effort during data analysis.

The strokes were saved in comma-separated text files, with one line corresponding to each "mouse" event with a position and a time tag. An example can be seen in Figure 5.9; the comma-delimited columns from left to right are time (in UNIX seconds since epoch), event type, N/A, x position, and y position.

```
Starting log -- Tue Apr 19 14:36:49 2011
1303238230.642790,eventNewPolyLine
1303238230.642790,eventAppendPolyLinePoint,0,-0.426622,-0.332406
1303238230.644626,eventAppendPolyLinePoint,0,-0.426622,-0.332406
1303238230.660109,eventAppendPolyLinePoint,0,-0.425827,-0.328769
1303238230.677640,eventAppendPolyLinePoint,0,-0.426093,-0.327448
1303238230.695120,eventAppendPolyLinePoint,0,-0.427439,-0.328527
1303238230.712628,eventAppendPolyLinePoint,0,-0.428555,-0.327712
1303238230.730048,eventAppendPolyLinePoint,0,-0.427751,-0.323224
...
1303238240.944086,eventEndPolyLine
1303238242.789542,eventNewPolyLine
1303238242.789542,eventAppendPolyLinePoint,0,-0.231393,0.010923
1303238242.790709,eventAppendPolyLinePoint,0,-0.231393,0.010923
1303238242.806961,eventAppendPolyLinePoint,0,-0.216608,0.048671
```

Figure 5.9 An example of the log file format used in recording the stroke gestures.

5.2.5.2 Spline-fitting and Tangential Velocity Computation

From the recorded (x, y) position versus time curves, we computed the tangential velocity versus time for each stroke. Initially, these curves were produced by backward differencing, which was then smoothed with a Butterworth lowpass filter (cutoff 100Hz).

Even after smoothing, these velocity curves were still quite jagged, and were ultimately deemed to be too noisy to be used in the analysis. Also, some timestamp jitter was present due to the fact that the messages from DIMPLE were sent via a network loop-back port, through Pure Data.

After this initial attempt, improved tangential velocity curves were created by using B-spline fitting. The advantages of spline fitting are that the fit estimates the true gesture curve in the presence of noise, and that the resulting curves are differentiable. The `fda` (Functional Data Analysis) library in Gnu R (Ramsay et al., 2009) was used to fit splines to the data.

In order to compute a curve for the tangential velocity, the individual velocity curves for x and y were fit independently and combined. The tangential velocity profiles were computed point-wise from the smooth spline derivatives in x and y as follows:

- Perform independent fits to the x and y position curves;
- take the first derivative of each using the appropriate function in the `fda` library;
- sample each curve at identical time points (evenly sampled);
- at each point, combine the two velocities to compute the tangential velocity in the 2D plane ($\sqrt{(x')^2 + (y')^2}$).

5.2.5.3 Analysis

Figures 5.10–5.14 show typical gestures and their associated tangential velocity curves for each of the haptic conditions. Note that the tangential velocity profile is the magnitude of the velocity in the direction of drawing, so it is always zero or greater.

Figure 5.15 shows tangential velocity plots for one participant, for the "straight lines with consistent speed" task, for strokes of similar durations. Each of the five haptic conditions is shown. The characteristic shape for the static friction condition can be seen, where the cursor "sticks" at the beginning of the stroke until enough force is applied, and then it is quickly released. The acceleration phases differ in the dynamic friction and cursor inertia cases, where the drawing velocity increases more gradually. We can also see characteristic dips at the beginning and end of many strokes; these occur as the pen first touches and later leaves the simulated drawing plane. Note that in each of the

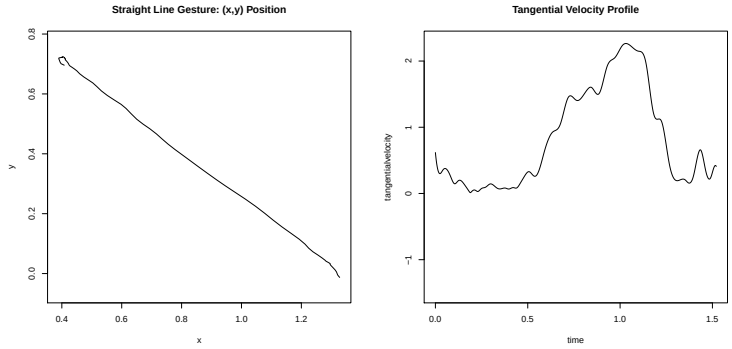


Figure 5.10 A gesture trace in (x, y) screen space, and its associated tangential velocity profile (participant 5cf1a6, straight line task, "no haptics" condition).

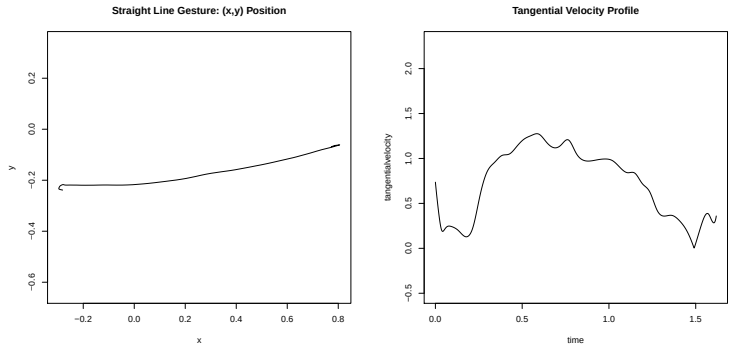


Figure 5.11 A gesture trace in (x, y) screen space, and its associated tangential velocity profile (participant 2d891a, straight line task, dynamic friction condition).

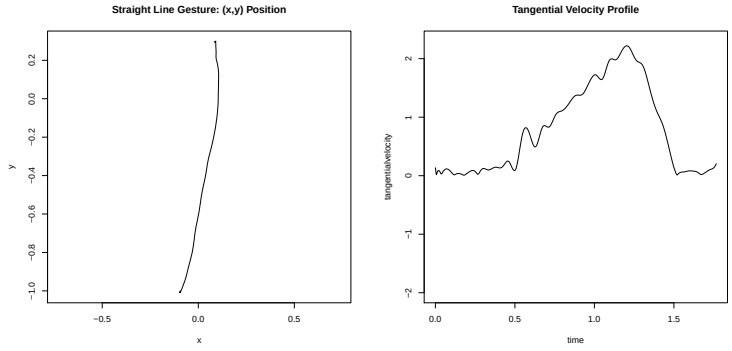


Figure 5.12 A gesture trace in (x, y) screen space, and its associated tangential velocity profile (participant c13fbc, straight line task, static friction condition).

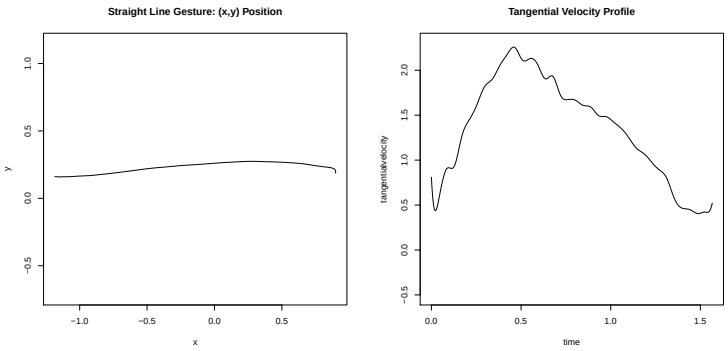


Figure 5.13 A gesture trace in (x, y) screen space, and its associated tangential velocity profile (participant 8b5d2e, straight line task, cursor inertia condition).

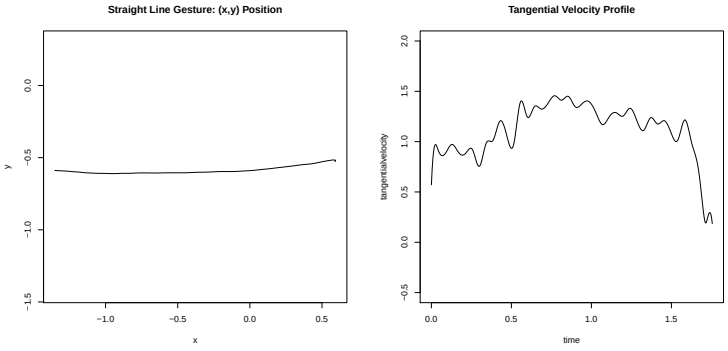


Figure 5.14 A gesture trace in (x, y) screen space, and its associated tangential velocity profile (participant c13fbc, straight line task, modulated cursor inertia condition).

cases, there is a fair amount of velocity variability in the central "steady" portion of the stroke.

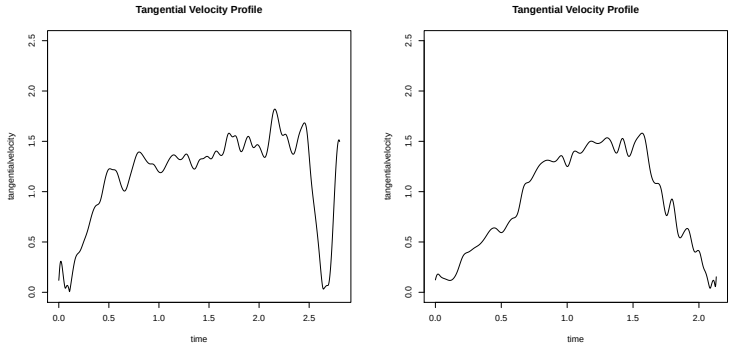
We can move on from the previous specific examples to take a more general view of the gesture curves. Figures 5.16 and 5.17 show what could be thought of as "functional box plots" for each haptic condition over different duration bins. The groups of velocity curves were first preprocessed as follows in order to be comparable:

- The given curves were normalized with respect to time. Curves of very dissimilar durations cannot be combined in this way since the evolution of the gesture should match between curves (hence partitioning into bins).

- The given curves were normalized with respect to speed. This was done by determining the average velocity along the middle section of the stroke and normalizing vertically so that the middle (steady) section would be centred around 1.0 units/second.

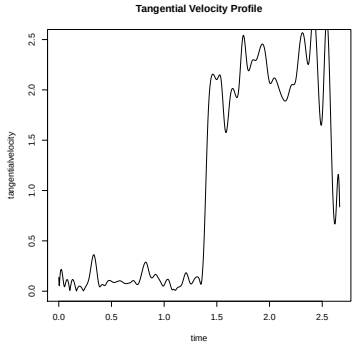
Once we computed and preprocessed the tangential velocity for each stroke, we computed the distribution of the velocity over time across the population of strokes. This was done by sampling all strokes in the population at a given time t to get a population of velocity values at that time. From these instantaneous velocity distributions, we plot the lower quartile, median, and upper quartile values over time. These give a sense of the evolution of the distribution of velocities over the course of a time-aligned set of strokes. Widely-spread moments in the distributions indicate increased velocity variability between participants at that point along the strokes. Narrower distributions indicate steadier (i.e., less variable) speeds. Also, plotting the quartile values does not assume the velocity distributions at each moment in time are normal, which they are not.

We can see visually that the haptic effects influence the drawing velocities for some stroke durations (figures 5.16 and 5.17). For most of the duration windows, the static friction distributions show more overall variability than the other haptic conditions. In particular, the static friction distribution taken over stroke durations 2.5–3.0 seconds shows the characteristic "sticking and slipping" at the beginning of the stroke (similar to Figure 5.15), and there is higher variability at the beginning of the strokes than for the other haptic conditions. The beginnings of the other static friction distributions also

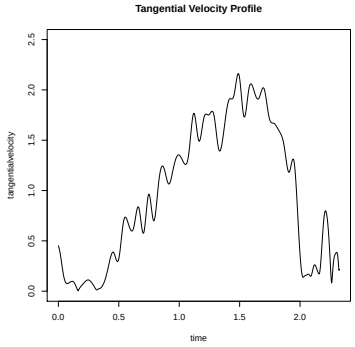


(a) "No haptics"

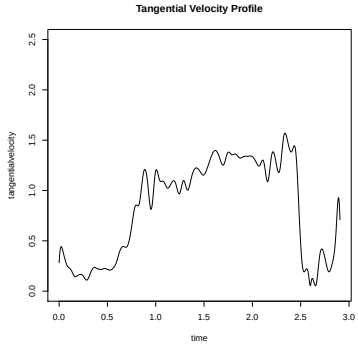
(b) Dynamic friction



(c) Static friction



(d) Cursor inertia



(e) Modulated cursor inertia

Figure 5.15 Selected individual tangential velocity curves for participant 16c29d.

show a narrow band of velocities, close to zero, which indicates the sticking portion of the stroke.

The dynamic friction condition for durations 1.5–2.0 seconds shows a higher variability than the other conditions at the beginning of the strokes.

The overall drawing speed appears to be a factor in velocity consistency. The shorter-duration strokes (0.5–1.0 seconds) tended to be drawn more quickly, and this can be seen in the velocity distributions. There is an overall consistency between the distributions for the different haptic conditions (though the static friction distribution is more variable), and the distributions show less jaggedness than for the longer durations. This stands to reason, since the longer strokes tended to be drawn more slowly and their velocities were thus susceptible to more jitter.

An equivalent indication of the evolution of the velocity variability can be seen by looking at the interquartile range, as shown in figures 5.18 and 5.19. The interquartile range is the distance between the first quartile and the third quartile of a distribution, and is an indicator of variability. We can see here again that the static friction condition shows more variability overall than the other conditions, especially for the 2.5–3.0 second duration window, and that the dynamic friction condition shows more variability at the beginning of the strokes over the 1.5–2.0 second duration window.

5.2.6 Participant Comments

During the trials, participants gave verbal feedback to relay their immediate impressions of the conditions. Here we summarize some of the themes in what was expressed.

The replication task was generally found to be difficult, albeit enjoyable. Participants were familiar with the sound palette, and the stroke topology for the task was quite simple (a loop of one sound triggering a straight stroke of other), but it still turned out to be complex for new users:

- "It would be helpful to have an example of the drawing for the replicate task."
- "The replication task is hard, regardless of haptics."
- "I can't tell what the topology should be from the sound."
- "The pattern one was tricky."

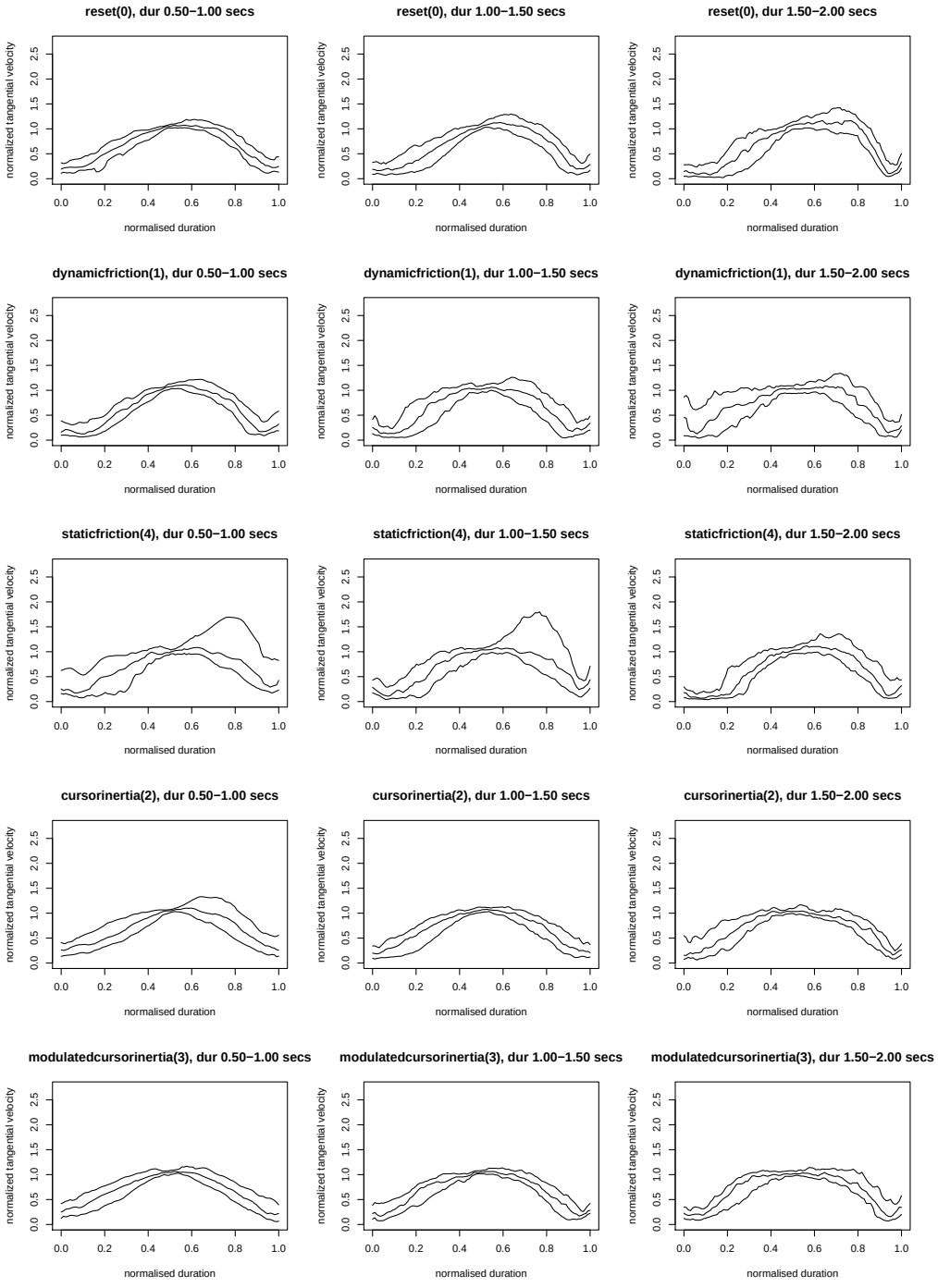


Figure 5.16 Velocity distributions for all five haptic conditions, over the duration intervals 0.5–1.0 seconds, 1.0–1.5 seconds, and 1.5–2.0 seconds. Each plot shows the first quartile, median, and third quartile for the distribution of velocities at a given point in time over the population of strokes. The stroke duration and velocity are normalized.

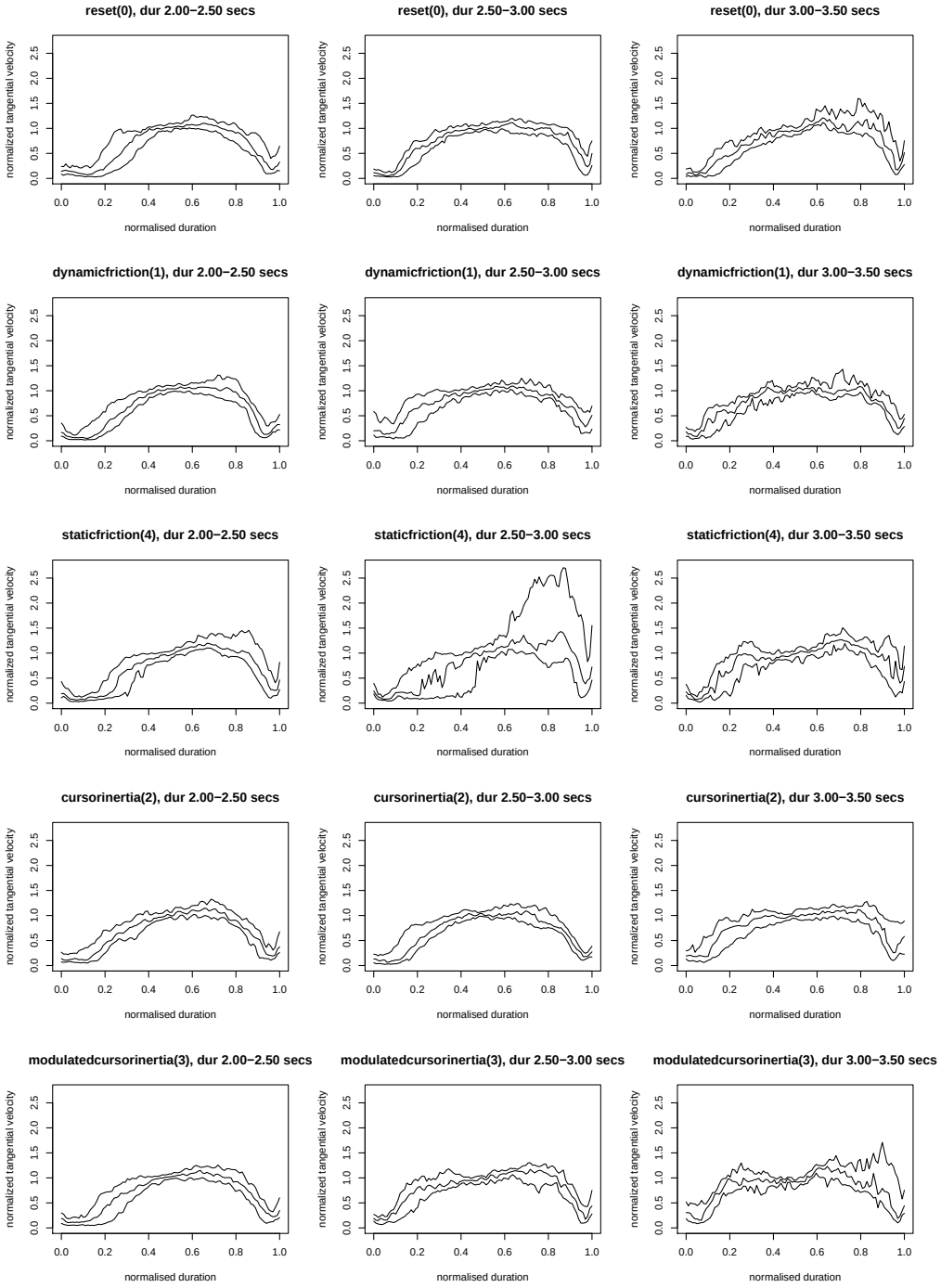


Figure 5.17 Velocity distributions for all five haptic conditions, over the duration intervals 2.0–2.5 seconds, 2.5–3.0 seconds, and 3.0–3.5 seconds.

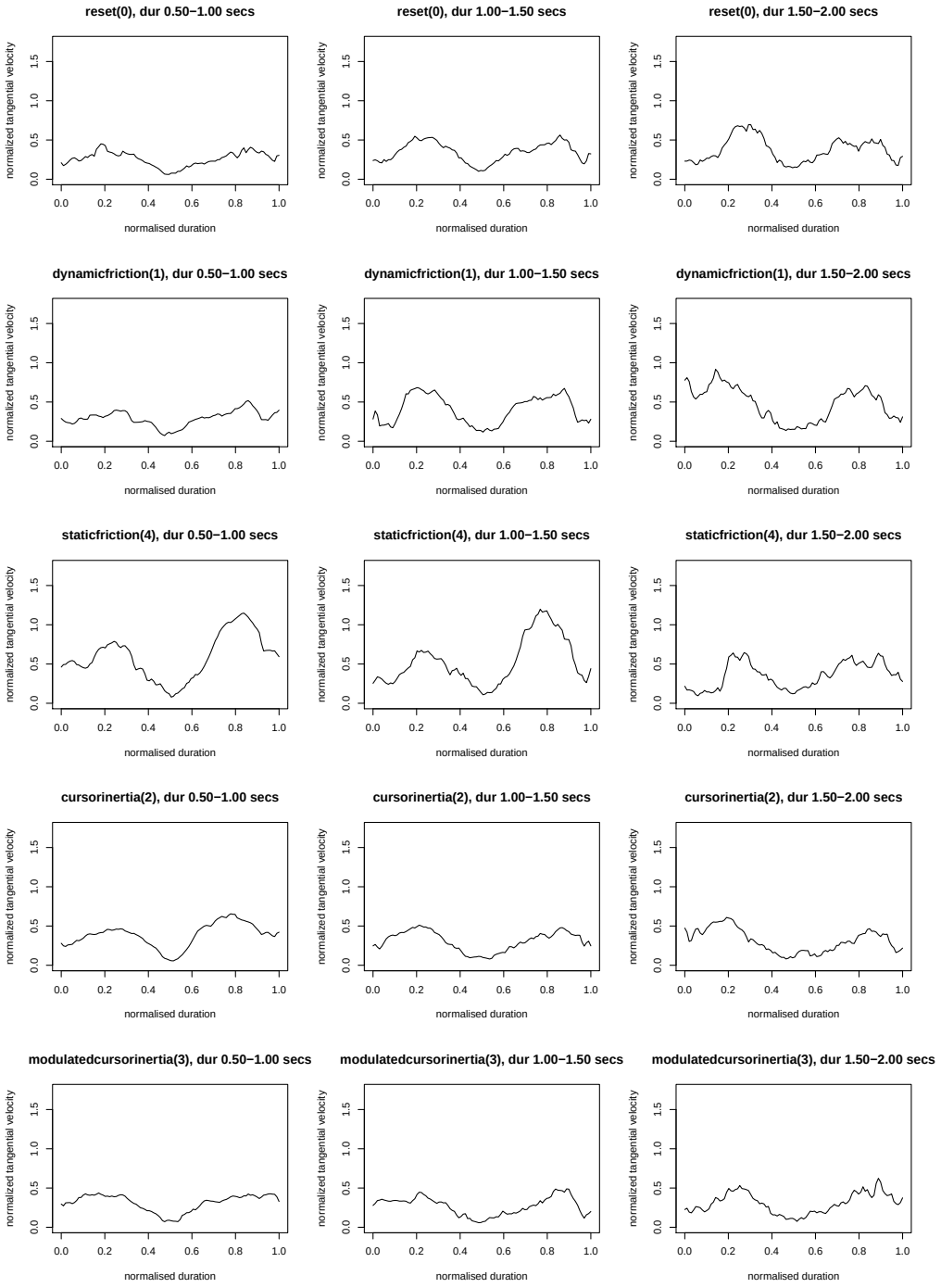


Figure 5.18 Interquartile ranges for the velocity distributions for all five haptic conditions, over the duration intervals 0.5–1.0 seconds, 1.0–1.5 seconds, and 1.5–2.0 seconds. The stroke duration and velocity are normalized.

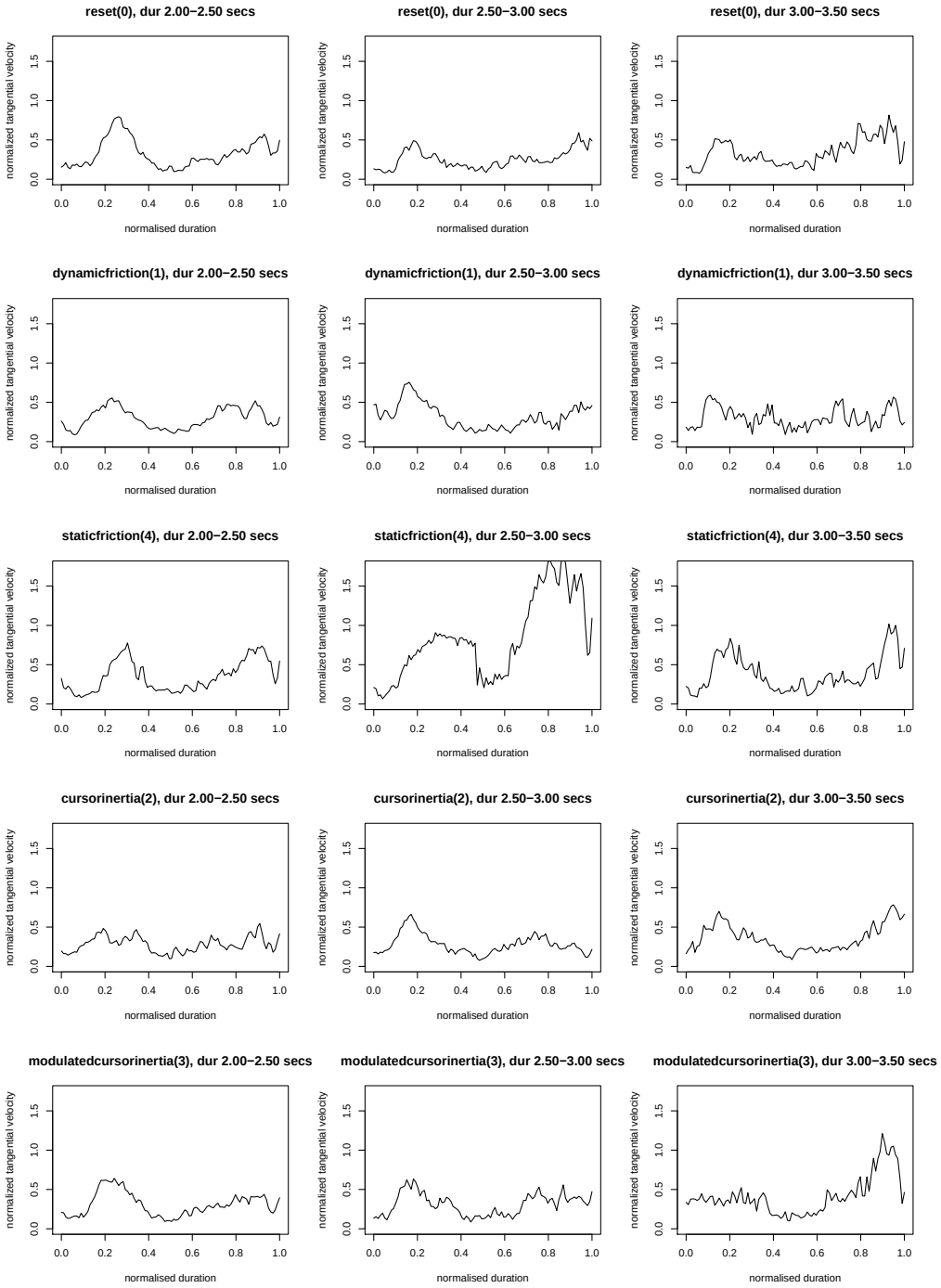


Figure 5.19 Interquartile ranges for the velocity distributions for all five haptic conditions, over the duration intervals 2.0–2.5 seconds, 2.5–3.0 seconds, and 3.0–3.5 seconds.

However, the task became simple once the participant discovered the correct topology:

- "It was hard at first, but it was easier when I figured out how to make the pattern."

Some participants felt that they were becoming more proficient at using the software as the experiment progressed. The inclusion of the free play period at the beginning of the experiment was intended to address this, but some participants still felt there was a strong effect of their increasing familiarity:

- "It takes time to learn the program [outweighing the haptics]."
- "I'm changing my approach with respect to rating now that I've done a few trials—there's a learning effect."
- "There is definitely a learning curve."

The questionnaire asked participants to rate how easy they felt a given task was, and if the haptics had an effect. Some participants were unsure as to how to calibrate their expectations:

- "We don't have a baseline for comparison—how easy in comparison to what?"
- "It takes a few trials to figure out what the scale is."
- "It is hard to say if it was easy when I don't know how I did."
- "I wanted to be able to A/B the haptics [i.e., actively compare haptic conditions]."
- "[For the consistent speed task,] I couldn't tell if I was doing it at a consistent speed since I couldn't hear the strokes."

Some participants found that their appraisals were more detailed than could be expressed in a single rating:

- "I can find different aspects of a task both easy and difficult, so it's hard to rate."
- "[For the consistent speed task,] haptics helped in the middle but hindered at the beginning/end of the stroke."

For some participants, there was a sense that the challenges of the tasks and learning the system outweighed the contributions of the haptic effects themselves:

- "I didn't feel that the haptic differences made the task too much easier/harder."
- "In the straight line case, it was easier to focus on the haptics since there was no sound."
- "I didn't notice the haptics since I was focused on the task."

The use of the virtual drawing plane to enable drawing was not obvious to all the participants. We intended for it to mimic a graphic tablet's drawing plane, but not all participants were comfortable with this design:

- "It's not intuitive to have to touch the plane to draw."
- "You're not sure when you'll hit the plane."
- "It's hard to control the start of a stroke since you don't know when you'll touch the virtual plane."

Participants expressed various opinions on the different haptic effects used:

- "[For the static friction condition,] is it normal for the pen to jerk at the start of a stroke? / I don't like the 'stickiness.' "
- "I like resistance in the haptics, but only up to a certain point."
- "I like / don't like the feel of this one because... [various reasons]"
- "It made it more enjoyable when *any* kind of dynamics was added."

Generally, participants found the system to be enjoyable, but some found it to be challenging in this configuration:

- "It's hard to control time."
- "It's fun."
- "It's a bit frustrating."

5.2.7 Discussion

From the analysis presented in Section 5.2.4, the effect of task on the "how easy" and "how enjoyable" questions was significant, but not in any of the other cases. It is surprising that from the questionnaire ratings, the haptic effects seemed to have no significant effect with respect to any of the questions. In particular, we would at least expect there to be a difference in the "no haptics" condition, confirming O'Modhrain (2000).

This may be due to the effect parameterizations, or perhaps the tasks themselves eclipsed the contribution of the haptic effects. In particular, the sound sample replication task was difficult for the participants since they had limited experience with the system. The free play time at the beginning of the experiment was meant to address this, but future experiments could be held over multiple sessions to control for the influence of the learning curve.

5.2.7.1 Potential improvements to the design

Letting the participants draw freely during the trials made it more laborious to segment out the strokes to analyze since we had to manually annotate straight strokes. Future experiments could include instructions to only allow participants to do the specific gestures needed.

There is an aspect of *Different Strokes*' design where the participant may not hear a stroke as he or she is drawing it. A stroke that has been or is being drawn starts sounding only when a new particle is introduced from elsewhere in the network of strokes. This means participants may have no auditory feedback when judging their drawing speed in the "straight lines with consistent speed" task, requiring them to rely on their internal kinesthetic sense of the motion. This specific design decision in DS could be seen as problematic for this task, since participants are ultimately controlling pitch without being able to directly monitor that parameter. For the purposes of this experiment and task we could find some additional way of conveying drawing speed that mimics the default mapping from drawing speed to pitch that we have in DS. For example, a pitched tone related to the drawing speed could be used so that participants could judge the speed in a way that is compatible with the mode of playing in DS. This would be a departure from the system's established design, but it would perhaps be useful in this experimental context. Another approach could be to play back the sound

associated with the stroke as it is being drawn. This poses a problem, however: the audio sample is stretched over the duration of the stroke, and so its overall pitch cannot be matched to the stroke duration as that is defined only after the stroke is drawn.

There were some bugs with the DS/DIMPLE/Pd setup: particles occasionally went missing, and sometimes did not loop properly even when an intersection was created. Multiple particles would also sometimes be triggered at the beginning of a stroke as the participant started to draw. This could be due to the physical dynamics that are introduced by the haptic device (e.g. slight bouncing when the pen touches the virtual plane might introduce unintentional micro-loops).

When participants started doing the ratings, they were not sure what the total range of the scales should be because they had not felt all of the haptic conditions and did not know what they should expect. (The initial free play phase for familiarization was done with the neutral, "no haptics" condition.) In future experiments, the participants could try all the haptic conditions beforehand during the experimentation phase, or the participants could be specifically instructed to use the full range of the Likert scales.

Each event's arrival time was recorded in *Different Strokes* by calling the UNIX `gettimeofday()` function. We can assume the network transmission and data processing added jitter to the recorded times. In future experiments, the position times might be recorded more accurately by adapting the driver for the haptic device to record the times immediately when the movement events are captured, locally at the device.

5.3 Conclusion

In this chapter, we have presented the integration of a haptic controller into the *Different Strokes* environment, along with an experiment designed to examine the influence of a set of haptic effects on the experience of using DS. Using a force-feedback device makes it convenient to experiment with simulated effects, mimicking the differences that can potentially be found between passive input devices.

An exploratory user test was performed using the haptically-enabled DS system. Users showed no clear preference for any of the force-feedback conditions used. Though there was not a particularly strong effect of the task, the replication task was found to be difficult but enjoyable, and the straight lines with constant speed task was rated to be less enjoyable. The analysis of the gesture data shows that there are gestural differ-

ences between the haptic conditions for the straight lines with constant speed task. In particular, the static friction and dynamic friction conditions led to increased velocity variability over certain stroke durations. Overall, the jaggedness of the velocity profiles shows that drawing with a constant speed may be difficult for users to perform. Thus, the appropriateness of mapping from drawing speed (as in *Different Strokes*) should be considered when designing new interfaces, depending on the desired effect. These observations serve to illustrate how physical differences in input devices can have gestural consequences, and may impact the user's experience with a graphical interface for music performance.

In the following chapter, we describe the adaptation of DS to use the *libmapper* framework, which permits us to easily reconfigure the input device that drives DS as well as its output to sound. Like the haptics extension, this improvement allows us to flexibly explore the differences between hardware contexts.

Chapter 6

libmapper Integration

This chapter describes an extension of *Different Strokes* to allow the easy interconnection of alternative input and output devices. This is useful for comparing input devices, and to allow non-technical users to reconfigure their setups. For example, a user might want to drive a new synthesis algorithm with DS, going beyond the built-in sample-based model. While it is of course possible to directly modify DS's source code to achieve this, artists might lack the necessary programming skills to do so. Also, for the purposes of comparison and validation, an easy method of reconfiguring the input and output would eliminate the extra work of developing custom software connections between the necessary system components. Being able to quickly reconfigure a given graphical software interface to be used in alternative hardware contexts encourages testing and comparison, as in the analysis approach that we propose in this dissertation.

This reconfigurability is achieved by adapting *Different Strokes* to use *libmapper*, a software library for communicating between musical devices over a network (Malloch et al., 2008). *libmapper* facilitates the easy interconnection of devices and mapping between their various input and output parameters. In the field of musical interaction, the term mapping typically refers to the deliberately-designed relationship between the outputs of a gestural controller and the inputs of the sound-producing synthesizer that is being controlled (Hunt et al., 2002). Mappings have a strong bearing on the overall feeling and behaviour of a digital musical instrument (Miranda and Wanderley, 2006, pp. 3–4). *libmapper* facilitates mapping design by establishing a protocol for automatic device discovery and interconnection, and taking care of many of the technical and practical

details involved. Its integration into *Different Strokes* allows DS to be easily reconfigured to use arbitrary input devices beyond the typical mouse and stylus.

First, we describe *libmapper* itself and explain how it was integrated into *Different Strokes*. Second, we describe the use of *libmapper* to connect DS to the *SuperCollider* media programming environment for performing audio synthesis. Finally, as one example of the easy hardware reconfiguration that this feature allows, we present an exploratory study where a Nintendo wiimote is used to control DS as an alternative to the usual stylus input.

6.1 *libmapper* integration

In the study of digital musical instruments, there is a common observation noting the separation between the input hardware device, which senses the performer's actions, and the synthesis process, which generates the sound digitally. In acoustic instruments, the performer's gestures are physically related to the sound production process, but this is not the case with their digital counterparts. We need to create a correspondence between the sensed playing gestures and the output sound, called a mapping (Hunt et al., 2002).

The instrument designer has to decide what this correspondence will be. This task can be complicated conceptually—how should a given gesture sound, considering both the audience and the performer?—and is further complicated by the logistics of managing multiple communicating hardware devices and software components, each with many input and output channels. The *libmapper* project (Malloch et al., 2008) was designed in the Input Devices and Music Interaction Laboratory to help to create and experiment with mappings quickly and efficiently.

We wanted to integrate *libmapper* into *Different Strokes* to be able to decouple DS's simulation model from the input hardware and output audio, exposing its internal logic and graphical representation. As mentioned above, we did this in order to be able to experiment more freely on the software and to use it in conjunction with other devices.

This part of the *Different Strokes* project was done after the original OSC integration for the haptics extension, as described in Section 5.1.3.1. *libmapper* could have been used for that project as well.

6.1.1 *libmapper*

libmapper allows the automatic discovery of *device* entities on the local network, and allows a user to make connections between the various control variables that these devices make available. A device can represent either a hardware controller, or a software application, such as a synthesizer. A *libmapper*-compliant device declares relevant metadata that describe its input and output control parameters. The system brokers mapping connections, considering these metadata and broadcasting administrative messages to all participating devices on the subnet.

The protocol was designed to ultimately make it simple for non-technical end users to create mappings between their devices and synthesizers. To this end, it uses predictable defaults and handles as much of the technical detail as possible, including automatic device discovery, resolving device name collisions, and sensibly converting between signal ranges. Further, the system's ethos is to impose no particular high-level interpretation of the meaning of individual signals, but rather to let the end user make those decisions based on their particular problem.

libmapper comes as a reference implementation (in the form of a C library) and a set of graphical user interfaces for making mappings. The protocol is implemented in terms of the popular Open Sound Control messaging standard. A notable feature of *libmapper* is that it uses IP multicast (called the *admin bus* in *libmapper* terminology) to broadcast administrative messages efficiently to all the participating entities on the network.

6.1.2 *libmapper* Terminology

As mentioned above, *libmapper* devices are individual entities that can communicate with each other. A T-Stick (Malloch and Wanderley, 2007), a joystick, a Max patch, or synthesis software are all considered devices. Each device provides one or more *signals*, which are sources or sinks of control data. Each signal is named explicitly, in plain language, allowing a human to easily understand its semantics and thus the affordances of a given device.

Output signals can be connected to input signals, so for example the `xposition` signal for a joystick can be connected to the `filtercutoff` signal in a synthesis patch. Once connected, signal data is transmitted from the source device to the destination device. The source and the sink normally reside on separate computers, and the data

is sent over the local network, though multiple devices running on the same computer are permitted. This communication is peer-to-peer: once the connection has been established, data is sent directly from the sending device to the receiving one, without passing through any intermediary nodes.

The data values flowing from an output signal to an input signal can be further modified by specifying a mathematical expression. For example, numbers flowing from an output signal can be scaled and offset to alter the correspondence between a sensor signal and a synthesis input. These expressions are typically specified via a *libmapper* GUI (Figure 6.1).

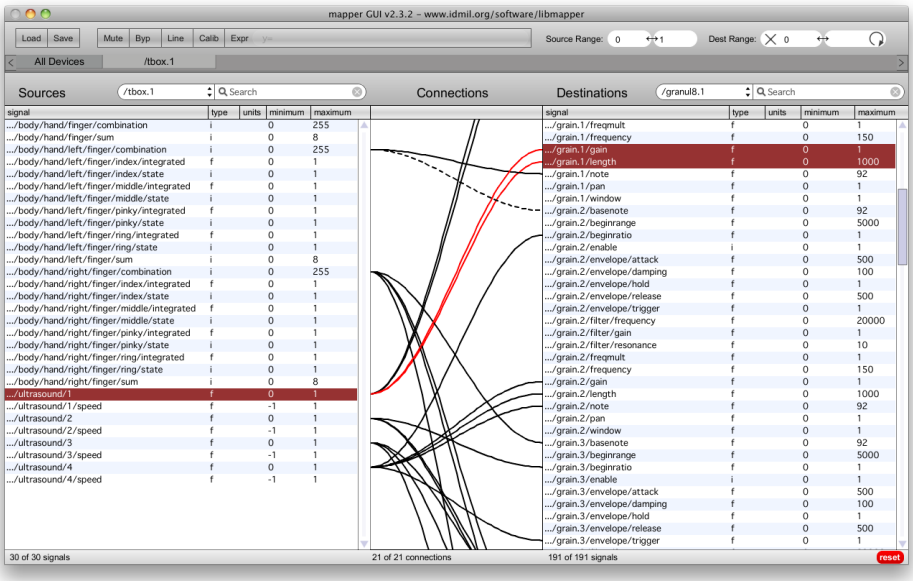


Figure 6.1 A screenshot of a *libmapper* GUI, showing the input and output signals for two devices. Output signals for a *tbox* are listed along the left, and input signals for a *granul8* software synthesizer instance are listed along the right. Connections between the signals are represented as curved lines from left to right. The properties of a given connection appear at the top of the screen when it is selected.

Each GUI is called a *monitor*, and monitors can be used to query the state of the network and to create connections between devices and signals. Many monitors can be used at once on a given network, and the *libmapper* protocol keeps them all synchronized with current information.

6.1.3 Instances

libmapper has an interesting feature for supporting *instances* of a given signal, for use in polyphonic systems. Essentially, the *libmapper* can include a unique ID number in messages for a given signal to disambiguate parallel instantiations of it. This allows, for example, mapping from multiple simultaneously engaged keys on a piano keyboard without having to manually declare unique signals for each key. Signal instances can exist for both output signals (e.g., on multi-touch surfaces) and input signals (e.g., on polyphonic synthesizers).

For example, a single contact point on a touch surface might transmit `/x` and `/y` signals for its x and y positions, respectively. A multi-touch surface, however, can support multiple points of contact. We could model this by creating separate signals, say `/touch1/x`, `/touch1/y`, `/touch2/x`, `/touch2/y`, etc., but that would require that signals be dynamically created and destroyed during performance, making it impossible to create stable mapping connections starting from those signals.

The instances feature allows a given signal to have multiple voices that are managed implicitly by the *libmapper* library. As new touches occur on the surface, they are assigned unique instance IDs and these are transmitted with the signals to distinguish between them. The signals received are thus of the form

```
/touch/x 142 0.3726
```

meaning that instance 142 is transmitting x position 0.3726. These instance numbers are transparent to the user and are handled automatically.

Another example could occur when mapping between a piano keyboard and a synthesizer. In order to handle chords, the keyboard side must register multiple keypresses, and the synthesizer must handle simultaneously sounding notes. The keyboard would declare this output signal:

```
/keyboard/frequency
```

which would report the pitch of a depressed note. The synthesizer would declare this input signal:

```
/synthesizer/frequency.
```

When a key is depressed, an instance would implicitly be allocated at the keyboard device, sending this message:

```
/keyboard/frequency 12 440.0
```

where 440.0 is the frequency of the note and 12 is the instance id. Upon receiving this message, the synthesizer would itself create a corresponding instance on its side, sounding a new note. When the instance is destroyed on the keyboard side, it sends a message to the synthesizer side to communicate that this has happened, and the synthesizer will destroy its corresponding instance, silencing the note. The reason that the instances feature is interesting is that multiple connections can be created and destroyed dynamically and implicitly during performance, obviating the need for the new connections to be manually managed.

6.1.4 Integration with *Different Strokes*

The main tasks in the *libmapper* integration were to call into the library from DS as appropriate, and to determine what the output signal namespace should be. The initial goal with the *libmapper* integration was to output enough control signals to be able to recreate the default *Different Strokes* mapping to sound with an external synthesizer (for instance, in a different physical location). This meant communicating when sounding playhead particles are created and destroyed, their play rate, and the sound file they are associated with. A system diagram illustrating the integration of *libmapper* into *Different Strokes* is given in Figure 6.2.

The `MPI_Mapper` class was implemented to encapsulate the interface between the *Different Strokes* code and *libmapper*. `MPI_Mapper` defines methods that can be called to send out a given message. When the simulation is `update()`ed, these output methods are called as the associated events are resolved. Doing this in the `update()` call means that the messages are dispatched at each animation frame. Though having a higher graphical framerate would thus mean that the *libmapper* messages would be sent out with more accurate timing, the current frame rate of 30 fps is sufficient for our purposes.

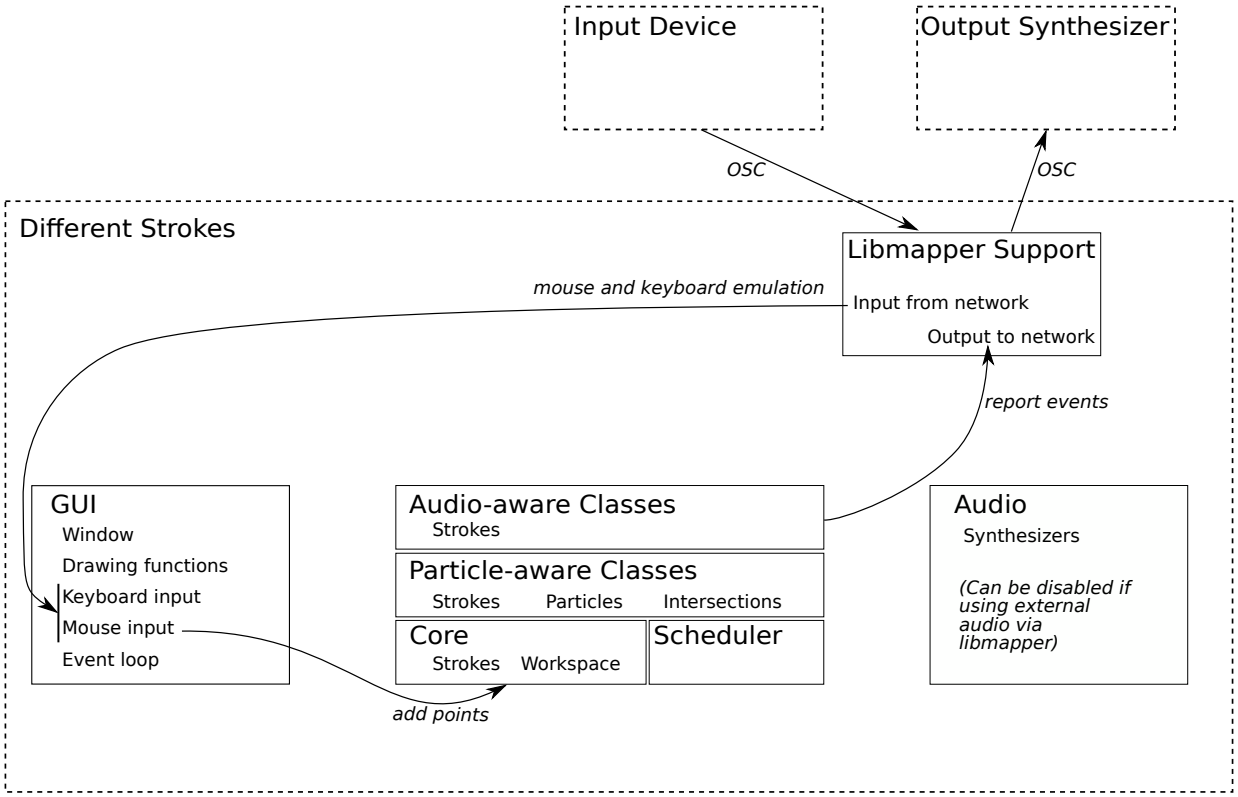


Figure 6.2 A system diagram for *libmapper* integration into *Different Strokes*.

6.1.4.1 Output Signals

In *Different Strokes*, there is a sound synthesizer associated with each stroke, so each stroke is a synthesis voice. The most-recently-added particle acts as the playhead. This means that as new particles are added, the playhead can jump around the wavetable abruptly (which was chosen deliberately for its aesthetic effect). The output signals that are defined for DS are

```
/differentstrokes/stroketype1/particle/position
/differentstrokes/stroketype1/particle/rate
/differentstrokes/stroketype2/particle/position
/differentstrokes/stroketype2/particle/rate
/differentstrokes/stroketype3/particle/position
/differentstrokes/stroketype3/particle/rate
...
```

where each stroke type refers to one on-screen colour, i.e. one entry in the list of selectable wavetables. These signals continuously transmit the current position and wavetable play rate for active playhead particles.

The particle position is normalized with respect to the arclength of a stroke, with the beginning of the stroke at 0.0 and the end of the stroke at 1.0. The rate is computed as the particle's speed along the stroke as a 1-dimensional parameter space, measured again by arclength. One particle per stroke can be transmitting these signals as there is only one active playhead per stroke.

Since there can be multiple strokes of a given colour on the screen at a time, and each one can potentially have an active playhead, we use *libmapper*'s instances feature to handle these parallel data streams. New particles are assumed to be new instances of the `position` and `rate` signals for the given stroke type (colour), and the appropriate instance ID is sent along with each message. The instance IDs are simply generated serially starting at zero as new particles are instantiated in the *Different Strokes* C++ code.

6.1.4.2 Input Signals

The following *libmapper* input signals are declared in *Different Strokes*:

```
/differentstrokes/pen/pos/x
```

```
/differentstrokes/pen/pos/y
/differentstrokes/pen/pos/z
/differentstrokes/drawingstate
/differentstrokes/removalstate
```

These signals parallel the usual events that are transmitted to DS via the stylus and keyboard: pen up/down, pen position, and state changes.

The three position signals were declared as operating on floating point data, though the `/differentstrokes/pen/pos/z` signal was programmed to clamp the signal at zero (meaning "pen down") or one (meaning "pen up"). The `/differentstrokes/drawingstate` signal accepts integers from zero to nine, corresponding to the wavetable number. The input to the `/differentstrokes/removalstate` signal is intended to represent the discrete event of engaging the stroke removal state, so it ignores its numerical argument and simply turns on stroke removal (i.e., is equivalent to the `r` key).

6.1.4.3 *libmapper* Namespace Design for *Different Strokes*

Figure 6.3 details a proposed namespace design for the signals that *Different Strokes* will expose. The design is intended to make it possible to implement the traditional mapping used by the DS system, but also to expose many other signals for designing new mappings.

The signals include input gestures measured at the input hardware (`/differentstrokes/input`), as well as the state of the particles as they move around (`/differentstrokes/particle` and `/differentstrokes/soundingparticle`). Some global signals are included as well.

6.1.5 External Synthesis Example Using *SuperCollider*

As stated above, one of the goals was to allow *Different Strokes* to be connected to external synthesis algorithms, recreating its default synthesis scheme. This section outlines this aspect of the project. The *SuperCollider* (SC) media programming environment was used. SC does not ship with *libmapper* support, so this was added first, and then an SC patch was written to emulate DS's sample playback.

```
/differentstrokes/input/pen/position/x floatpos
/differentstrokes/input/pen/position/y floatpos
/differentstrokes/input/pen/down floatpos
/differentstrokes/input/pen/up

/differentstrokes/particle/created particleid
/differentstrokes/particle/died particleid
/differentstrokes/particle/positiononstroke particleid arclengthposition
/differentstrokes/particle/screenpos/x particleid
/differentstrokes/particle/screenpos/y particleid

/differentstrokes/soundingparticle/stroketype1/created particleid
/differentstrokes/soundingparticle/stroketype1/died particleid
/differentstrokes/soundingparticle/stroketype1/voicestolen particleid
/differentstrokes/soundingparticle/stroketype1/positiononstroke particleid floatarclengthposition
/differentstrokes/soundingparticle/stroketype1/playrate particleid floatrate
/differentstrokes/soundingparticle/stroketype1/screenpos/x particleid floatpos
/differentstrokes/soundingparticle/stroketype1/screenpos/y particleid floatpos
/differentstrokes/soundingparticle/stroketype2/created particleid
/differentstrokes/soundingparticle/stroketype2/died particleid
...
/differentstrokes/soundingparticle/stroketype9/screenpos/x particleid floatpos
/differentstrokes/soundingparticle/stroketype9/screenpos/y particleid floatpos

/differentstrokes/stroke/addsegment strokeid xpos ypos
/differentstrokes/stroke/numparticles strokeid intcount

/differentstrokes/global/numparticles intcount
```

Figure 6.3 The proposed *libmapper* signal namespace for *Different Strokes*

6.1.5.1 *libmapper* Support for *SuperCollider*

As part of the *libmapper*/DS integration work, *libmapper* bindings were written for the *SuperCollider* computer music system. *SuperCollider* is a productive environment for rapid prototyping of synthesis algorithms, so being able to connect DS and SC opened up a wealth of possibilities.

The basic idea was to expose *libmapper*'s data structures and commonly-used functions in SC. Two *SuperCollider* classes were added: `MapperDevice` and `MapperSignal`, corresponding to *libmapper*'s device and signal constructs. `MapperDevice`s are allocated, have signals added to them, and can start polling for messages.

`MapperSignal`s can act either as inputs or outputs. Once an output signal has been created using `MapperDevice:addOutput`, it can be updated using `MapperSignal:update` to send out new values. For input signals, an SC callback function is registered in `MapperDevice:addInput`. This function is then called when the device receives a new value for the signal. Its arguments are the input signal's name and the newly received value.

The initial version of the *libmapper*/SC bindings included support for instances, and this version was used in the mapping from *Different Strokes*. The callback for each signal would receive three arguments: the input signal name, the instance ID, and the incoming value. The instance ID was used to distinguish between different strokes of the same colour (i.e., assigned to the same sound sample).

6.1.5.2 *Different Strokes* Synthesis Emulation in *SuperCollider*

Once the above bindings were written and SC could be mapped to and from external devices, we moved to the recreation of the DS synthesis model. DS's synthesis uses variable-rate wavetable playback. The playback supports starting and stopping at any point in the wavetable, and variable-rate playback as the sound plays. A sample playback synth was created in SC that could dynamically modulate the playback speed in an equivalent way.

The output signals from DS are described in Section 6.1.4.1. As mentioned, instances here correspond to simultaneously sounding copies of a given sound (i.e., disambiguating between multiple strokes of the same colour). The input signals to SC were defined as `/position` and `/rate`: the former for receiving the current position of the playhead

particle, and the latter for receiving its current play speed. A hash table mapping from instance IDs to running SC synths was maintained, and as new `/rate` values were received for a given instance, these were transmitted to the appropriate running synth. When an instance is killed, its associated wavetable player is stopped. This setup was sufficient to recreate the original DS mapping to sound.

The mapping scheme worked well, effectively recreating the standard DS sound aesthetic. Some slippage between the on-screen position of the sounding particle and the playback position occurred, however. This may be due to the network messaging; further work will be required in the future to investigate this issue.

6.2 Freehand Input Device Test: Using the Wiimote

DS is typically controlled with a stylus-like input device. Integrating *libmapper* into *Different Strokes* allowed us to easily connect arbitrary *libmapper*-supported controllers to DS, opening up the possibilities for experimentation on input devices and comparisons between them.

An informal feedback session was conducted to explore the suitability of Nintendo's wiimote video-game controller for drawing gestures and for use with DS. The DS interface was projected on a large screen in the IDMIL, and participants stood in front of it with the wiimote, as seen in Figure 6.4.

Seven participants were recruited to experiment with the system and give verbal feedback. All but one of the participants had prior experience with DS, giving almost all the participants a baseline for comparison. They were asked to interact with the system and to give feedback on the appropriateness of the wiimote for using DS, as well as comment on their overall impressions of the system.

6.2.1 Implementation Details

The *Different Strokes* interface was projected on a large screen, and the cursor was controlled by pointing the wiimote toward the desired location. The design of the wiimote interaction was made to closely mimic mouse control, using the trigger button on the wiimote to correspond to the mouse button. When the trigger was depressed, drawing would begin at the current cursor position, and would end when the trigger was

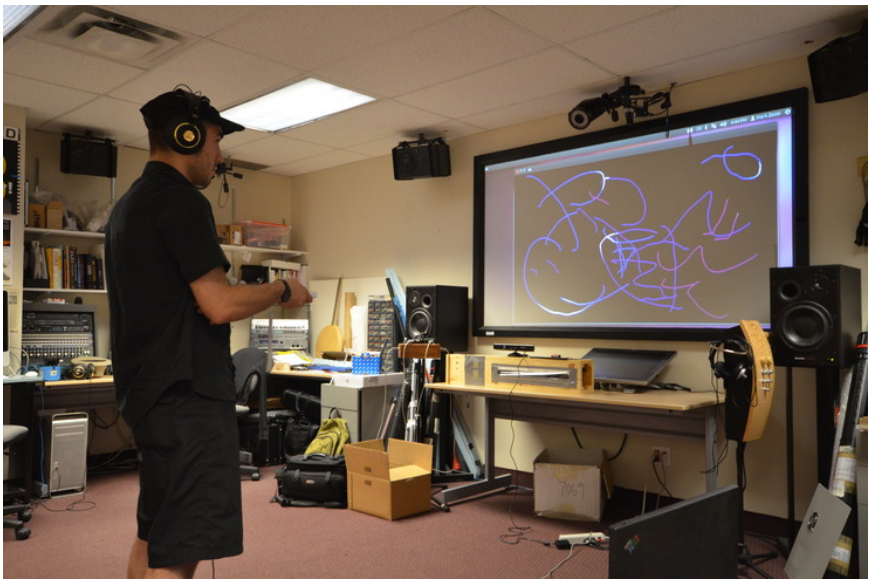


Figure 6.4 A participant in the wiimote controller test. The participant is wearing headphones and holding the wiimote in his right hand, controlling the *Different Strokes* interface projected on the large screen.

released. This matches the case with the left mouse button, and the graphic tablet when the pen is in contact with the tablet.

The wiimote's movements are sensed using its infrared camera and a standard Nintendo "IR sensor bar". The bar contains two infrared (IR) LEDs, which the wiimote's infrared camera sees as two blobs whose positions are reported in the 2D space of the camera frame. The sensor bar is assumed to be stationary, so any changes in the apparent position of the IR LEDs are attributed to movements of the wiimote itself. The bar is intended to be placed nearly coincident to the display (in our case, the projection screen) in order to allow the wiimote to be pointed at the display where the participant's attention is focused.

Movement and rotation gestures are thus sensed through the IR camera in the handheld controller. Both lateral movements and changes in rotation will lead to a displacement in the blob positions. Ultimately, this sensing is designed to allow the user to point to a location on the screen, and it provides a sufficient approximation of orientation and position to achieve this. There is no digital compass in the wiimote, so it is not sensing true orientation.

Only one of the blobs tracked by the IR camera was used in the mapping. The *Dif-*

ferent Strokes cursor was judged to be in the centre of the canvas at position $(0, 0)$ when the blob was at the centre of the camera's image. This meant that the cursor was centred when the wiimote was pointed at the IR bar and not at the centre of the projected image. This discrepancy was generally transparent to the user since relative motions of the wiimote created consistent, corresponding motions of the cursor, like a computer mouse.

Wiimote button	***Different Strokes* function**
B (trigger)	Initiate / end drawing (corresponding to the left mouse button)
A	Select the silent stroke state
D-pad left	Select audible stroke 1
D-pad up	Select audible stroke 2
D-pad right	Select audible stroke 3
D-pad down	Select audible stroke 4
Minus button (-)	Select removal state

Table 6.1 The correspondence of wiimote buttons to standard *Different Strokes* keyboard and mouse controls.

The mapping of wiimote buttons followed the standard mouse and keyboard commands closely, and can be seen in Table 6.1. The system is controlled entirely by the motion of the input device and by its built-in buttons, so no additional computer keyboard was needed (as is used in the typical DS setup). The stroke's colour (i.e., its associated sound) was chosen using the directional buttons on the wiimote.

The wiimote signals are transmitted to *Different Strokes* using a Pure Data patch that makes use of the `mapper` external provided by *libmapper*, as well as the `wiimote` external, which exposes the sensor and button data from the wiimote as Pure Data messages. The blob and button data from the wiimote is passed to the mapper object, and flows out through *libmapper* connections (Figure 6.5). The following output signals are defined in the patch (and thus for the wiimote itself):

```
/posx
/posy
/posz
/strokestate
/removalstate
```

These correspond directly to each DS input signal, so in the *libmapper* environment, each output is mapped directly to its corresponding input signal in DS (Figure 6.6).

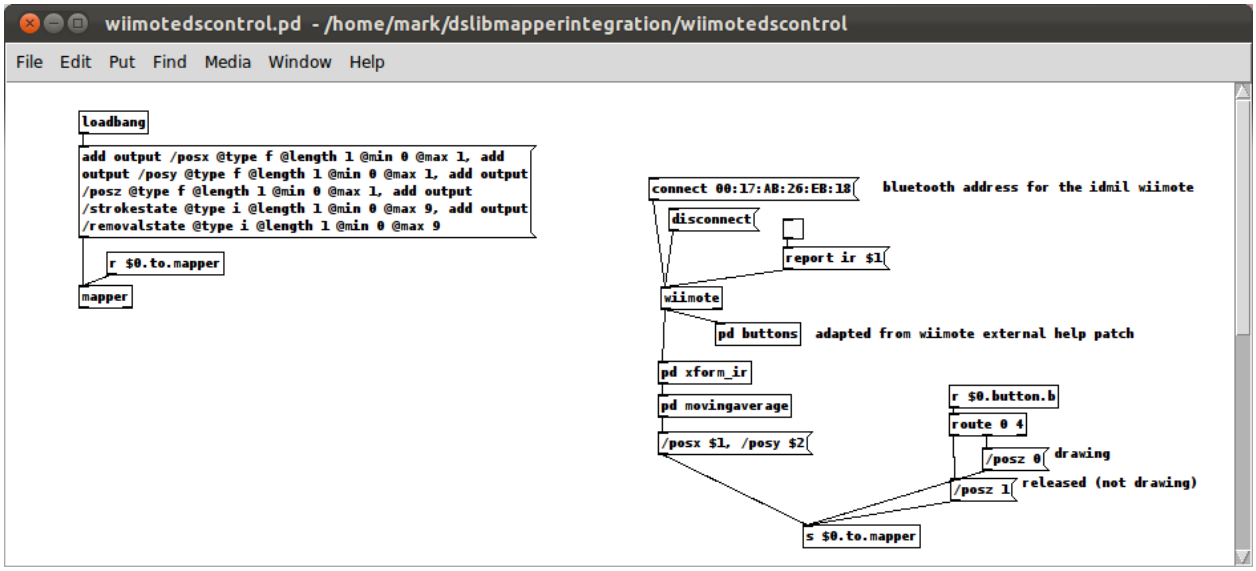


Figure 6.5 The main part of the Pure Data patch, showing the mapper and wiimote objects. The wiimote IR data and button presses are propagated to DS through *libmapper*. The declaration of the *libmapper* output signals can be seen in the input to the mapper object, on the left.

Since the position signal derived from the IR blob tracking was slightly noisy, it tended to cause small, unintentional micro-loops when drawing a stroke. This would trap particles and would cause the particles to behave differently than the participant might expect. To address this, we implemented a moving average filter to reduce the high frequency noise. The filter is implemented in the Pure Data patch, after the `wiimote` external has received the IR blob position data. The x and y coordinates of the position are filtered independently, and a four-sample window is used. This filter served to eliminate the unintentional micro-loops without introducing a noticeable lag to the pointer movement.

6.2.1.1 Observations

The movements of the cursor on the screen felt surprisingly responsive: the on-screen cursor followed the wiimote's orientation fairly tightly, and the latency was not so large as to be distracting. This was somewhat surprising given that the sensor signals are

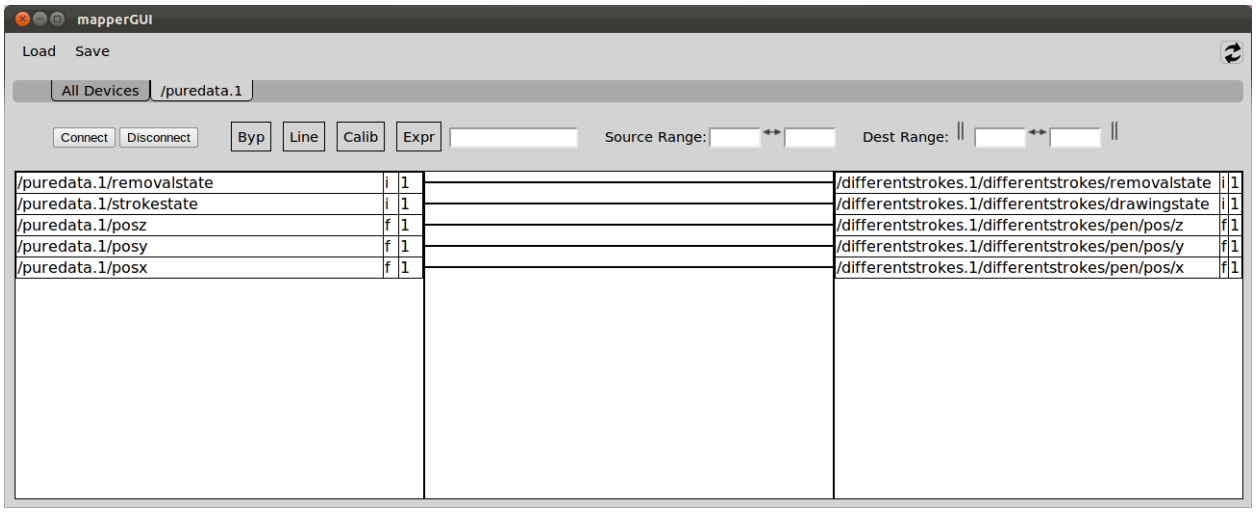


Figure 6.6 A screenshot of webmapper, showing the *libmapper* connections. Each output from the Pure Data patch is connected directly to an input in DS.

propagated through several steps on its way to DS (the wiimote hardware, bluetooth transmission, Pure Data, *libmapper*, and *Different Strokes* itself). This may be partly due to the fact that in the experimental setup, all of this software was running on the same computer, taking advantage of the computer's fast loopback network connections.

The battery power to the IR sensor bar was an unexpected source of noise. As the battery wore down, the IR LEDs dimmed, leading to increased noise in the reported blob positions. This was noticeable to at least one of the participants. The battery also ran down surprisingly quickly.

6.2.2 Participant Comments

The participants freely vocalized their feelings during the test. Here we summarize the important themes that arose.

6.2.2.1 Overall Wiimote Intuitiveness / Learning

Overall, the participants found the wiimote easy to understand and found it generally intuitive:

> "Overall using the wiimote was nice. It's not too noisy [in terms of position], and there's a little bit of lag, but not too much." [P1]

> "The interface hasn't been getting in the way at all...Yeah, the wiimote's great." [P2]
>
> "It's easy to use...It's intuitive...It's a great way to explore if you just want to try it out." [P3]
>
> "It's really clear and easy to learn." [P6]
>
> "It's easy enough to draw" [P7]

Some participants had a few reservations:

> "I like the wiimote less than a drawing tablet." [P1]
>
> "It took one or two minutes to get used to it." [P7]

6.2.2.2 Controller Accuracy for Drawing

Generally, the participants found the accuracy of spatial positioning to be lacking in the tested setup:

> "The wiimote isn't accurate enough—sometimes I draw elsewhere unintentionally...It is not as accurate as the stylus." [P1]
>
> "There's a bit of lag and jumping, but it doesn't interfere too much." [P2]
>
> "For precise drawings, it's not the best." [P3]
>
> "It looks messy when writing...For precision, the stylus would be better" [P5]

Not all of the participants felt that way, however:

> "I think it's precise. There's more control than I expected." [P6]

The wiimote magnified the natural jitter in the participants' hand motions, especially for slow strokes. This would sometimes lead to small self-intersections in the stroke:

> "Sometimes the stroke would self-intersect as I drew, adding little microloops unintentionally." [P1]
>
> "It's really hard to do long, slow strokes due to both the wiimote and the mapping...I often found myself looping back within a stroke without meaning to...Also, if you stop in the middle of the gesture, your hand jerks back a bit and that tends to create small loops unintentionally." [P4]

"Smaller movements are really hard." [P4]

"I get better control if I have some momentum." [P5]

One participant found the lack of physical feedback to be problematic:

"Since it's wireless, it's harder to control in space and time since you're only relying on proprioception. It'd be easier to use a touch screen to navigate quickly since you have more physical feedback." [P4]

6.2.2.3 IR bar offset

The IR sensor bar was placed at the bottom of the screen, and the mapping necessitated that the wiimote was pointed at the IR bar to centre the cursor in the middle of the screen. While most participants did not comment on it, some noticed the discrepancy:

"It takes a bit of adjustment at the very beginning to understand the angle adjustment you need to make, but it's easy to get used to it." [P2]

"For comfort, I'd like the sensor bar higher so I could position my wrist higher." [P5]

One participant expressed the opposite opinion, however:

"I feel there is a big relation between the place I'm pointing and the place I'm drawing" [P6]

6.2.2.4 Visual Feedback

Some participants gave suggestions for improving the visual feedback in the system to better represent its internal state:

"Here's an idea: have the cursor give feedback as to what mode you are in (removal, sounding stroke...). I find myself forgetting." [P1]

"I would forget which sample I had selected and which one corresponded to which direction on the D-pad. I would also sometimes delete strokes without meaning to." [P4]

"I can't see the cursor against the maze of particles." [P5]

"There should be more visual feedback to reflect the state of the system." [P7]

Participants requested having an indication in the interface as to what direction a stroke was drawn in:

> "It's not intuitive how it doesn't indicate the direction that a stroke was drawn in." [P2]
>
> "I don't know where the beginning and the end of the stroke is. If I drew it a while ago, I forget." [P7]

6.2.2.5 Spatial Issues

Drawn stroke topologies take up space, and participants would sometimes unintentionally divide up the canvas in undesirable ways:

> "Oh, there's a topological problem: I made a loop in one part of the canvas that I want to connect, but that other structure is blocking the way." [P1]
>
> "You end up drawing obstacles and navigating around obstacles is really hard to do quickly and in time." [P4]
>
> "I'd like to have a button to play the strokes that I've already drawn without drawing a new one...just playing, not creating structures." [P6]

Participant four commented on the size of the drawing canvas relative to the size of the gestures that the wiimote and its spatial mapping engendered:

> "You don't have much control over the speed of the stroke; you have too much jitter in your hand to do a slow stroke, but then a quicker stroke crosses the entire space...It feels like a small space since the size of the gestures you're required to make are large compared to the space." [P4]

6.2.2.6 Mapping

Different Strokes was mapped to draw while the wiimote's trigger button was depressed. Participants generally found this easy to understand.

> "I like the trigger button [for drawing]." [P6]

One participant found that the using trigger button compromised the possible gestures:

> "Also in terms of writing and precision, pressing the trigger button is a bit tricky...I'd do better holding it upside down [with the index finger on the trigger]...I could get tighter, smaller movements that way. I'd write and draw better." [P5]

The lack of positional accuracy of the wiimote interacted with the default DS mapping, which binds drawing speed to playback speed. This was not always desirable:

> "Sometimes I have to slow down to close a circle, but not because I want the pitch to be lower." [P6]

6.2.2.7 Engagement

The subjects generally found the wiimote control and the software itself to be generally engaging:

> "It has a lot of potential; there's stuff to do and it doesn't wear out its welcome too fast." [P2]
>
> "It's fun, I like using it." [P5]
>
> "I like it. I was getting into it." [P7]

6.2.2.8 Other Comments

> "Having the projection screen is great. It's easier to focus on specific subsections." [P1]
>
> "The wiimote's nice since it opens up the potential for having multiple users drawing in the same canvas." [P1]
>
> "I'd have to practice to find strategies to do what I want to do." [P4]
>
> "I'd like to be able to control the volume of the sound to be able to do more subtle things. For example, I can't really do background and foreground [sounds] the way it's set up now." [P1]

Some participants wanted an option to be able to delete many strokes at once:

> "It's hard to erase the small strokes. I wanted to have an 'erase all' button sometimes." [P4]
>
> "I would like to be able to select a region and delete it all at once. I could finish the whole session in one interaction." [P6]

6.2.3 Summary

The Nintendo wiimote controller was found by participants to be generally intuitive, quick to understand and use. The mapping from the controller buttons to DS state changes was easily understood by the participants as well; in particular, the use of the wiimote's trigger button to draw appeared to be intuitive to the participants. The large projection screen format also worked well. As one of the participants commented, this setup would be well suited to use as an installation in a gallery or children's museum due to its intuitiveness and format.

The spatial control afforded by the wiimote under this mapping left something to be desired: some participants found it to be insufficiently accurate for certain gestures, especially small or slow strokes. This is due to the noisiness in the position signal; perhaps this behaviour could be improved by lengthening the window in the moving average filter, at the cost of adding latency.

Some participants found the relationship between the scale of the physical gestures they made and their on-screen representations to be inappropriate, in that the traces took up a relatively large portion of the space of the drawing canvas. The large size of the traces consequently fills up the space quickly, dividing it into regions that may not be desirable to a given performer. This is due to the relatively narrow field of view of the wiimote's IR camera, where relatively small rotations of the wiimote can cause the LEDs to cross the entire space. When using a stylus, drawing gestures rely on displacements rather than rotations, and the size of the graphic tablet can be made large enough to accommodate the natural size of most drawing gestures. The result is that the traces from typical stylus gestures take up less on-screen space. Though this was not an issue for all participants, it could be addressed by using a different controller with an alternate position sensing mechanism that would allow an increased range of motion.

The vertical position offset introduced by the IR LEDs seemed to be generally transparent to the participants, though two participants noticed and mentioned it.

The main purpose of this test was to demonstrate the simplicity of attaching alternate input devices to the *Different Strokes* system through *libmapper*. Once the wiimote signals were exposed by the Pure Data patch, it could be attached to DS by simply connecting the relevant signals in a graphical *libmapper* interface. Once the *libmapper* adaptation itself was completed, no internal DS code needed to be modified to connect this new device.

6.2.4 Future Work

The wiimote's use in this *Different Strokes* setup is generally intuitive, but advanced users would benefit from an increased range of motion and positional accuracy. Using a different input device with better properties in this respect would be more suited to serious use.

More visual feedback should be introduced into *Different Strokes* to make its state explicit and to convey necessary information to the performer. The currently selected drawing state could be displayed, either in the cursor shape, or by using an indicator at the edge of the screen (e.g., by adapting the palette presented in Section 4.3.4). Also, the directionality of each stroke should be shown, perhaps by a subtle, animated "flow" effect or directional markers. Finally, a palette of sounds could be used along the side of the screen as a more intuitive, visually explicit method of choosing sounds.

Other input gestures could be added as well to increase the performer's interaction options. For example, the user could lasso regions of the screen to delete strokes at once without having to remove each one manually.

6.3 Conclusion

We have presented extensions to *Different Strokes* that allow its simulation and graphic representation to be used in more general contexts, with arbitrary hardware inputs and synthesis outputs. This was accomplished by integrating *libmapper* support into DS, allowing the free interconnection of *Different Strokes* and *libmapper*-compliant input controllers and sound synthesizers.

This extension allowed us to easily test non-stylus inputs to DS. In particular, we tested the use of a wiimote controller. Participant feedback indicated that the controller

was generally easy to understand and use, but lacked sufficient accuracy for detailed drawn figures.

The user feedback from the exercise presented above differs from that reported in Section 5.2.6, but suggests general interface improvements that go beyond the particular hardware context used. For example, comments about potential additional visual feedback and the spatial scaling of input gestures could be applicable to the software design in general. This is one example of how using a graphical performance interface in multiple hardware contexts can help bring overall design improvements to light.

Chapter 7

Conclusion and Future work

7.1 Contributions and Discussion

The main contribution of this work is to propose a novel approach to the analysis of software systems for musical expression that involves using the software in varied hardware and usage contexts. Graphical software instruments can be run using a variety of input devices and displays that offer ostensibly equivalent affordances, but the physical differences in these devices alter the user's experience. Each viewpoint provides different insights into the interaction design and suggests improvements that can benefit the design in general. This analysis approach was studied using the specific case of *Different Strokes*, a graphical music performance application based on drawing gestures. The original version of DS was adapted to work in a variety of different hardware contexts and with different input devices.

Other contributions include the software extensions to DS itself. Various features were implemented in the context of the *d_verse* performance piece, which included multi-channel audio, MIDI support, and robustness improvements. The project served as a valuable real-world test case, with specific requirements that included using DS on a very large projection screen and interacting with individuals (dancers) in the projection space.

DS was adapted for use on a large multi-touch table. This allows it to be used with multiple users in a collaborative way, and engenders alternate interaction strategies to those seen in the standard setup because it allows multiple points of contact, and be-

cause the interface is used with direct finger contact instead of being mediated by a tool (e.g., stylus).

The DIMPLE framework was integrated into *Different Strokes* to allow it to support haptically-enabled input devices and force feedback. This served as a rich test bed for exploring how the differing physical characteristics of the input device can influence the performer's experience of using the system.

DS was also integrated with the *libmapper* framework for easy experimentation with alternate input devices and sound output. This, in particular, is an important development because it simplifies the task of porting the application to new devices and hardware contexts, thereby facilitating the proposed testing approach.

Exploratory user testing was performed in two contexts: in the haptically-enabled version and in the *libmapper*-enabled version. For the haptics testing no clear preference for any of the force feedback effects was shown. The gesture analysis shows that the simulated forces may have affected the velocity profiles in the "constant speed" task.

The tangential velocity profiles in the recorded gesture data demonstrate that it is generally difficult for a user to maintain a constant drawing speed. This brings into question the usefulness of using drawing speed as a mapped parameter if it is to be kept constant. While the DS design was intended to exploit the natural variability in the drawing speed to lead to an interesting sound aesthetic, this highlights the fact that mappings need to be chosen with an understanding of human control capabilities, both physical and cognitive.

In the *libmapper* user testing, participants controlled DS with a wiimote and used a large screen projection. The user feedback showed that while this configuration was easy to get started with, the wiimote and mapping lacked the desired spatial accuracy for some participants. The test showed how *libmapper* can make the process of reconfiguring the input to DS quick and easy, which is conducive to doing varied hardware testing. Notably, the user feedback differed from the feedback recorded in the test with haptics, but in both cases, improvements were suggested that could benefit the DS design in general. This shows how testing the software in differing hardware contexts can lead to new generally-applicable insights into the software design.

One detail we should consider in adopting this approach is its overall cost and feasibility. Software development is expensive in terms of time, money, expertise and effort. Is it realistic to design across multiple hardware platforms in the way described here?

It is our belief that the approach presented in this dissertation is feasible if applied at an early stage of the development process. Certainly it can be costly to adapt existing, mature software to new hardware platforms. Production-quality software typically focuses on robustness, completeness and efficiency, often resulting in corresponding code complexity. However, if lightweight prototyping tools and software libraries are used during initial development (e.g., *Processing*, *SuperCollider*, *libmapper*), it is possible to experiment with new interface design ideas and new hardware platforms cheaply. Sketching out the interface quickly and iteratively in the early design stages, taking into account multiple hardware contexts, can benefit graphical performance software designs while still managing their development costs.

7.2 Interface Design Principles

The interface design should respect the user's physical and cognitive abilities. On-screen, visible objects should tightly coordinated with their corresponding sounds, and the user should not be expected to track more than a few moving items. The tasks that the user is expected to perform should be physically achievable. In the case of *Different Strokes*, where the drawing velocity is an important parameter in the mapping, the gesture data show that it is difficult to maintain a constant velocity. This makes it difficult for performers to play a sound sample at a constant rate if they desire to do so.

The input device needs to be appropriately matched to the task. Generally, for 2D planar interfaces, this means integral control in two dimensions.

The scale of the physical gestures that are natural for the task should match the spatial range of the input device and mapping. For example, in the wiimote task, the combination of cursor position mapping, large-scale wiimote gestures and screen space was too spatially constrictive for some users.

In terms of the implementation, modular code separating the internal system from the input device should be considered from the outset, at least for prototyping, to be able to reconfigure the system for testing. From the perspective of the user, there is a convenience tradeoff between providing a self-contained software package and a modular one that depends on external components such as *libmapper*. For testing with alternative input devices, however, it is helpful to be able to easily reconfigure the system.

Robustness and code stability are key considerations for graphical software instru-

ments that are intended to be played live. For the *d_verse* project, robustness improvements were essential to be able to trust the system enough to use it expressively.

Since the performer needs to focus on the screen when using a graphical software instrument, the interface designer should not require that they shift their attention elsewhere. This problem arose in the *d_verse* project, where the performer was required to attend to the local input hardware and simultaneously to a distant screen projection.

Finally, the designer should consider how tightly the input gesture is related to the output sound. If the sound needs to be controlled gesturally (e.g. allowing quick crescendos or silences), there should be a signal-level mapping between the input and the output (Malloch et al., 2006). In the case of DS, it can be difficult to gesturally control the output sound (say, in response to a dancer's movements) since the sound playback itself is strongly mediated by the autonomous action of the simulation.

7.3 Future Work

The system analysis approach put forward in this thesis could be further explored by applying the methodology to other graphical music performance software applications. *Different Strokes* itself could also be tested in other hardware contexts. For example, free-hand interaction with a depth camera, such as the Microsoft Kinect, could be explored. Additionally, the DS application could be ported to smartphone devices. This would also allow more third-party users to test the software and provide feedback. This work has already begun with Stephen Sinclair's port of DS to the Android operating system. In this vein, its use by unaffiliated third-party musicians in live settings could also provide important feedback to refine the design.

User testing on the multi-touch table could be done in order to explore, extend and refine the interaction design in the multi-touch context. New interaction strategies could be tested (for example, pinching, multi-finger editing gestures, and non-keyboard interaction).

Functional Data Analysis (FDA) techniques could applied to the gesture data analysis. FDA (Ramsay and Silverman, 2002; Ramsay et al., 2009) is a statistical technique that considers each datum to be a continuous function over some domain. Generally, the curves are taken to be time series for the value of a given variable. Collections of continuous curves can then be analyzed using techniques like Principal Components

Analysis, and the resulting statistics are expressed as functions over time.

Though it is outside the scope of the present work, a potentially interesting investigation to undertake could be a comparison of DS performance practice to that of turntablism. A scratch deejay's use pre-recorded material, looping and gestural pitch control overlaps with certain aspects of performance using DS, and it could be instructive to compare notions of expressivity and virtuosity between the two domains.

The ultimate goal of our overarching research project is to adapt principles from HCI to develop a general framework for designing and evaluating on-screen interfaces for live, expressive use. We would like to answer the question, "how should we design interfaces to be amenable to live, on-stage use?" This thesis work is one step toward answering this question. We believe that there is potential for new innovations in HCI, and that the popularity and ubiquity of graphical interfaces make this a potentially fruitful area of research. A theory of live performance-oriented interface design and evaluation might also yield benefits outside of the arts, in interaction design for real-time systems in general.

References

3Dconnexion. SpaceNavigator product page, 2012. http://www.3dconnexion.com/products/spacenavigator.html. Accessed 11 May 2012.

Ableton AG. Ableton homepage, 2012. http://www.ableton.com/. Accessed 5 August 2012.

Apple, Inc. iPod touch product page, 2012. http://www.apple.com/ipodtouch/. Accessed 5 August 2012.

D. Arfib and J. Dudon. A digital emulator of the photosonic instrument. In *Proceedings of the Conference on New Interfaces for Musical Expression*, pages 128–131, 2002.

Ascension Technology Corporation. Flock of birds, 2007. http://www.ascension-tech.com/products/flockofbirds.php.

L. Bartram. Enhancing visualizations with motion. In *Proceedings of IEEE Information Visualization 1998*, 1998.

M. Beaudouin-Lafon. Instrumental interaction: an interaction model for designing post-wimp user interfaces. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 446–453, The Hague, The Netherlands, 2000. ACM.

R. Bencina. The metasurface: applying natural neighbour interpolation to two-to-many mapping. In *Proceedings of the Conference on New Interfaces for Musical Expression*, NIME '05, pages 101–104, 2005.

R. Bencina. Audiomulch interactive music studio, 2012. http://www.audiomulch.com/. Accessed 5 August 2012.

R. Berry, M. Makino, N. Hikawa, and M. Suzuki. The augmented composer project: the music table. In *Proceedings of the Second IEEE and ACM International Symposium on Mixed and Augmented Reality*, pages 338–339, 2003.

T. Blaine and T. Perkis. The Jam-O-Drum interactive music system: a study in interaction design. In *Proceedings of the 3rd conference on Designing interactive systems: processes, practices, methods, and techniques*, pages 165–173, New York City, New York, United States, 2000. ACM.

R. Boulanger and M. Mathews. The 1997 Mathews Radio-Baton & improvisation modes. In *Proceedings of the International Computer Music Conference*, pages 395–398, 1997.

A. S. Bregman. *Auditory Scene Analysis: The Perceptual Organization of Sound*. MIT Press, Cambridge, Mass, 1990.

A. S. Bregman. Auditory scene analysis: hearing in complex environments. In S. McAdams and E. Bigand, editors, *Thinking in Sound: The Cognitive Psychology of Human Audition*, pages 10–36. Clarendon Press, Oxford, 1993.

E. Brown, W. Buxton, and K. Murtagh. Windows on tablets as a means of achieving virtual input devices. In *Human-Computer Interaction — INTERACT '90*, pages 675–681, 1990.

W. Buxton, R. Sniderman, W. Reeves, S. Patel, and R. Baecker. The evolution of the SSSP score editing tools. *Computer Music Journal*, 3(4):14–25, 1979.

W. A. S. Buxton. Chunking and phrasing and the design of human-computer dialogues. In R. M. Baecker, J. Grudin, W. A. S. Buxton, and S. Greenberg, editors, *Readings in Human-computer interaction: toward the year 2000*, pages 494–499. Morgan Kaufmann Publishers Inc., 1995.

S. K. Card, J. D. Mackinlay, and B. Shneiderman, editors. *Readings in information visualization: using vision to think*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1999.

K. Cascone. Laptop music — counterfeiting aura in the age of infinite reproduction. *Parachute*, 107:52–59, 2002.

K. Cascone. Grain, sequence, system: Three levels of reception in the performance of laptop music. *Contemporary Music Review*, 22(4):101–104, 2003.

J. Chadabe. Interactive composing: An overview. *Computer Music Journal*, 8(1):22–27, 1984.

Chocopoolp. Chocopoolp software site, 2006. `http://www.chocopoolp.com/`. Accessed 5 August 2012.

I. Choi, R. Bargar, and C. Goudeseune. A manifold interface for a high dimensional control space. *Proceedings of the International Computer Music Conference*, pages 385–392, 1995.

N. Collins. Generative music and laptop performance. *Contemporary Music Review*, 22: 67–79, 2003.

Community Core Vision. Home page, 2012. `http://ccv.nuigroup.com/`. Accessed 5 August 2012.

W. J. Conover. *Practical Nonparametric Statistics*. Wiley series in probability and mathematical statistics. Wiley, New York, 2nd edition, 1980.

F. Conti, F. Barbagli, D. Morris, and C. Sewell. CHAI: an open-source library for the rapid development of haptic scenes. Demo paper presented at IEEE World Haptics, 2005.

J.-M. Couturier. A model for graphical interaction applied to gestural control of sound. In *Proceedings of Sound & Music Computing 2006*, Marseille, France, 2006.

J.-M. Couturier and D. Arfib. Pointing fingers: using multiple direct interactions with visual objects to perform music. In *Proceedings of the Conference on New Interfaces for Musical Expression*, pages 184–187, Montreal, Quebec, Canada, 2003.

A. Crevoisier and G. Kellum. Transforming ordinary surfaces into multi-touch controllers. In *Proceedings of the Conference on New Interfaces for Musical Expression*, pages 113–116, 2008.

A. Crevoisier and P. Polotti. Tangible acoustic interfaces and their applications for the design of new musical instruments. In *Proceedings of the Conference on New Interfaces for Musical Expression*, pages 97–100, 2005.

J. Croft. Theses on liveness. *Organised Sound*, 12(01):59–66, 2007.

N. D'Alessandro and T. Dutoit. Handsketch bi-manual controller: Investigation on expressive control issues of an augmented tablet. In *Proceedings of the Conference on New Interfaces for Musical Expression*, pages 78–81, 2007.

P. L. Davidson and J. Y. Han. Synthesis and control on large scale multi-touch sensing displays. In *Proceedings of the Conference on New Interfaces for Musical Expression*, pages 216–219, 2006.

J. d'Escriván. To sing the body electric: Instruments and effort in the performance of electronic music. *Contemporary Music Review*, 25(1):183 – 191, 2006.

J. Driver and C. Spence. Multisensory perception: Beyond modularity and convergence. *Current Biology*, 10(20):R731–R735, 2000.

EMF Institute. the Hub, 2006. `http://emfinstitute.emf.org/exhibits/hub.html`.

M. O. Ernst and H. H. Bülthoff. Merging the senses into a robust percept. *Trends in Cognitive Sciences*, 8(4):162–169, 2004.

European Bridges Ensemble. Home page, 2012. `http://www.e-b-e.eu/`. Accessed 5 August 2012.

R. Fiebrink, G. Wang, and P. R. Cook. Don't forget the laptop: using native input capabilities for expressive musical control. In *Proceedings of the Conference on New Interfaces for Musical Expression*, pages 164–167, New York, New York, 2007. ACM.

G. W. Fitzmaurice, H. Ishii, and W. A. S. Buxton. Bricks: laying the foundations for graspable user interfaces. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 442–449. ACM Press/Addison-Wesley Publishing Co., 1995.

E. Franco. *Miró: a flexible expressive audiovisual system for real-time performance & composition*. M.S. thesis, University of Limerick, 2004.

E. Franco, N. J. L. Griffith, and M. Fernström. Issues for designing a flexible expressive audiovisual system for real-time performance & composition. In *Proceedings of the Conference on New Interfaces for Musical Expression*, pages 165–168, Hamamatsu, Shizuoka, Japan, 2004.

A. Freed. Application of new fiber and malleable materials for agile development of augmented instruments and controllers. In *Proceedings of the Conference on New Interfaces for Musical Expression*, pages 107–112, 2008.

G. Geiger. Using the touch screen as a controller for portable computer music instruments. In *Proceedings of the Conference on New Interfaces for Musical Expression*, pages 61–64, 2006.

S. Handel. *Perceptual Coherence: Hearing and Seeing*. Oxford University Press, 2006.

A. Harker and A. Atmadjaja. Worldscape laptop orchestra. In *Proceedings of the International Computer Music Conference*, 2008.

S. Harris. LibLo homepage, 2012. `http://liblo.sourceforge.net/`. Accessed 5 August 2012.

W. S. Harwin and N. Melder. Improved haptic rendering for multi-finger manipulation using friction cone based god-objects. In *Proceedings of Eurohaptics conference*, pages 82–85, 2002.

V. Hayward, O. R. Astley, M. Cruz-Hernandez, D. Grant, and G. Robles-De-La-Torre. Haptic interfaces and devices. *Sensor Review*, 24(1):16–29, 2004.

hexler.net. TouchOSC product page, 2012. `http://hexler.net/software/touchosc`. Accessed 14 June 2012.

A. Hunt, M. M. Wanderley, and M. Paradis. The importance of parameter mapping in electronic instrument design. In *Proceedings of the Conference on New Interfaces for Musical Expression*, pages 1–6, Dublin, Ireland, 2002.

R. Huott. An interface for precise musical control. In *Proceedings of the Conference on New Interfaces for Musical Expression*, 2002.

M. Irving. *Interpretive Practice in Laptop Music Performance*. M.A. thesis, Simon Fraser University, 2006.

T. Iwai. Electroplankton, 2005. `http://electroplankton.nintendods.com/`. Accessed 5 August 2012.

R. J. K. Jacob, L. E. Sibert, D. C. McFarlane, and J. M. P. Mullen. Integrality and separability of input devices. *ACM Transactions on Computer-Human Interaction*, 1(1):3–26, 1994.

T. Jaeger. The (anti-)laptop aesthetic. *Contemporary Music Review*, 22(4):53, 2003.

JazzMutant. Lemur product page, 2008. `http://www.jazzmutant.com/lemur_overview.php`. Accessed 5 August 2012.

S. Jordà. FMOL: toward user-friendly, sophisticated new musical instruments. *Computer Music Journal*, 26(3):23–39, 2002.

S. Jordà, G. Geiger, M. Alonso, and M. Kaltenbrunner. The reacTable: exploring the synergy between live music performance and tabletop tangible interfaces. In *Proceedings of the 1st international conference on Tangible and embedded interaction*, pages 139–146, 2007.

S. Jordà Puig. *Digital Lutherie: Crafting musical computers for new musics' performance and improvisation*. Ph.d., Universitat Pompeu Fabra, 2005.

M. Kaltenbrunner, G. Geiger, and S. Jordà. Dynamic patches for live musical performance. In *Proceedings of the Conference on New Interfaces for Musical Expression*, 2004.

M. Kaltenbrunner, T. Bovermann, R. Bencina, and E. Costanza. TUIO: a protocol for table-top tangible user interfaces. In *Proceedings of the 2nd Interactive Sonification Workshop*, 2005.

L. Kessous. Bi-manual mapping experimentation, with angular fundamental frequency control and sound color navigation. In *Proceedings of the Conference on New Interfaces for Musical Expression*, pages 1–2, 2002.

L. Kessous and D. Arfib. Bimanuality in alternate musical instruments. In *Proceedings of the Conference on New Interfaces for Musical Expression*, pages 140–145, 2003.

Korg USA. KP3 KAOSS pad product page, 2007. `http://korg.com/KP3`. Accessed 5 August 2012.

G. Kramer. An introduction to auditory display. In G. Kramer, editor, *Auditory Display: Sonification, Audification, and Auditory Interfaces*, pages 1–77. Addison-Wesley, 1994.

J. Lamping and R. Rao. Visualizing large trees using the hyperbolic browser. In *CHI '96: Human factors in computing systems (Conference companion)*, pages 388–389, New York, NY, USA, 1996. Association for Computing Machinery.

p. k. langshaw. *d_verse: transitional algorhythms of gesture*. Concordia University, 2011.

S. K. Lee, W. Buxton, and K. C. Smith. A multi-touch three dimensional touch-sensitive tablet. *ACM SIGCHI Bulletin*, 16(4):21–25, 1985.

G. Levin. *Painterly Interfaces for Audiovisual Performance*. M.A. thesis, Massachusetts Institute of Technology, 2000.

T. Magnusson. Affordances and constraints in screen-based musical instruments. In *Proceedings of the 4th Nordic conference on Human-computer interaction: changing roles*, pages 441–444, Oslo, Norway, 2006a. ACM.

T. Magnusson. Screen-based musical interfaces as semiotic machines. In *Proceedings of the Conference on New Interfaces for Musical Expression*, pages 162–167, Paris, France, 2006b. IRCAM — Centre Pompidou.

J. Malloch and M. M. Wanderley. The t-stick: From musical interface to musical instrument. In *Proceedings of the Conference on New Interfaces for Musical Expression*, pages 66–69, 2007.

J. Malloch, D. Birnbaum, E. Sinyor, and M. M. Wanderley. Towards a new conceptual framework for digital musical instruments. In *Proceedings of the 9th International Conference on Digital Audio Effects*, pages 49–52, 2006.

J. Malloch, S. Sinclair, and M. M. Wanderley. A Network-Based framework for collaborative development and performance of digital musical instruments. In R. Kronland-Martinet, S. Ystad, and K. Jensen, editors, *Computer Music Modeling and Retrieval. Sense of Sounds*, volume 4969 of *Lecture Notes in Computer Science*, pages 401–425. Springer Berlin / Heidelberg, 2008.

G. Marino, M.-H. Serra, and J.-M. Raczinski. The UPIC system: Origins and innovations. *Perspectives of New Music*, 31(1):258–269, 1993.

M. V. Mathews and F. R. Moore. GROOVE—a program to compose, store, and edit functions of time. *Communications of the ACM*, 13(12):715–721, 1970.

J. McCartney. Continued evolution of the SuperCollider real time synthesis environment. In *Proceedings of the International Computer Music Conference*, pages 133–136, Ann Arbor, MI, 1998.

Milwaukee Laptop Orchestra. Home page, 2008. `http://www.uwm.edu/~gssurges/milo/`. Accessed 5 August 2012.

E. R. Miranda and M. M. Wanderley. *New digital musical instruments : control and interaction beyond the keyboard*. The computer music and digital audio series, v. 21. A-R Editions, Middleton, Wis., 2006.

A. Momeni and D. Wessel. Characterizing and controlling musical material intuitively with geometric models. In *Proceedings of the Conference on New Interfaces for Musical Expression*, pages 54–62, 2003.

MooCowMusic. MooCowMusic:Band — the iPhone music studio, 2008. `http://moocowmusic.com/Band/`. Accessed 5 August 2012.

Native Instruments GmbH. Reaktor product page, 2012. `http://www.native-instruments.com/en/products/producer/reaktor-55/`. Accessed 5 August 2012.

S. M. O'Modhrain. *Playing by Feel: Incorporating Haptic Feedback into Computer-Based musical Instruments*. PhD thesis, Stanford University, November 2000.

openFrameworks. openFrameworks homepage, 2012. http://www.openframeworks.cc. Accessed 9 April 2012.

J. Ott and M. Packard. Thicket product page, 2010. http://itunes.apple.com/us/app/thicket/id364824621. Accessed 14 June 2012.

J. Patten, B. Recht, and H. Ishii. Audiopad: A tag-based interface for musical performance. In *Proceedings of the Conference on New Interfaces for Musical Expression*, Dublin, Ireland, 2002.

Plogue Art et Technologie. Bidule product page, 2012. http://www.plogue.com/bidule/. Accessed 5 August 2012.

PowerBooks_UnPlugged. Home page, 2012. http://pbup.goto10.org/. Accessed 5 August 2012.

Propellerhead Software AB. Reason product page, 2012. http://www.propellerheads.se/products/reason/. Accessed 5 August 2012.

M. Puckette. The patcher. In *Proceedings of the International Computer Music Conference*, pages 420–425, 1988.

M. Puckette. Pure data. In *Proceedings of the International Computer Music Conference*, pages 269–272, Hong Kong, 1996.

M. Puckette. Max at seventeen. *Computer Music Journal*, 26(4):31–43, 2002.

V. Pulkki. Virtual sound source positioning using vector base amplitude panning. *Journal of the Audio Engineering Society*, 45(6):456–466, 1997.

J. O. Ramsay and B. W. Silverman. *Applied Functional Data Analysis: Methods and Case Studies*. Springer series in statistics. Springer, New York, 2002.

J. O. Ramsay, G. Hooker, and S. Graves. *Functional Data Analysis with R and MATLAB*. Springer, 2009.

G. G. Robertson, S. K. Card, and J. D. Mackinlay. Information visualization using 3D interactive animation. *Communications of the ACM*, 36(4):57–71, 1993.

X. Rodet, J.-P. Lambert, R. Cahen, T. Gaudy, F. Guedy, F. Gosselin, and P. Mobuchon. Study of haptic and visual interaction for sound and music control in the phase project. In *Proceedings of the Conference on New Interfaces for Musical Expression*, pages 109–114, Vancouver, Canada, 2005.

K. Salisbury, F. Conti, and F. Barbagli. Haptic rendering: introductory concepts. *IEEE Computer Graphics and Applications*, 24(2):24–32, 2004.

L. Sasaki, G. Fedorkow, W. Buxton, C. Retterath, and K. Smith. A touch sensitive input device. In *Proceedings of the International Computer Music Conference*, pages 293–297, Denton, Texas, 1981.

M. Scaife and Y. Rogers. External cognition: how do graphical representations work? *International Journal of Human-Computer Studies*, 45(2):185–213, 1996.

G. Scavone. STK homepage, 2012. `http://ccrma.stanford.edu/software/stk/`. Accessed 12 May 2012.

G. Scavone and P. Cook. RtMidi, RtAudio, and a synthesis toolkit (STK) update. In *Proceedings of the International Computer Music Conference*, pages 327–330, Barcelona, Spain, 2005.

W. A. Schloss. Using contemporary technology in live performance: The dilemma of the performer. *Journal of New Music Research*, 32(3):239–242, 2003.

B. J. Scholl. Objects and attention: the state of the art. *Cognition*, 80(1-2):1–46, 2001.

B. Shneiderman. Direct manipulation: A step beyond programming languages. *IEEE Computer*, 16(8):57–69, 1983.

B. Shneiderman, C. Williamson, and C. Ahlberg. Dynamic queries: database searching by direct manipulation. In *CHI '92: Human factors in Computing Systems*, pages 669–670, New York, NY, USA, 1992. Association for Computing Macinery.

S. Sinclair and M. M. Wanderley. A run-time programmable simulator to enable multimodal interaction with rigid-body systems. *Interacting with Computers*, 21(1-2):54–63, 2009.

S. Smallwood, D. Trueman, P. R. Cook, and G. Wang. Composing for laptop orchestra. *Computer Music Journal*, 32(1):9–25, 2008.

SMCQ (La Société de musique contemporaine du Québec). Light Music program notes, 2009. `http://www.smcq.qc.ca/smcq/en/oeuvres/248/61.php`. Accessed 9 December 2012.

S. Soto-Faraco and A. Kingstone. Multisensory integration of dynamic information. In G. A. Calvert, C. Spence, and B. E. Stein, editors, *The Handbook of Multisensory Processes*, pages 49–67. MIT Press, 2004.

L. Spiegel. *The Music Mouse Manual*. 1995. `http://retiary.org/ls/progs/mm_manual/mouse_manual.html`. Accessed 5 August 2012.

C. Stuart. The object of performance: Aural performativity in contemporary laptop music. *Contemporary Music Review*, 22(4):59 – 65, 2003.

J. Szpirglas. L'informatique du mouvement. *Thierry De Mey, cahier spécial Mouvement*, 59:35–37, 2011. `http://www.compositeurs.be/pdf/presse/47-CECN_Thierry_de_Mey_janv2011.pdf`. Accessed 9 December 2012.

D. Trueman. Why a laptop orchestra? *Organised Sound*, 12(02):171–179, 2007.

D. Trueman, P. Cook, S. Smallwood, and G. Wang. PLOrk: the Princeton laptop orchestra, year 1. In *Proceedings of the International Computer Music Conference*, pages 443–450, 2006.

B. Tversky, J. B. Morrison, and M. Betrancourt. Animation: can it facilitate? *International Journal of Human-Computer Studies*, 57(4):247–262, 2002.

L. Tweedie. Characterizing interactive externalizations. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 375–382, Atlanta, Georgia, United States, 1997. ACM.

D. Van Nort. Montreal genetic laptop orchestra, 2006. `http://www.music.mcgill.ca/~doug/mglo.html`. Accessed 5 August 2012.

D. Van Nort. *Modular and Adaptive Control of Sound Processing*. Ph.D. dissertation, McGill University, 2009.

D. Van Nort and M. Wanderley. Control strategies for navigation of complex sonic spaces. In *Proceedings of the Conference on New Interfaces for Musical Expression*, pages 379–382, 2007.

D. Van Nort, M. M. Wanderley, and P. Depalle. On the choice of mappings based on geometric properties. In *Proceedings of the Conference on New Interfaces for Musical Expression*, pages 87–91, 2004.

P. Viviani and C. Terzuolo. Trajectory determines movement dynamics. *Neuroscience*, 7 (2):431–437, 1982.

J. T. von Falkenstein. Gliss : An intuitive sequencer for the iPhone and iPad. In *Proceedings of the Conference on New Interfaces for Musical Expression*, pages 527–528, 2011.

Wacom Co., Ltd. *Wacom Intuos User's Manual for Macintosh, English V3.1*, 2000.

Wacom Co., Ltd. *Installation Guide & Hardware Manual for Cintiq 21UX (DTZ-2100D), English version 2.2*, 2007.

M. Waisvisz. Crackle history, 2004. `http://www.crackle.org/CrackleBox.htm`. Accessed 5 August 2012.

M. M. Wanderley and N. Orio. Evaluation of input devices for musical expression: Borrowing tools from HCI. *Computer Music Journal*, 26(3):62–76, 2002.

G. Wang, D. Trueman, S. Smallwood, and P. R. Cook. The laptop orchestra as classroom. *Computer Music Journal*, 32(1):26–37, 2008.

D. Wessel and M. Wright. Problems and prospects for intimate musical control of computers. *Computer Music Journal*, 26(3):11–22, 2002.

M. Zadel. A software system for laptop performance and improvisation. Master's thesis, McGill University, 2006.

M. Zadel and G. Scavone. Laptop performance: Techniques, tools, and a new interface design. In *Proceedings of the International Computer Music Conference*, pages 643–648, New Orleans, LA, 2006a.

M. Zadel and G. Scavone. Different strokes: a prototype software system for laptop performance and improvisation. In *Proceedings of the Conference on New Interfaces for Musical Expression*, pages 168–171, Paris, France, 2006b.

M. Zadel and G. Scavone. Recent developments in the Different Strokes environment. In *Proceedings of the International Computer Music Conference*, pages 1–4, Belfast, Northern Ireland, 2008.

M. Zadel, S. Sinclair, and M. M. Wanderley. Haptic feedback for Different Strokes using DIMPLE. In *Proceedings of the International Computer Music Conference*, pages 291–294, 2009.

M. Zbyszyński. An elementary method for tablet. In *Proceedings of the Conference on New Interfaces for Musical Expression*, pages 245–248, 2008.

M. Zbyszyński, M. Wright, A. Momeni, and D. Cullen. Ten years of tablet musical interfaces at CNMAT. In *Proceedings of the Conference on New Interfaces for Musical Expression*, pages 100–105, New York, New York, 2007. ACM.

D. Zicarelli. M and Jam Factory. *Computer Music Journal*, 11(4):13–29, 1987.

C. Zilles and J. Salisbury. A constraint-based god-object method for haptic display. In *Proceedings of the 1995 IEEE/RSJ International Conference on Intelligent Robots and Systems*, volume 3, pages 146–151, 1995.