MetaMuse

Using metaphor to design and play expressive instruments.

by

Ashley Gadd

B.A.Sc., University of British Columbia, 2000

A THESIS SUBMITTED IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE DEGREE OF

Master of Applied Science

in

The Faculty of Graduate Studies

(Electrical and Computer Engineering)

The University Of British Columbia

April 26, 2005

ABSTRACT

This thesis provides a framework for understanding and predicting the expressive nature of devices. We introduce *transparency*, or the ease with which a device's mapping can be understood, as the main pillar of this framework. Transparency is a predictor for *expressivity*, the ease with which meaning or emotion can be communicated through the device. We explore the role of metaphor for improving the amount of expression possible with a device, and examine its use as both a design aid and an affordance during use.

To validate this theory we built MetaMuse, a controller for a rain-sound synthesiser that employs water metaphors to obviate its use. MetaMuse uses physical props to embody three metaphors: pouring, rainfall, and landscape. These metaphors acted as a guide during MetaMuse's design, and aid users in the prediction of system operation and sound output.

We performed a series of experiments comparing MetaMuse to two other interfaces to test our claim that metaphor can be used to improve expressivity. The tests measured controllability with speed and accuracy of sound target acquisition, and expressivity with qualities such as creativity. Accuracy was similar across all interfaces, and speed slightly improved for MetaMuse, but the controllability results lacked significance. Results for expressivity were significant, however: users preferred MetaMuse as more creative than two more traditional controllers ($F=6.0$, $p<0.01$). From these results we conclude that metaphor is a worthy design approach to creating expressive devices.

This thesis also presents a novel form of synthesis combining techniques from real-time stochastic synthesis and granular synthesis. Finally, it includes a software library for simplifying the use of the Polhemus Fastrak magnetic sensor used in the project.

CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Why is it so difficult to create a novel expressive musical device? This paper provides a framework for understanding and predicting the expressive nature of devices. We introduce *transparency*, or the ease with which a device's mapping can be understood, as the main pillar of this framework. Transparency is a predictor for *expressivity*, the ease with which meaning or emotion can be communicated through the device. We explore the role of metaphor for improving the amount of expression possible with a device, and examine its use as both a design aid and an affordance during use. Metaphor depends on a *literature* — a common body of knowledge which forms a basis for improving transparency.

To validate this theory we built MetaMuse, a controller for a rain-sound synthesiser that employs water metaphors to obviate its use. MetaMuse uses physical props to embody three metaphors: pouring, rainfall, and landscape (see Figure 1.1). These metaphors acted as a guide during MetaMuse's design, and aid users in the prediction of system operation and sound output. Our experiments with MetaMuse support our claim that metaphor can be used to improve expressivity. Users significantly preferred MetaMuse as more creative than two more traditional controllers.

1.1 Transparency of Device Mappings

Our discussion centres around device mappings. We consider mappings to encapsulate the entire process of translating the player's control gestures into sound output. In the example of a piano, the mapping includes everything from the position and shape of the keys, through the physics of the hammers and strings, to the acoustic properties of the piano body itself. In Chapter 3, we explore those qualities of device mappings that are important to expression. Traditional instruments such as violins and pianos have direct, physical mappings from the player's gestural input to the sound output, and are therefore easily understood. In computer-based instruments, the map-

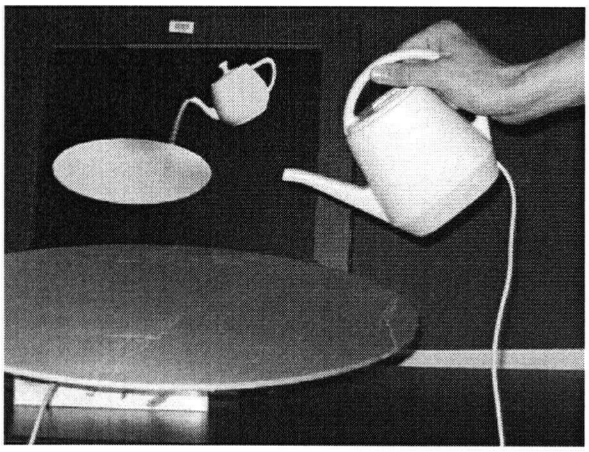


Figure 1.1: MetaMuse. In the foreground, we see the watering can and landscape props used to control a rain sound synthesiser. In the background the props are rendered on the screen along with the virtual water that flows between them.

ping is arbitrary. Many new instruments have non-obvious, or opaque, mappings, which make it difficult for the player or the audience to understand the instrument, creating a barrier to expression.

We introduce transparency as a quality of a mapping. Similar to Moore's notion of control intimacy [41], transparency provides an indication of the psychophysiological distance, in the minds of the player and the audience, between the player's input and the sound output of a device mapping. The more transparent the mapping is, the more expressive the device can potentially be. Since new technologies are often poorly understood they tend to produce opaque mappings. Metaphor is one technique that facilitates moving from an opaque mapping to a transparent mapping.

Metaphor enables device designers, players, and audiences to refer to elements that are "common knowledge" or cultural bases which we call *literature*. The Oxford Dictionary [19] defines literature as "books & written composition esp. of the kind valued for form & style, the production of these or their authors as a class, the realm of letters, the writings of a country or period, *the* books & c. treating *of* a subject, (colloq.) printed matter." We use the term in a broader sense, referring to all cultural common knowledge rather than just published works. By grounding a mapping in the literature, it is made transparent to all parties.

Metaphor restricts and defines the mapping of a new device. This can aid in the design process by providing a central theme around which the designer can build. As we shall in see Chapter 3 this theme can aid in matching existing synthesis techniques to new controllers or vice versa, or can provide a new starting point for choosing both the controller and the synthesiser. Similarly, metaphor can provide the player and the audience with a better understanding of the device during performance, facilitating communication. Through metaphor, transparency increases, making the device more expressive.

The framework of expressivity, transparency, and metaphor is presented in this thesis with respect to computer-based sound and music devices. It may also be applied to other fields of human interaction, including human-human, human-computer, and human-machine interaction.

1.2 MetaMuse

We built MetaMuse, discussed in Chapter 4, to explore our theory of transparency. MetaMuse is a rain-sound synthesiser that is controlled by two props: a watering can to simulate pouring and a surface on which to pour (see Figure 1.1). The prop-based control of MetaMuse was designed and built based on metaphors of rainfall, which provide a cognitive match to the process used by the synthesis engine. These metaphors motivated the design and construction of the instrument and assist in learning to play it.

MetaMuse is defined by three metaphors closely related to water or rainfall: *pouring*, *rainfall*, and *landscape*. Virtual water droplets are created, fall due to simulated gravity, and make a sound when intersecting a surface. The pouring metaphor suggests how users create water droplets, while the rainfall metaphor dictates how the droplets will behave. Finally, the landscape metaphor indicates that the sounds of the droplets change with the type of virtual surface the rain lands on.

The props used to control the system were chosen to suggest and reinforce the metaphor. A small watering can supports the pouring metaphor, while a round palette provides the surface on which the virtual water pours. Magnetic sensors are used to measure the props' positions and orientations. This allows a graphical representation of the props and virtual water to be viewed on a computer monitor.

The props control a parametric granular synthesiser based on a stochastic real-time synthesis model. Granular synthesis is a technique that blends multiple discrete sounds to create a single auditory effect. Rainfall's similarities to this process make it an excellent metaphor for such a synthesiser. Indeed, the match of metaphor to synthesiser motivated the design of the props used as controllers.

1.3 Contributions and Lessons Learned

In a series of user experiments, discussed in Chapter 5, we evaluate MetaMuse to explore the validity of our theory. The tests measured how *controllable* and *expressive* the system is compared to a baseline of graphical user interfaces controlling the same synthesiser. The results suggest that MetaMuse is an expressive instrument. In particular, users found MetaMuse to be more creative than the other interfaces. Statistical trends supported the hypothesis that MetaMuse is no less controllable than the baseline interfaces, but these results were not significant. These non-significant trends indicate a need for further research.

Users' comments during the post-experiment interview were telling. One user indicated that MetaMuse "just felt more intuitive," while another reported that "using the prop, the sounds sound like what you would expect." The metaphors embodied in the interface, for these users, offer a clear advantage to understanding the device.

The conclusion that MetaMuse is an expressive instrument is important. It implies that the metaphor-based design process and the embodiment of those metaphors in the interface were successful. From this we can conclude that metaphor is an important tool for designing and playing expressive instruments.

Informal observations made during testing also support this conclusion. The primary difficulty users had with MetaMuse was in determining *where* on the landscape to find a certain sound. In retrospect, this result should have been predicted. The landscape metaphor implies that the sound will change as droplets are moved across the surface, but doesn't suggest *how* it will change. That transparency and metaphor can not only aid in creating expressivity, but also predict how a device might fail to be expressive, strengthens the theory.

A final contribution of this thesis is the Polhemus library. Created to facilitate communications with the Polhemus Fastrak magnetic sensor [14], this library has found use in several other projects and has been modified to work with other sensing devices. Since it is not central to the thesis, it is discussed in more detail in Appendix C.

To summarise, this thesis contributes the following:

1. The theory of transparency, which provides a framework to predict and evaluate the expressivity of musical devices.

2. The investigation of metaphor as a facilitator for designing expressive devices, and for learning and playing them, by increasing transparency.

3. Demonstration of the above concepts in the form of MetaMuse, a novel expressive device based on metaphor. MetaMuse and its evaluation show that transparency and metaphor are useful constructs.

4. Parametric granular synthesis, a novel form of synthesis combining techniques from real-time stochastic event modeling and granular synthesis.

5. The Polhemus library, which provides simplified multi-threaded access to the Polhemus Fastrak and creates a standard framework that extends to other serial devices.

CHAPTER 2

RELATED WORK

To fully understand metaphor as a way to design and play expressive instruments, we must consider previous attempts at creating such instruments. Past computer-based musical instruments have been built based on many different principles. Some have been motivated by technology, and are based on either sensor and controller technologies, or current hardware or software synthesisers. Such systems tend to lack completeness — when the overriding design principle only takes into account one aspect of the instrument, its expressivity will suffer. Other systems are designed based on accepted HCI principles such as usability, learnability, and discoverability. While these are important design elements, they fail to address expression.

Our approach to computer-based expressive devices focuses on the mapping: the way in which the user's input gestures are mapped to sound output. Since the mapping extends from input interface to output interface, it takes both interfaces into account. When we apply our metaphor to this mapping it encompasses the entire device, providing a vital central theme that guides both design and play of the device. To understand metaphor, we will consider previous systems that use it.

In Chapter 4 we will see that MetaMuse is a prop-based controller to a rain-sound synthesiser. To complete our exploration of previous work we will explore both of these topics.

2.1 Previous Approaches

The problem of classifying musical instruments has been approached in many ways, with the classical approach being that of Hornbostel-Sachs [65]. In their system, instruments are classified based on the physical mechanism of sound production. The violin, for example, is classified as a chordophone, or stringed instrument, and subclassified as bowed. The clarinet is classified as aerophone > pipe aerophone > reed pipe instrument > single reed. Computer musical instruments are much more difficult to classify because of their lack of physicality [59].

One approach is to categorise them by the input technology used [9]. Different types of controllers affect the performance of the instrument; examples include free gesture sensing at various scales, biometric sensing, and variations or extensions of input techniques for traditional musical instruments. Properties of the controller, such as its continuous or discrete nature, have a profound effect on the instrument.

Output technology is also used to categorise new instruments. Different synthesis engines are used depending on whether the instrument is intended for performance, composition, or sound design. Once again, properties of the output technology fundamentally affect the resultant instrument.

A third approach to the domain is from the field of computer-human interfaces [48]. The design of a usable interface is the primary aspect of this approach, with an emphasis on analysing discoverability, learnability, usability, controllability, and similar properties. For example, a drum has a very low barrier to novices — it is very easy to learn to hit a drum with a drumstick. Similarly, the drum has good controllability and usability as its interface is very straightforward. Transferring these properties to more complex computer-based musical instruments is a challenge that the field of HCI has many well-documented approaches for tackling. These properties are will be revisited in later sections.

The approach taken herein focuses on the mapping used in the device. We consider mappings to encapsulate the entire process of translating the player's gestures into sound output. Since the mapping entails knowledge of both the input and the output interfaces, consideration of these aspects of the instrument are included in this approach. More importantly, mapping is one of the major challenges currently facing computer musical instrument practitioners [31].

As we shall see in the next section, computer musical instruments lack a physical connection between their control input and their sound output [46]. This is problematic for instrument designers, who lack guidelines for their creations, and for performers and their audiences, who are no longer able to share expression through the instruments. Many designers have based their devices on existing instruments to resolve this shortcoming, but this practice limits the designer to existing classes of instruments.

We suggest metaphor as a way to create new classes of instruments based on different controllers and synthesis techniques. Metaphor provides a basis for understanding the mapping of a device, allowing novel devices to be more

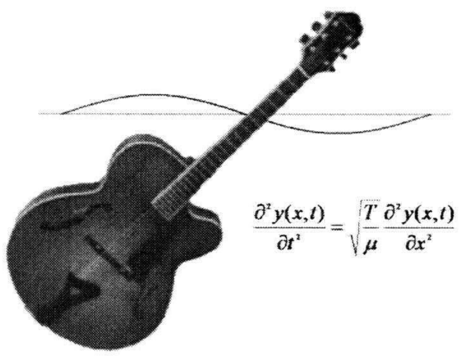

$\frac{\partial^2 y(x,t)}{\partial t^2} = \sqrt{\frac{T}{\mu}} \frac{\partial^2 y(x,t)}{\partial x^2}$

Figure 2.1: In a traditional instrument, the physics of the device dictate the manner in which it is played. In the case of this guitar, the vibrating string is controlled by manipulating the variables of the wave equation, in particular string length and tension.

easily understood. In particular, we examine the use of gestural controllers based on props to control granular synthesis. The following section motivates the mapping-oriented approach to the field of computer musical instruments that is used throughout this chapter and affects this thesis.

2.2 The Importance of Mappings

Traditional musical instruments have a clear connection between their control inputs and the sounds they produce [8, 46], as shown in Figure 2.1. This connectedness is entirely physical: from the standing wave on a guitar's plucked string to the column of air in a flute, the physics of the sound production mechanism dictates how the instrument is played. Physics also guides many aspects of instrument design, creating classes: bowed string, woodwind, percussion, and so on [65]. Within a class there is a limited number of ways to build an instrument and a limited number of ways to play it.

Modern computer-based musical instruments lack the natural connection of traditional instruments [24, 46]. As shown in Figure 2.2, synthesis engines are controlled by a stream of numerical parameters. Input devices produce similar streams of parameters. The way in which these parameters are mapped from input to output can be completely arbitrary. While this arbitrariness infinitely expands the number of possible ways to build an in-

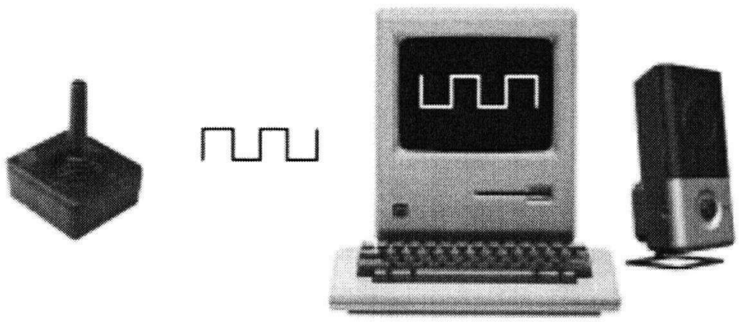


Figure 2.2: In computer-based instruments, there is no inherent connection between the control and the sound output. The signal from the controller is converted into a series of ones and zeros that can be arbitrarily transformed by the computer. In effect, the computer is a black box device.

strument, it also creates an equal number of ways to play them. Additional classes of instruments can come at the expense of expressiveness.

An example of this lack of connection is found in our own unpublished research. A precursor to MetaMuse attempted to control a synthesis engine based on the posture of the hand, as measured by a CyberGlove dataglove [11]. Principle Component Analysis was used to match the input parameter space to the output parameter space, so that gross hand movements corresponded to large changes in sound and fine movements to subtler nuances. This method has also been used in systems such as Glove-TalkII [15]. While our model was mathematically correct, it was meaningless to the player — hand movement caused changes in sound output, but it was unclear how the two were correlated. The resulting instrument proved impossible to play.

Other instruments that suffer from similar problems are laptop bands, Expressive Footwear [51], and Musical Trinkets [52]. Laptop bands exemplify opacity in computer music: one or more people on stage interacting with laptops to produce synthesised music. Because the audience knows nothing about the software running on the laptop or its control, expressivity is difficult. Even other musicians may have trouble determining the specific configuration of the laptop. Expressive Footwear uses instrumented shoes to control music, but the parameters of control (sole flex, heel pressure, etc.)

are very difficult for the audience to perceive. Musical Trinkets are based on passive magnetic tags built into small toy figures. While not intended to be complex, they remain confusing because of the difficulty in determining what synthesis effect each figure represents.

In general, this non-connectedness results in instruments that are harder to learn and harder to play virtuosically [49, 66], or instruments that are so simplistic virtuosic performance is ruled out. Often they suffer from stability problems: Cook [10] discusses the infinitely customizable interface, which can never be learned because its mapping changes from one session to the next. Non-connectedness also causes problems for the audience, who no longer understand the correlation between what they see and what they hear [24].

Instrument designers have attempted to solve the problem of non-connectedness in many ways. Instruments have been made that are so simple it is very easy to perceive their mappings [5]; others use algorithms that force the music to sound good under any input. The simplest method, however, is to fall back on the known, physical connection that makes traditional instruments so expressive.

2.2.1 Traditional Musical Instruments as a Basis for Creating New Instruments

Attempts to re-establish the connection between input and output have mainly centred around recreating or extending traditional musical instruments. The most prolific example of this practice is the ubiquitous keyboard synthesiser, which imitates the piano in form and function. For many players, the two are equivalent, though progress is still being made in duplicating the feel of the piano keyboard exactly [45]. Other instruments have been similarly duplicated to various degrees of success, including the violin [22], the clarinet [67], and the drum [13].

Other research has extended the functionality of an existing instrument by adding to it or changing its form. BoSSA [2, 63] decomposes the physical components of the violin, allowing additional control in the form of neck bending. The Accordiatron [24] is a musical controller that duplicates the action of an accordion but allows control of different musical parameters. Jam-O-Drum [6] uses the basic interface of a hand-struck drum surface to create a collaborative social environment with both musical and visual components.

Recreating or extending traditional musical instruments in the electronic domain is clearly a worthwhile pursuit; many interesting and effective instruments have been created. Relying solely on such techniques, however, inherently limits the instruments to classes that already exist — BoSSA still follows the bowed string interaction paradigm; Jam-O-Drum is still a percussion instrument. In order to create novel classes of instruments, other approaches must be pursued.

2.2.2 Other Approaches to Creating New Instruments

There have been a multitude of approaches to new musical instruments that don't involve traditional instruments as their basis. New sensors and synthesisers have allowed many new classes of instruments to be created. Some of these approaches are presented in this section.

Whereas traditional musical instruments have, by necessity, involved a one-gesture, one-output relationship, the decoupling afforded by electronic instruments allows different structures. Composed instruments [57] blur the boundary between performance and playback, with the performer controlling individual sounds as well as affecting precomposed scores.

Tooka [18] is a two-person musical instrument based on breath control. While it is very similar to a recorder, the lack of a vibrating column of air allows the players to blow into both ends of the device. The primary control parameter is the pressure in the tube, creating a new class of instruments.

2Hearts [40] also extends instruments into two-person interactions, but in this case the instrument doesn't exist *per se*. In 2Hearts, musical flow is controlled by the interaction of the two players' heartbeats. Since one has little control over one's own heart rate, the instrument played by each of the participants becomes the other participant.

By instrumenting different vessels, Tangible Sound 2 [39] turns the flow of water into a musical instrument. By directing a stream of water into various vessels, different sounds and sound combinations can be created. This approach produces an instrument for which the number of players is limited only by the physical space available.

Several instruments, such as Piano Cubes [54] or Musical Trinkets [50], make the creation of music accessible to anyone. Piano Cubes are small cubes whose position and orientation affect the properties of pre-scored piano music. Musical Trinkets are toys that affect synthesised music in complex ways. Both instruments are easily approachable by novices.

Many of these instruments have been successful, but some have problems that relate directly to standard HCI principles. Cook's infinitely customizable interfaces, discussed earlier, present a lack of interface consistency. Piano Cubes and Musical Trinkets oversimplify their control, making the device easy to learn but lacking in long-term engagement. Other instruments have interfaces that are too obscure, lacking discoverability and usability and therefore making it difficult for both player and audience to understand the instrument (this issue will be revisited in greater depth in Chapter 3). These and other difficulties have been discussed by Cook and Wessel [66], among others.

A consistent metaphor is one way to overcome these obstacles and design complex, understandable instruments.

2.2.3 Metaphor as a New Basis for Creating New Instruments

One way to create a connection between the control input and sound output of an instrument is to use metaphor. Metaphor allows the player and the audience to create a mental model of the system by likening it to something that is already understood. This is formalised in semiotics, where the term *signifier* denotes the device in question and the *signified* is the concept to which it refers [36]. Those who are familiar with the signified will also understand the signifier. In other words, if the metaphor matches the system appropriately the new instrument will be understood. In this way metaphor allows us to create new classes of instruments by referring to things other than existing musical instruments.

Metaphor has provided a basis for non-musical systems, and its strengths and shortcomings are well known [1, 29, 44]. One well-known example is the desktop metaphor used by most modern personal computers. In order to help new users make the transition from the real world to the computer, the screen is modeled as a desktop. Documents and folders are modeled as objects on the desktop surface, and open applications appear as overlapping sheets of paper. Where the desktop metaphor falls short, however, is in applications that don't fit that metaphor, such as online shopping. There is no desktop analogue to the Internet, so the metaphor must be broken to provide this ability. This example reinforces the idea that metaphors are inherently restrictive. This is less of an issue when designing a single-purpose device such as a musical instrument than it is for an extensible system such as a general-purpose computer. However, it must be taken into consideration when designing a system around a metaphor.

Metaphor has also been used in sound-based systems with mixed success. Sound Sculpting [42] makes use of metaphor by presenting a virtual rubber sheet, which can be stretched and pulled to modulate a sound as it played. Some aspects of this mapping are properly explained by the metaphor — moving the surface left to right pans the sound, for example. Interestingly, the sound change caused by physically "stretching" the sheet, though difficult to describe, also fits the metaphor well. Other aspects of the mapping, such as volume or reverberation, have no equivalent in the signified, and are therefore non-obvious and difficult to learn. The rubber sheet metaphor is successful where it matches the required control but is not successfully extended to other controls.

Another system that uses metaphor is the tymbalimba [58]. This system's controller duplicates the cicada's sound-producing mechanism to control a synthesiser. In this case, the attempted range of control does not extend beyond the metaphor's explicative abilities, and the resulting instrument is quite engaging.

We have explored several instruments whose mappings make use of metaphor. We consider the mapping to be the main component of musical devices, and will expound on its relevance in Chapter 3. Also important to the device are the input and output interfaces, which specify how the user interacts with the mapping, and how the mapping produces sound output. In the following sections we will examine props and free gesture, which are relevant to our input interface, and the synthesis techniques used to create the output interface.

2.3 Props and Free Gesture

MetaMuse uses props to help define the input interface to the mapping. Prop-based interfaces are similar to those using free gesture, with the props providing a frame of reference that improves the usability of the interface. Free gesture and props have been used in previous systems, both expressive and practical. We will examine free gesture, its shortcomings, and the use of props to resolve those shortcomings.

2.3.1 Free Gesture and Its Shortcomings

Free gesture is a common method for interhuman communication. It has been applied to computer interfaces, and computer music interfaces specifically, because it allows the simultaneous control over multiple parameters. However, it lacks precision [49], making expressive musical timing difficult.

One example of a free-gesture instrument is the Imaginary Piano [61]. This instrument uses optics to determine the player's hand positions in space, and triggers notes when the hands enter certain zones. By setting the zones in the form of a piano keyboard, the Imaginary Piano is created.

Glove-TalkII, first discussed in Section 2.2, is a gestural controller that is not musically-oriented. The absolute position and shape of the hand control a formant synthesiser, turning free gesture into human vocal sounds. It allows speech and singing, but is limited by a lack of pre-audio feedback.

Both of these devices suffer from the same problem: they are bound to an external frame of reference. This makes timing and selection difficult to do accurately, as the player has no feedback from the device until its audio output is heard. One way to improve on gestural interfaces is to obviate the frame of reference.

In Section 2.2.3 we discussed Sound Sculpting as a metaphor-based controller. Sound Sculpting uses a relative framework — the player's hand positions are determined relative to one another. This, coupled with the system's compelling metaphor of object manipulation, allows the user to create more accurate positions. These are sufficient for the system's purpose of sound exploration rather than musical performance.

2.3.2 Props: Providing a Frame of Reference

One way to provide a frame of reference is to use props. Though force feedback is lacking, props help the user to orient about a frame of reference. Guiard [23] showed that people use the non-dominant hand to create a working reference for the dominant hand. Balakrishnan [3] extended this research to determine the importance of matching the kinesthetic and visual reference frameworks. In other words, the frame of reference set up by the non-dominant hand will be successful if it is matched by the system's visual feedback.

Props have been successfully demonstrated in non-musical devices such as a neurosurgical visualization tool [25] and the ToolGlass [4, 35]. In the former system, a doll's head prop in the left hand sets the context for the cutting plane prop in the right hand, allowing the user to specify the slice of volumetric data he wishes to view on screen. In ToolGlass, a second mouse is used in a drawing program to create a frame of reference for the primary mouse, effectively creating a tool set that is positioned with the non-dominant hand.

Clearly, props provide a referential framework that is not present in free-gesture interfaces. Before determining whether props can be used to create an expressive device in the following chapter, we need to consider the output interface for such a system.

2.4 Synthesis Techniques for New Musical Interfaces

MetaMuse is based on a granular synthesiser that uses a real-time stochastic synthesis process instead of pre-recorded samples. Granular synthesis was the genesis of the project, motivating the choice of metaphor and thus the rest of the system design. Once the system was constructed, the synthesiser was found to be insufficient for the metaphor because of its use of discretely-selected pre-recorded samples, so the sound source was replaced with a real-time stochastic synthesiser.

2.4.1 Granular Synthesis

Granular synthesis, first described by Truax [62], creates a continuous sound by playing short, prerecorded samples, overlapping and in quick succession. The sound creation process is similar to that of hand-held percussive instruments such as maracas and shakers: a series of small, discrete sounds ("granules") that blend together to create a larger effect. Whereas maracas tend to provide sounds in bursts, granular synthesis allows a continuous sound that could be likened to a cabasa being steadily rotated. However, the possibility of basing the granules on any sound sample or combinations of samples greatly increases the scope of the synthesiser. There is potential here for new classes of instruments depending on the discrete sounds used.

Previous controllers for granular synthesis have been purely statistical. These include the controller used by Truax himself and an implementation of granular synthesis used in jMax [12]. These implementations are based on a relatively lengthy prerecorded sound sample, from which the shorter granules are played. The controllable parameters include the starting point, duration, and playback rate of the granules and the number of granules played per second. In these implementations, these parameters are set according to simple statistical processes with means and standard deviations. While this method proves the concept of granular synthesis, it affords little in the way of expressive control. In Chapter 4 we shall see how these statistical processes were replaced with more expressive controls.

Since MetaMuse was first published [20, 21], two systems, Pebblebox and Crumblebag, have been developed to use real granules to control granular synthesis [47]. In these systems, the sound made by physically manipulating several small items is analysed to extract quantities such as number of collisions and collision intensity. These parameters are then used to control a granular synthesiser using arbitrary sound samples. Effectively, the sound of the physical objects interacting is modified by the computer. This is a very tangible interface that makes its interaction paradigm clear at a glance.

2.4.2 Physical Synthesis

Once MetaMuse was constructed, a mismatch was found between the granular synthesiser and the control techniques used. This mismatch is described in greater detail in Section 4.3.2. In short, MetaMuse affords continuous control, but the underlying process for granular synthesis discretely selects the originating sound sample. A new synthesiser was required to sit underneath the granular synthesis process.

The Synth ToolKit [56] (STK) by Perry Cook provided the needed synthesiser. STK includes "PhISEM," Physically Informed Stochastic Event Modelling. This synthesis technique uses stochastic processes to excite a filter bank, producing a variety of possible sounds. STK's PhISEM can reproduce the sounds of maracas, bamboo wind chimes, tambourine, and other such instruments fairly accurately, and uses continuous parametric control. It was used as a starting point for our own synthesiser.

Physical synthesis is also used in van den Doel's liquid sounds synthesiser [64]. This synthesiser identifies bubbles as the primary cause of sound in water environments. It uses a stochastic model to simulate the sounds of many bubbles, and is reported to be fairly accurate for a wide range of rain sounds. Unfortunately it was not developed until after the MetaMuse system was completed.

This foundation of previous work provides a launching pad for the rest of the thesis. We will now examine the MetaMuse project in greater detail, starting with its theoretical basis.

CHAPTER 3

TRANSPARENCY, EXPRESSIVITY, AND LITERATURE

We consider expression to be an act of communication, in which the player and the listener are both responsible for determining to what extent a performance is expressive. Expression is the act of communicating meaning or feeling. Both player and listener are involved in an understanding of the mapping between the player's actions and the sounds produced. The mapping, and the ease of understanding it, are therefore critical to determining the success of an instrument.

Both player and listener understand device mappings of common acoustic instruments such as the violin. This understanding is possible because the instrument's physical nature obviates the control mechanism and its form factor allows the instrument to be learned. Both participants are able to make a clear cognitive link between the player's control effort and the sound produced, facilitating the expressivity of the performance. We recognise that expressivity is not guaranteed: expression is complex, having dimensions apart from transparency that contribute to it. For example, the mapping of a tuning fork is transparent, but few would argue that such a simple device is particularly expressive. Worded formally, transparency is not a sufficient condition for expression.

For many instruments, the cognitive link is sufficiently integrated into the culture as to make it bi-directional. In this situation, observing either the sound or the effort provides access to the other. For example, one can picture the vigorous sawing of a virtuoso violinist while listening to an audio-only recording of a particularly exuberant performance. Likewise, watching a good pantomime of a vigorously sawing virtuoso violinist evokes an expressive sound performance. Together, the effort and the sound reinforce one another, increasing the expressivity of the performance.

Instruments with a strong link between control effort and sound are more likely to become part of the literature. Here we are distinguishing the concept of literature from its literal definition of *that which is written*. What we intend is the more general definition of that body of knowledge understood and accepted as part of a culture. It is *common knowledge* and is used as a referent rather than being explained by reference to something else. For example, scents are often compared to that of a rose, but the scent of a rose is rarely identified by comparison to something else[1]. Literature in this sense has a cultural basis that the designer must be aware of when considering users of the device.

3.1 Transparency of Device Mappings

One of the key attributes of instruments required for adoption into the literature is expressivity; this is a necessary condition for acceptance. We argue that the expressivity of an instrument is in part dependent on the transparency of the mapping, or the ease with which the device's mapping can be understood by both the player and the audience. With this factor in mind, we can attempt to identify how an instrument, based on a new technology, can make its way into the literature and become a referent. This course depends in large part on the mapping from control to sound.

The mapping component is placed within the larger context of the instrument or device in Figure 3.1. The device itself is composed of three parts: the input interface, the mapping, and the output interface. The input interface consists of the set of control gestures used to control the device. This is different from the physical input device, which can restrict or suggest certain control gestures but also interprets them, so has a mapping aspect. The output interface consists of the possible range of sound outputs that the device can make, as distinct from the actual synthesis engine used. The mapping defines how the control gestures translate into sound output and comprises the whole system, from the input interface to the output interface. This is important because understanding of the mapping is critical to the expressivity of the device.

In the case of traditional acoustic musical instruments, physics drives the mapping between control and sound. Traditional instruments are typically

[1]One notable exception: rose scents are identified by comparison to other things when differentiating scents between breeds of roses.

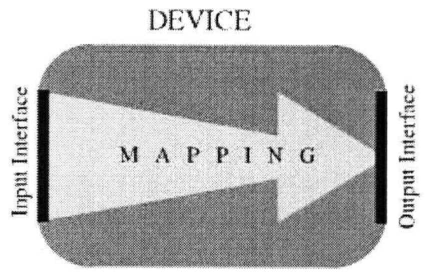


Figure 3.1: The instrument has an input interface and an output interface. The two are related by the mapping.

implemented with mechanical systems. As such, the mapping usually is easily understood by the player. Further, the physical form factor makes learning to play the instrument possible on a reasonable human time scale. These two factors make the mapping between instrument control and sound production psychophysiologically *transparent* for the player. Similarly, the audience's understanding of the instrument benefits from the physical nature of the mapping. The audience also benefits from a long cultural association with traditional instruments, expecting certain inputs to result in certain outputs. Both of these factors make the mapping transparent for the audience. Transparency for both the player and the audience makes expressivity possible.

As an example, the acoustic guitar is a well-known instrument. The lay audience understands the manner in which the player's control gestures map to sound output, even if they lack the physical proficiency to play the guitar themselves. This common understanding makes the guitar's mapping transparent to the audience. With enough practice, it also becomes transparent to the player. Under these (common) conditions, the guitar is an expressive instrument.

The advent of electronic musical instruments complicates the understanding of whether a musical instrument is expressive. This complication arises because such instruments allow the separation of control from sound [30, 33, 68]. Most modern synthesis engines are controlled by time-varying sets of numerical parameters. These parameters can be produced in many ways and by using many different mappings. This *physical* separation requires an effort on the part of the designer to avoid a corresponding cognitive separation. Many instruments based on these engines have arbitrary mappings, which can make the mapping very opaque to both player and listener. Learning an opaque mapping is difficult for both parties, making expressivity problematic.

The synthesiser keyboard provides an excellent example of how control and sound can become separated. One of the presets for many synthesiser keyboards maps key presses to a variety of percussion sounds. However, the standard mapping, in which pitch increases to the right, is not valid for percussion instruments. This means that the different sounds are mapped somewhat arbitrarily to the keys. While it may be apparent that individual key presses map to individual sounds, the specific mapping is opaque to both the player and the audience. Learning to play percussion on the synthesiser keyboard is very difficult, as is understanding such a performance.

These examples suggest a two-dimensional continuum of mapping transparency, with one axis for the player and one for the audience. We arbitrarily set the range of each axis to be from 0 (opaque) to 1 (transparent), as shown in Figure 3.2. The transparency of the mapping depends on different factors for the player and the audience.

The transparency of a mapping for the player depends both on cognitive understanding and on physical proficiency. Cognitive understanding requires that a player must be familiar with the expected effects of the control parameters on the sound output. Such familiarity can be improved by exposure to performances with the instrument. Proficiency is the level of dexterity that a player has with the controls, and therefore can improve with practice. Thus, familiarity and practice make a mapping more transparent for the player. This concept is very similar to Moore's [41] concept of control intimacy:

> The best musical instruments are ones whose control systems exhibit an important quality that I call "intimacy". Control intimacy determines the match between the variety of musically desirable sounds produced and the psychophysiological capabilities of a practiced performer.

Moore's control intimacy, however, refers to the entire device, whereas transparency refers specifically to the mapping between the input and output interfaces. The player's degree of transparency provides one axis for evaluating and predicting the expressivity of the device.

The audience's degree of transparency provides an orthogonal axis. However, the audience does not require physical proficiency with the interface. Instead, they only need to have an understanding of how the instrument works to appreciate the proficiency of the player. For the lay audience, this

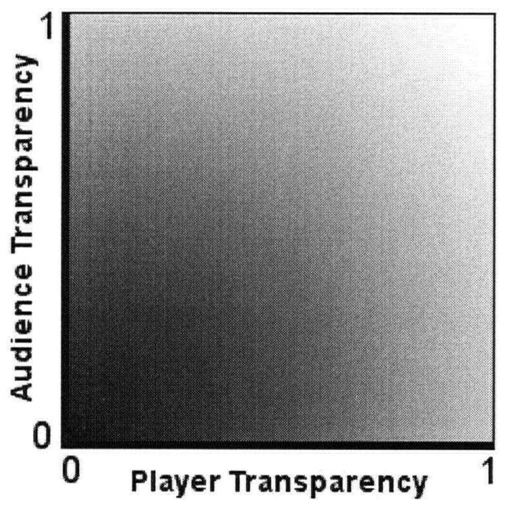


Figure 3.2: The graph created by mapping transparency for the player and for the audience.

understanding is derived from cultural knowledge, including percepts of physical causal relationships, which we have called the literature. Interestingly, this model would predict that it is possible for the *audience* to increase the expressivity of the instrument. This could be accomplished by studying the theory of the instrument or by learning to play the instrument, both of which would increase the transparency of the mapping. Increased transparency contributes to the audience's appreciation of the player's proficiency, leading to increased expressivity.

3.2 A Framework for Expressivity

We have defined orthogonal axes representing mapping transparency for both the player and the audience. Though the axes are continuous, for referential convenience we roughly divide the square into four quadrants corresponding to opaque or transparent for player or audience. Then "transparent/opaque" refers to the region that is transparent for the player but opaque for the audience, and so on. Some existing instruments can be seen on this graph in Figure 3.3.

Most traditional instruments lie in the transparent/transparent quadrant, transparent for both the player and the audience. The violin, for example, is well known to both player and audience due to cultural exposure. The mapping of control gestures to sound output is embodied in the mechanical construction of the instrument. This embodiment, along with the form factor of the instrument, makes the affordances [44] of control apparent to the player and the audience. Because the violin is a culturally familiar instrument, the gestures that control it affect the output in known, predictable ways. These gestures include string choice, finger position, and bowing parameters. The violin's form factor and control predictability also make it learnable on a reasonable human time scale, though many young students may complain to the contrary. These attributes make the violin's mapping transparent for both the player and the audience.

On the other end of the spectrum, many new technologies fall in the opaque/opaque quadrant, opaque for both the player and the audience. New controllers require both parties to learn the mapping from unfamiliar control gestures to existing output interfaces. New synthesiser engines frequently attempt to create novel sound output spaces, which must be mapped from an existing input interface. The worst-case scenario, new controller mapping

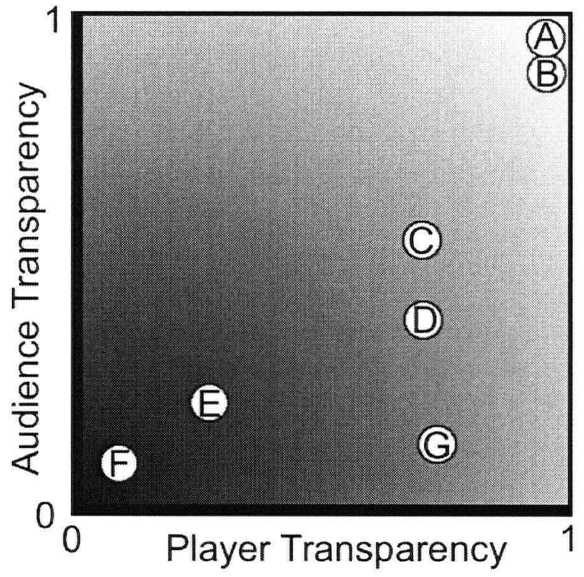


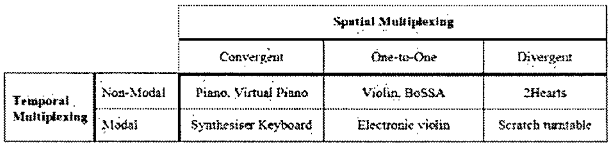
Figure 3.3: The transparency gradient showing the approximate positions of several instruments: violin (A), piano (B), scratch turntable (c), synthesiser keyboard in percussion mode (D), Iamascope (E), Very Nervous System (F), and laptop bands (G). Violin, Iamascope, Very Nervous System, and laptop bands are discussed in the text. Piano is shown slightly less transparent to the audience because parts of the sound production mechanism are hidden inside the body of the instrument. The scratch turntable is largely transparent but has modes that are hidden to the audience (choice of disk) and the player (exact position on disk). The hidden modes of the synthesiser keyboard, coupled with the non-intuitive mapping of keys, make it even more opaque than the turntable.

to new synthesis engine, is increasingly common. In all these cases, there is a gap in familiarity for both player and audience. Neither party knows what output to expect based on a given input. The player can improve on this situation by gaining physical proficiency, but this is difficult when the mapping is not clear.

The Very Nervous System (VNS) [55], a gestural controller, is an example of an opaque/opaque instrument. It uses Fourier analysis to determine the frequency components of video input, mapping these to musical parameters. The mapping is so complex, however, that it is extremely difficult for either the player or the audience to understand what is happening. Another opaque/opaque instrument is the Iamascope [16], in which unseen zones in front of a video camera correspond to strings on an imaginary instrument. Movement in a zone maps to plucking of its string, but the link is non-obvious. This, coupled with the fact that the zones themselves are impossible to perceive, makes Iamascope difficult to play expressively even for those who do understand its mapping.

There are two common ways to move a new technology out of the opaque/-opaque quadrant. The first is to make the instrument simple; the second is to add desirable functionality. These methods tend to move instruments in different directions, to opaque/transparent and transparent/opaque respectively. Simplifying an instrument tends to make it easier for the audience to understand, but doesn't necessarily make it easier to play. Often simplifications reduce the dynamic range of the output, lowering the expressive capacity. Adding functionality creates a motivation for early adopters [43] to learn the instrument but provides no explanation of the instrument's mapping to the audience.

The common problem that both of these methods share is that neither of them relates to existing literature. This displacement from a common reference point causes opacity for both player and audience. A new mapping, based on reference to the literature, would avoid such drawbacks. Metaphor can be used to relate new technology to the known, cultural basis of the literature. The literature may be from any culture, and metaphors from two or more literatures can be combined in a device. In the following section, we present metaphor as a way to increase the transparency for both the player and the audience.



		Spatial Multiplexing		
		Convergent	One-to-One	Divergent
Temporal Multiplexing	Non-Modal	Piano, Virtual Piano	Violin, BoSSA	2Hearts
	Modal	Synthesiser Keyboard	Electronic violin	Scratch turntable

Figure 3.4: The categories of instruments formed by the intersection of temporal and spatial multiplexing, and examples of instruments in each category.

3.3 Increasing Expressivity Using Metaphor

The application of a metaphor to an interface has the effect of increasing the transparency for both the player and the audience. However, depending on the mapping type, metaphor is effective through different mechanisms. Depending on whether the mapping is modal or non-modal, or is convergent, one-to-one, or divergent, six possible mapping types exist [7]. Modal mappings are those in which the input is multiplexed temporally. That is, depending on the active mode, a given input can produce one of multiple outputs. Convergent, one-to-one, or divergent mappings are based on the amount of spatial multiplexing — the degree to which groups of gestures are mapped to groups of simultaneous sounds. The six possible mapping combinations are shown in Figure 3.4 along with examples.

Modal mappings can benefit from metaphor as a way to obviate the instrument's current mode. Convergent, divergent, and one-to-one mappings can all use metaphor to explain their behaviour. The following sections discuss examples of convergent non-modal, convergent modal, and one-to-one non-modal mappings. Finally, metaphor is presented as a design tool.

3.3.1 Convergent Non-Modal Mappings

Convergent, non-modal mappings generalise groups of control gestures into common outputs. An example from the literature of musical instruments is the piano. Many finger positions activate the same key, sounding the same note. There are no internal modes[2], so the note played is the same each time the key is pressed. Metaphor can be used to cognitively group the control

[2]We can safely ignore the piano's pedals, which arguably don't change the meanings of the finger positions and therefore don't create internal modes.

gestures associated with one sound output. In the case of the piano, a range of finger positions is understood to activate a single key. This metaphor has been used in instruments that use a key model but don't have explicit keyboards, such as in the Virtual Piano created by Leonella Taraballa and Graziano Bertini at the CNUCE in Pisa in 1997. The Virtual Piano removes the keyboard entirely, relying on the familiar gestures of a pianist without the physical keys.

3.3.2 Convergent Modal Mappings

Modal mappings use internal modes to choose which sound output will result from each single gesture. For example, the synthesiser keyboard uses different modes to map convergent key presses to different outputs. Pressing the same key in the same way can, in different modes, produce the sound of a piano, a tuba, a raindrop, or any other arbitrary sound. In this case, the piano keyboard metaphor, which has pitch increasing to the right, can be maintained if the sounds produced contain a pitch element. However, the mode selection is arbitrary, hidden from the audience. Furthermore, it is often poorly indicated to the player, usually consisting of a set of buttons with some indicator light, or a menu system. This interface could be improved with the application of an appropriate metaphor defining and explaining the mode selection process. One rather simplistic solution would be to use a tangible interface [32] based on small figurines of actual instruments. These would be placed on the keyboard to indicate mode selection to the player and the audience. The obvious problem with this metaphor is that it requires the player to find the correct figurine in order to switch modes during a performance. This may be too time-consuming, especially in instruments with many tens or hundreds of possible modes.

3.3.3 One-to-One Non-Modal Mappings

One-to-one mappings exemplify a direct relationship between control and output. With a complex instrument, it can be difficult to remember what the relationship is. Metaphor can be used to provide a control framework for the mapping. This framework creates relationships to the individual control gestures. BoSSA, for example, bases its control gestures on those of the violin. Instead of directly affecting a vibrating string, the BoSSA player bows a set of force-sensing-resistor–based vanes, while fingering a pressuresensitive fingerboard on an attached neck. In this way he directly interprets the violin metaphor. BoSSA then builds on that base by allowing gestures not normally useful on the original instrument, such as changing the angle of the neck relative to the body of the instrument.

One interesting offshoot of this approach is the possibility of combined mapping types. The acoustic guitar, for example, is similar to a violin in its control gestures. However, it also incorporates components of a convergent mapping through the inclusion of frets. Frets allow many finger positions on the strings to be mapped to one string length, which produces a single sound output. The use of frets improves the transparency of the instrument by making it more apparent which finger positions will produce which notes. Novice violinists spend a long time learning the correct finger positions for each note, while frets ease this process for novice guitarists. This increase in transparency comes at the expense of expressivity[3]. Guitarists can no longer create glissandos, trills, or vibratos using the same gestures as violinists. However, guitarists have found ways to regain this expressivity that would not be possible without the frets. Pitch bends are accomplished on a guitar by sliding the string sideways on the fret, thereby stretching the string. Vibrato can also be achieved by varying finger pressure behind the fret, also stretching the string. Such gestures are not possible on a violin because they require frets, and because the cocked wrist position of a violinist doesn't provide a strong enough grip to affect the strings in these ways.

As an aside, one variation for the guitar, suggested by this comparison to the violin, would be to remove the frets after the player has learned the correct note positions. In this case, the frets would act as training wheels for the guitarist. Removed when no longer needed, the guitarist could then return to the more transparent one-to-one mapping of a fretless guitar. Indeed, there is a growing community devoted to the subculture of fretless guitar.

3.3.4 Metaphor as a Design Tool

We have seen that metaphor can be applied to new technologies in many ways in the previous sections and in [38] and [60]. Metaphor can also be used as a design tool when creating new instruments. If a new synthesis engine is implemented, it may suggest a metaphor that encompasses its main

[3]This demonstrates the idea that transparency is a necessary but not sufficient condition for expression. In this case, increasing transparency has decreased the dynamic range of the instrument, which decreases its expressivity.

characteristics. The metaphor may then dictate an appropriate controller for the device, so that the entire device is self-consistent.

One example given in Section 3.2, Iamascope, used the metaphor of plucking strings to determine how movement in front of a video camera would control the music. Movement in each of ten zones would pluck the string corresponding to that zone. While this is a reasonable metaphor, two factors mitigate its usefulness in creating an expressive instrument. First, the computer automatically chose chords on the ten strings. The user could choose which string to pluck, but the note corresponding to that string might have changed due to an internal mode switch. This factor alone reduced the controllability and learnability of the interface, creating an opaque mapping. Second, the metaphor depends on the user recognizing the ten zones and being able to consistently select between them. Since the zones were unmarked and were in an external frame of reference, the gestures required to trigger them were very difficult to reproduce. One lesson learned from Iamascope is that metaphor alone is not sufficient: the metaphor must be represented in the interface for it to be meaningful to the user.

In the next chapter we will examine how MetaMuse is designed around metaphors appropriate for granular synthesis. The discrete event-based nature of granular synthesis suggested the rainfall metaphor used in the device, which then indicated a watering can as an appropriate controller. In MetaMuse, the synthesiser suggested the metaphor, which suggested the controller. This design strategy can also be reversed: a new controller may suggest a metaphor, which may then dictate an appropriate synthesis engine for the device. Finally, an instrument can be based on an original metaphor, from which both the input and the output interfaces are drawn. By applying these design strategies to the mapping types discussed above, metaphor can lead to more transparent instrument mappings, which in turn create expressive devices.

CHAPTER 4

THE METAMUSE SYSTEM

From our discussion in the previous chapter, we predict that metaphor is a valuable tool for both designing and playing expressive instruments. We built MetaMuse, a metaphor-based instrument, to explore this theory. In this chapter we will see how MetaMuse centres on specific metaphors which guided critical decisions during the system's design. In the next chapter we see how MetaMuse was evaluated as an expressive device for players.

MetaMuse is a rain-sound synthesis device based on metaphors of pouring water, falling rain, and changing landscape. The device, shown in Figure 4.1, uses two props, a watering can and a flat landscape, to control a real-time granular synthesiser. The three system metaphors provided the central theme for MetaMuse's design. Originally chosen to match the synthesis engine used, the metaphors guided the design of the prop paradigm used for the controllers and the mapping from control gesture to synthesis parameters. Ultimately the metaphor-based design came full circle, prompting significant modifications to the synthesis engine itself to provide a better cognitive fit.

MetaMuse is rooted in its metaphors, so elucidating them is important to explaining the system. Once we examine the metaphors we can discuss their impact on the system design, from the choice of props to the mapping and synthesis model. We begin with an examination of the metaphors.

4.1 The use of Metaphor in the MetaMuse System

The MetaMuse system is centred around three metaphors: pouring water, falling rain, and a virtual landscape. The pouring metaphor encompasses the act of pouring and the control implied therein. The concept of falling rain includes the fall of water particles and their striking a surface. The virtual landscape metaphor implies the ability to change the surface on which the rain falls, and therefore the resultant sound. The ensuing discussion of these

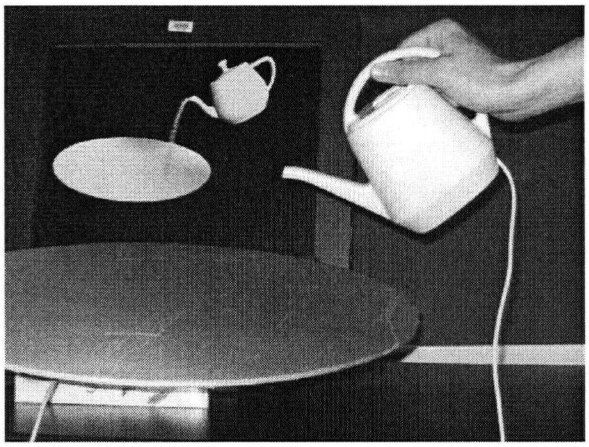


Figure 4.1: MetaMuse. In the foreground, we see the watering can and landscape props used to control a rain sound synthesiser. In the background the props are rendered on the screen along with the virtual water that flows between them.

metaphors includes forward references to the props used; these props are discussed in greater detail in coming sections.

4.1.1 Pouring Metaphor

The pouring metaphor is based on the act of pouring water from a vessel. More specifically, in MetaMuse it is based on the watering can found in many gardener's sheds. One example of such a watering can is depicted in Figure 4.2.

The metaphor comes from the literature of gardening tools, but it also draws from other, more general experience. Any literature that involves pouring water from a vessel, including such every-day activities as serving drinks at dinner, can be invoked to explain the pouring used in MetaMuse. The rose is similarly generalised, as most people who might use MetaMuse have experience either with showers, and therefore shower heads, or with similar attachments for garden hoses.

The metaphor implies that the watering can, which embodies the metaphor, will conform to specific behaviours. Tilting the can forward produces a flow of water from the spout, which is broken into individual droplets by

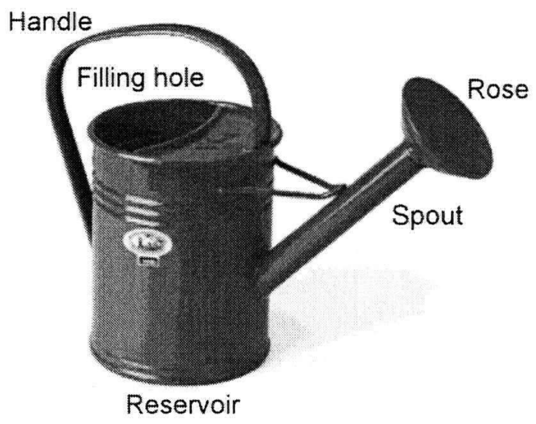


Figure 4.2: A typical watering can. An open hole is used to fill the reservoir from the top, and a spout emerges upwards from near the base of the reservoir. The rose at the end of the spout causes the stream of water to break into individual droplets.

the rose. The farther the can is tilted, the more water flows from the spout. Tilting in other directions reduces the flow of water or, in the case of tilting about the axis of the spout, has no effect.

Our interpretation of this metaphor has two noticeable inaccuracies, and one less-noticeable one. The greater pair are that the water level never changes, and that water cannot be poured from openings other than the spout. Having an unchanging water level is by design, in that it is desirable to have a continuing flow of water for any length of time without the user having to stop to "refill" the can. This departure from the expected behaviour of a watering can is acceptable because it is easily understood. In all but the smallest watering cans, the reservoir of water is large enough that the change in water level during the first few moments of pouring has little effect on the water flow. The duration is long enough that we can consider the literature to include a constant-pressure pour from a reservoir. The pouring metaphor refers to that aspect of the literature only, and user testing, discussed in Chapter 5, suggests that the distinction is accepted.

The second inaccuracy in our interpretation involves tilt angles that are not used during normal pouring. For example, a real watering can has a hole at the top for filling; if the can were inverted the water would pour out of the hole in a stream. Similarly, most watering cans have spouts that emerge from the base of the can as in Figure 4.2. Tilting such a can greater than 90°actually causes a reduction in the spout water pressure. Neither of these behaviours is modeled by the system, but they are not required for its operation. They would, however, greatly complicate its implementation, so we have left these behaviours for future work (see Section 6.5). As we shall see in Chapter 5, many users tried to test this aspect of the metaphor, apparently not because this behaviour was expected but because they were curious to see if it had been implemented. In some cases, users seemed to by trying to "break" the system by using it in a way they thought was unexpected. Users generally had no issue with the restriction on this behaviour once they'd determined it wasn't possible in our system.

The more minor inaccuracy in our interpretation of the metaphor has to do with how the flow of water from the watering can's spout is calculated. The correct calculation of the amount of water flowing from the spout would take into account the shape of the reservoir at the current angle of tilt to determine the water pressure in the spout, then consider the cross-section of the spout to calculate the resultant flow of water. Similarly, calculating the correct velocity of the water from the spout should take into account the flow, the spout's orientation, and the velocity of the watering can itself. Instead, the amount of water produced at the spout, and its initial velocity, are calculated simply based on the downward component of the spout. This simplification means that certain interactions, such as sloshing the water to vary the throw from the spout, are not possible. Unlike the inaccuracies noted above, this issue was not commented on by our users. It did not appear to affect the operation of the device.

4.1.2 Rainfall Metaphor

The rainfall metaphor is based on a water droplet falling from some height and hitting some surface. When the droplet hits the surface a splash is heard, its properties depending on the velocity of the droplet. The splash itself, evoking a certain surface composition, is determined according to the landscape metaphor described in the following section. The rainfall metaphor specifically refers to the way in which the water droplets fall separately from one another, the velocity of their impact, and the fact that they splash on striking the landscape. It is the cumulation of many splashes that makes the sound of rainfall.

The literature for the rainfall metaphor is a combination of rainfall itself and other water sounds. Most people have experienced the sound of rainfall and know what to expect when many water droplets fall onto a surface. Variations of this effect are common to many literatures, including the shower present in most North American bathrooms and the sprinklers on many North American lawns. It is the group of all of these processes that provides the expectation of droplets' behaviours and their sounds given the surface type.

Being based on a group of processes rather than one process specifically, the rainfall metaphor is open to interpretation in some ways. Two ways in which the metaphor used in MetaMuse differ from real-world rainfall are terminal velocity and distance effects.

When rain falls from the sky, it falls far enough to reach terminal velocity. In ideal conditions, rain will always hit the ground with the same velocity. Other sources of water are closer to the ground and can also vary in velocity. Our system operates by the latter model, allowing the user to vary the strength of the droplets hitting the surface. Due to wind effects, rain falling from the sky doesn't fall with constant velocity, and can hit harder or softer as the wind gusts. By allowing the user to control the droplets' velocity, wind-related rainfall effects can be created.

The sound of rain is a cumulative sound, with many raindrops' splashes heard together. In particular, some raindrops hit the ground closer to the observer, while some hit farther away. This results in a variation in volume over the accumulation, shown in Figure 4.3, that is not modeled in the system. Section 4.3.3 discusses the limitations imposed on the system due to finite computing resources, and the methods by which these limitations were mitigated.

As with the watering can metaphor, there is an inaccuracy in our interpretation of the rainfall metaphor: no runoff, pooling, or other surface effects are modeled. For the same reason as the constant water level in the watering can metaphor, the effect is not required to provide consistency with the metaphor. Specifically, rainfall has an initial stage, before enough water has fallen to form pools, in which the droplets land on a dry surface. It is reasonable to draw the metaphor from that period alone.

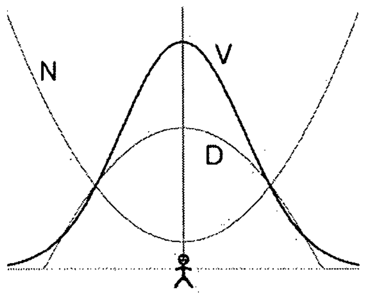


Figure 4.3: Droplets at different distances contribute different amounts to the total sound. The number of droplets, N, increases exponentially with distance away from the listener in the centre. Meanwhile, the volume of the individual droplets, D, attenuates to zero. The total contributed volume with distance is estimated as curve V.

4.1.3 Landscape Metaphor

The landscape metaphor implies the ability to vary the surface on which the droplets land. Whereas the rainfall metaphor, described previously, indicates that there will be a splash heard when the droplet hits, the landscape metaphor describes the possibility of different surface compositions and therefore different splashing sounds. The metaphor is based on landscape maps, where a large area of land is represented on a smaller scale. However, its representation is abstracted to avoid giving incorrect preconceptions.

The literature for different surfaces producing different sounds is as varied as the literature for the rainfall metaphor. The primary source for this metaphor is, of course, rainfall: people's experiences with rainfall under different conditions provide an expectation for different sounds depending on surface composition. One such experience may be a person's walking from a grassy area to a paved area while it is raining. The person will experience a change in the sound of the rain as he or she moves. While this type of experience necessitates a certain time separation between the perception of the different sounds, it generalises to quickly-changing sounds. Indeed, other water experiences will provide that experience directly; spraying a garden hose on various surfaces is one. Other water sounds, such as running a shower in different shower stalls at different times or sitting in a car as it moves through an automatic car wash, provide additional background from which a user of MetaMuse can draw.

To evoke the varying landscape metaphor, the surface of the palette prop represents a landscape, with areas of trees, grass, rock, and pond, as well as transitions between these areas. However, the depiction of this landscape is very abstract, as seen on the landscape in Figure 4.5. Though the sound of rain is a common one that many people have heard, depending on a user's background she may expect rain on a certain surface type to sound in a certain way. MetaMuse intentionally avoids providing a grounding point for these expectations by its lack of specific landscape features. For example, a person who grew up near a forest may expect rain over trees to sound a particular way depending on the makeup of that forest. Since the palette has an abstract pattern of colour rather than being specifically illustrated, the user will have more general expectations – green may represent foliage, but no particular foliage is implied.

4.2 Props

MetaMuse is controlled by manipulating two hand-held props. As the input interface to the system, the props define the range of valid control gestures available to the user. They are also designed to embody the metaphors present in the system so that they give the user appropriate expectations regarding the use of the system. The two props used in the system are the watering can and the landscape, shown in Figures 4.4 and 4.5.

Recall in Section 3.1 we defined the input interface of a device mapping as the set of control gestures used to control the device. The input interface of MetaMuse consists of gestures that result in water pouring from watering can and impacting the surface some way. By using a watering can prop to embody the pouring metaphor, and an abstract landscape to embody the landscape metaphor, the input interface is clearly implied. In Norman's [44] framework, the watering can *affords* pouring from, and the landscape *affords* pouring onto.

The following subsections discuss the props, their affordances, and their embodiment of the three metaphors discussed in the previous section.

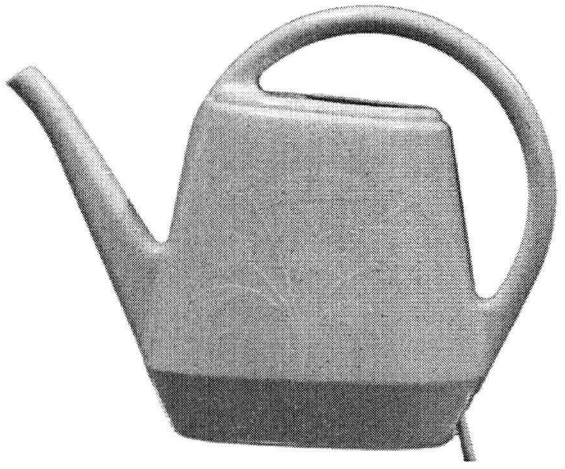


Figure 4.4: The watering can prop. It is plastic, about 20 cm high.

4.2.1 Watering Can

The watering can prop is composed of a small plastic watering can, shown in Figure 4.4. The can is small enough to use comfortably with one hand - about 20 cm high.

The can's handle and spout afford the act of pouring. It is natural for a new user to try tilting the can in a pouring motion to see what happens. The part of the input interface embodied by the watering can matches this affordance well, as control with this prop consists entirely of normal pouring actions. As such, the watering can embodies the pouring metaphor.

Note, however, that the watering can prop does not include the physical representation of the rose discussed in Section 4.1.1. User reactions during the first pilot experiment, discussed in Section 5.4.1, indicated that it was not necessary — users had no expectation of a single stream of water as the spout might imply. Since it was not a concern, no rose was added to the prop.

4.2.2 Landscape

The landscape prop is composed of a large wooden circle with a handle glued to the bottom and a pattern overlaid on paper, as shown in Figure 4.5. An

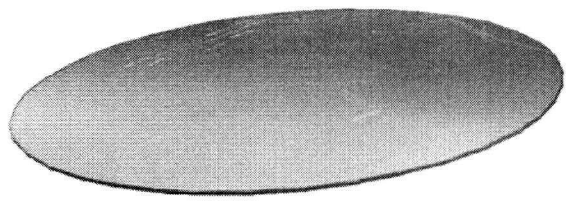


Figure 4.5: The landscape prop. It is made of thin wood, covered in paper printed in a colour gradient. Nearly circular, it is about 70 cm across.

earlier version of the landscape was smaller, based on a painter's palette with a hole for grasping. In Section 5.4.1 we discuss why it was discarded in favour of a larger palette.

The purpose of the landscape prop is not as obvious as that of the watering can. It certainly affords holding, presenting itself as something to be positioned horizontally, but its use could be confused. However, placed in context with the watering can its use becomes clear — it is the obvious sink to the watering can's source.

The abstract colour pattern on the landscape presents a placeholder for the user, indicating that position on the landscape is important. As such, it affords varied droplet placement, thereby embodying the landscape metaphor. As discussed in Section 4.1.3, the colour pattern was purposefully left abstract to avoid false expectation.

The final metaphor used in MetaMuse is that of rainfall. This metaphor is not embodied in the props because it describes the behaviour of the virtual water that mediates their interaction. Just as the purpose of the landscape becomes clear with experimentation, however, so does the use of the rainfall metaphor. For both of these issues new users are aided by the visualisation of the system, shown in Figure 4.6.

The watering can, landscape, and their interactions together define the input interface. The interpretation of these actions takes place in the mapping.

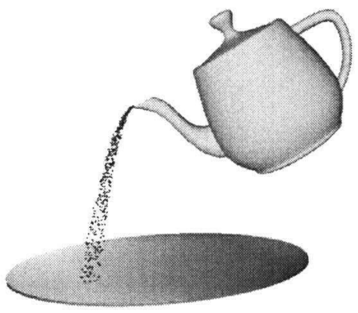


Figure 4.6: The graphics scene, rendered in OpenGL. The watering can and landscape are represented directly, while the virtual water seen flowing between them has no physical analogue.

4.3 Mapping

As described in Section 3.1, the mapping defines the interpretation of the user's input gestures and their conversion to sound output. In MetaMuse, the device mapping consists of the physical model, which mediates the pouring of virtual water between the props; the synthesiser, which converts the virtual droplets to sound output; and the link between these components.

4.3.1 Physical Model

The physical model provides the mechanism for the interaction between the two props, partially defining the mapping from input interface to output interface. It consists primarily of a particle model that creates particles at the spout of the virtual watering can, updates their positions to fall freely under simulated gravity, and tests for collisions with the virtual landscape. It also includes a Tcl/Tk interface to manipulate configurable parameters so that they match the physical props' sizes and locations to the virtual model, as well as OpenGL code to render the virtual model on the screen.

The particle model consists of the prop positions and a set of droplets, called "sprinks"[1]. The prop positions are updated by Polhemus magnetic

[1]So called because they are sprinkled.

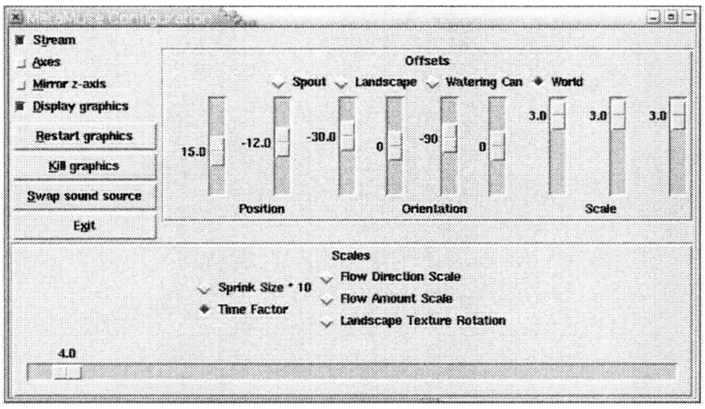


Figure 4.7: The configuration interface for MetaMuse. The bank of vertical sliders allows the model to be positioned relative to the Polhemus. The horizontal slider along the bottom adjusts model parameters such as a flow gain control and rendered droplet size. The controls on the top left perform system tasks such as turning Polhemus streaming on and off.

sensors, and new sprinks added to the system based on the position and tilt of the watering can. New sprinks have an initial position and velocity, with a small random velocity added to ensure a slight spread of flow when many droplets fall. Droplets are updated based on simulated gravity. As each droplet is updated, it is tested against the current landscape position. Droplets that intersect the landscape initiate synthesiser events based on their position and velocity relative to the landscape. The consumption of these events by the synthesiser is discussed in Section 4.3.4.

The other two components of the physical model are the Tcl/Tk configuration interface and the OpenGL graphical visualisation. The configuration interface, shown in Figure 4.7, provides access to Polhemus calibration and physical model tuning. Adjustments with this interface were made during development, but the interface was not exposed to the test subjects so it will not be discussed further.

The graphics in MetaMuse are not complex, as rendering is not the focus of this thesis. Basic OpenGL commands are used to create a scene with the three required elements: watering can, landscape, and sprinks. The full scene is shown in Figure 4.6. Users were encouraged to view the scene to help calibrate themselves to such parameters as amount of flow and exact placement of sprinks of the landscape. However, we believe that an experienced user would not require visual feedback to competently manipulate the interface. This belief is not tested, and the graphics are not discussed further.

As discussed above, each time a sprink intersects the landscape a droplet event is sent to the synthesiser. The event consists of six variables: position and velocity in three dimensions, relative to the landscape. Before discussing how these variables are converted to synthesiser parameters we will describe the synthesis engine itself.

4.3.2 Sound Synthesis

The MetaMuse synthesis engine uses a combination of granular synthesis and real-time synthesis. As described in the introduction to this chapter, granular synthesis provided the genesis of the system metaphors, which then prompted the modification of the synthesis engine. In granular synthesis, short sound samples are extracted from a sound source and played one over another, blending to create an overall effect. Normally, one or more longer pre-recorded sound samples are used as the sound source, so that the samples are selected discretely. MetaMuse is a continuous instrument, so the sound source was replaced with a real-time synthesiser that allowed continuous parametric control over the sound produced. The sound output of the real-time synthesiser was then blended and layered in the manner of a granular synthesiser, creating a new type of synthesis with aspects of both. We call this new synthesis technique "parametric granular synthesis".

Granular Synthesis

Granular synthesis, as described by Truax and summarised in Section 2.4.1, blends and overlaps many short-duration sound samples to create a gestalt sound different from the original samples. These short samples are selected from one or more lengthier prerecorded sounds. Because of this underlying selection of existing sounds, granular synthesis has an inherent discreteness to it: to change the underlying sound, a new sound file must be loaded.

While the underlying sound sample can only be changed discretely, the MetaMuse controller provides a continuously variable input. Therefore the underlying sound source was replaced while the overall structure of granular synthesis was maintained. Instead of using a pre-recorded sample as the sound source, we used the output of a parametric real-time synthesiser that produced short-duration sounds, then blended its output using the layering effect for which granular synthesis is known.

Real-Time Synthesis

The real-time synthesiser used for MetaMuse is based on the Synthesis Toolkit (STK) [56]. STK is a set of audio synthesis classes written in C++ that includes a stochastic event model called Shakers. By modifying the Shakers class, a water droplet "Instrument" was built. Several such Instruments were used in parallel to synthesise the sound of rain. The Shakers class, its modifications, a heuristic filter, and the mapping from physical model parameters to synthesiser parameters are discussed in this section.

The Shakers class provides PhISEM, the Physically Informed Stochastic Event Model, which simulates collisions of multiple sound-producing objects. Effectively, a real-time stochastic excitation process provides input to a bank of resonant filters. By varying such parameters as the initial excitation energy, reverberation, decay, and gains at various points in the system, and filter characteristics such as centre frequency and resonance, a variety of instruments can be simulated. These include maracas, bamboo wind chimes, and tambourine. The PhISEM system is summarised in Figure 4.8.

Several changes were made to the Shakers class to provide a water droplet synthesiser. Foremost of these was the replacement of the stochastic processes with pseudo-random signals. In Shakers, exciting the model with identical parameters can result in varying sounds, depending on the random signal generated at runtime. This results in an inconsistent sound, inappropriate for an instrument where repeatability is a desirable quality. Therefore, the random function was pre-generated and saved, to be used as the "random" input every time the instrument is excited.

Similarly, the excitation function was a random process in the Shakers class. It was replaced with a recorded excitation function obtained from recording a water droplet landing on a non-resonant surface. This provides the basic excitation for our water droplet synthesiser. Other changes to the Shakers class were for programming purposes and did not affect the sound. All changes are discussed in greater detail in Appendix B.

The result of the modifications is an Instrument class that uses 26 parameters plus activation energy to define the output sound. Different regions in that sound space are used to synthesise different instruments, with one

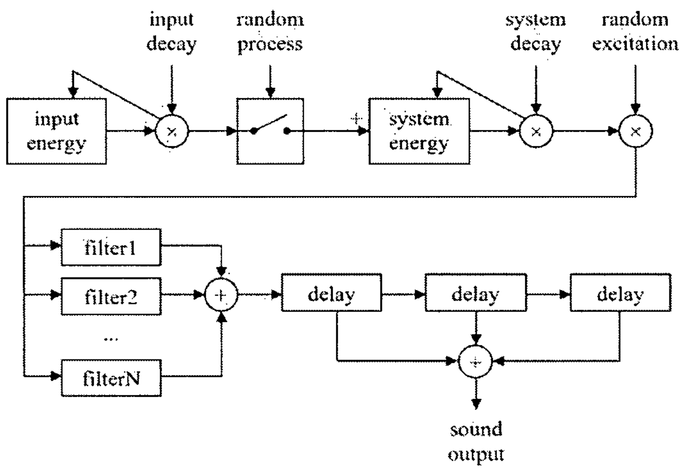


Figure 4.8: The sound generation process. The input energy is set to an initial value when a droplet is initiated, and decays with each iteration. Based on a random process, the input energy is sometimes added to the system energy, which decays similarly. A random excitation process perturbs the system energy before it is input to the resonant filter bank. Finally, the filter outputs are fed into a delay line and three delays are summed to form the sound output. For MetaMuse, the random processes were replaced with pre-recorded sequences to ensure consistency across droplets.

region producing water droplet sounds. Several such Instruments are used in parallel to allow multiple concurrent droplets to sound. Each time a droplet event is received, the next Instrument is set up with the appropriate parameters and excited. The sum of all Instruments' output produces the rain sound output.

4.3.3 Heuristics in the Synthesis Engine

The above synthesis method creates an Instrument that produces a convincing range of single water-droplet sounds. However, when many such sounds are combined to create rainfall sounds, the synthesiser sounds "flat". There is a noticeable difference between the sound of rain and the sound of the synthesiser, the most prominent effect being a lack of bass.

One of the main causes of this difference is the limited processing power of the hardware used for the synthesiser. Rainfall inherently consists of many thousands of droplets landing nearly simultaneously in audible range. Our PC hardware was only able to synthesise ten to fifteen concurrent instruments before the sound degraded with the processor running at full capacity. To compensate for this shortcoming, heuristics are employed to give the illusion of more rain.

Three heuristics were introduced: bass boost, low-pass filtering, and gain control. These are shown in Figure 4.9. All three are based on the number of currently-active Instruments, or the instantaneous frequency. As the number of concurrent droplets increases, the gain is increased on the bass filters of each Instrument, filling in the sound. A low-pass filter is employed as more droplets become active, slurring the sounds together to make them less distinct. Finally, the overall system gain is increased, raising the volume of the system. These three heuristics greatly improve the sound of the synthesiser.

4.3.4 Converting Sprinks to Droplets

With the synthesiser creating such a broad sound space, the mapping from the physical model's sprinks to the synthesiser's droplets is very important. It must restrict the synthesiser's parameter space to a meaningful subset of the available sounds, yet provide enough freedom to allow a fairly broad range of rain sounds.

The large sound space of the synthesiser was reduced to a controllable region by restricting navigation to a plane in that space. Specifically, three pa-

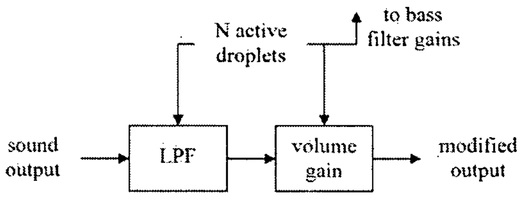


Figure 4.9: Three heuristics were employed to improve the quality of the synthesiser's output. All three were based on the number of currently-sounding droplets. As the number of droplets increased, the gain on two low-frequency filters in the filter bank of Figure 4.8 was increased; the amount of low-pass filtering on the sound output was increased; and the overall synthesiser volume was increased. These heuristics helped simulate the sound of a greater number of droplets.

rameter sets producing very different-sounding droplets were defined. These are virtually "positioned" on the rim of the landscape, equidistant from one another. The three droplets then define a plane in parameter space that covers the landscape, and all sprinks' positions are mapped to synthesiser parameters by interpolation.

This method results in a sound space that does not include all water-droplet sounds. However, it does provide a wide range of such sounds, and more importantly it excludes all sounds that are too far removed from water-droplet sounds to be used.

This chapter provided an overview of the MetaMuse system. Appendices B and C provide further details on the sound synthesiser and the Polhemus libraries respectively. In the following chapter we discuss the evaluation of the system.

CHAPTER 5

SYSTEM EVALUATION AND USER TESTING

Having used metaphor to aid in the design of the MetaMuse system, we performed a series of user experiments to evaluate its expressivity. In Chapter 3, we posited that metaphor provides guidance in both designing and playing a novel device. By the mechanism of metaphor, the device's mapping is made more transparent, increasing the expressivity of the device. In this chapter we will describe how MetaMuse was tested and the effectiveness this approach verified.

Since metaphor was an integral part of the design process and is imbued in the system, we expect that it will aid users in playing MetaMuse, and that the device will be more expressive than similar devices not using metaphor. Our approach to testing MetaMuse, therefore, is a comparative one. Three typical graphical user interfaces (GUIs) were built, using increasing levels of abstraction to control the same synthesiser. We evaluated MetaMuse and these GUIs in a series of experiments designed to measure the controllability and expressivity of the different interfaces.

Three pilot experiments were run to validate and refine our experimental process. The final experiment was a formal user test consisting of a sound target matching exercise and a questionnaire. Our hypotheses were:

1. MetaMuse is more expressive than a typical GUI controller.
2. MetaMuse is more controllable than a typical GUI controller.

The results confirm that metaphor is a valuable tool for designing and understanding novel devices. In particular, users found MetaMuse to be more creative than the non-metaphoric interfaces, confirming the expressivity hypothesis. The experiments also found some support for the second hypothesis. Our results are summarised in the following section, which provides an overview of the experiments run. The remainder of the chapter will describe the experiments in greater detail.

5.1 Overview

Four experiments were run to evaluate MetaMuse. These were a pilot experiment, a just-noticeable difference (JND) experiment, a second pilot experiment, and the final, formal user test. These experiments, and the lessons learned from them, will be introduced in this section. First, here is a summary of the experiments.

1. Pilot I

   Purpose Validate use of comparative target-matching task with typical GUIs for measuring controllability.

   Results Validated basic approach. Exposed some system reliability issues. Prompted adjustment of experimental process to reduce learning effects. Indicated need for refinement of target distance metric. Obviated significant controllability deficiency in one GUI, motivating its removal from further experiments.

2. Just-noticeable difference experiment

   Purpose Determine an appropriate metric for measuring subjects' accuracy in the target-matching task.

   Results Indicated that the number of significant variables in the accuracy calculation could be reduced to two while increasing the perceived accuracy of the metric.

3. Pilot II

   Purpose Validate the systemic and procedural changes made since the first pilot. Verify the questionnaire.

   Results System and procedure verified. Fine-tuning of questionnaire indicated.

4. Formal Experiment

   Purpose Compare the controllability and expressivity of MetaMuse to that of the typical GUIs, per the two hypotheses introduced in the previous section.

Results Users consider MetaMuse more "creative" than the GUIs, a result that is statistically significant. MetaMuse tends to be better in other aspects of expression and in target acquisition speed, but without significance. We can therefore claim support for the expressivity hypothesis, but not for the controllability hypothesis.

The experiments were based on comparisons between MetaMuse and two or three traditional GUIs. The GUIs allowed parameter-level control over synthesiser parameters, primarily using sliders. They are referred to herein as "parametric interfaces", and described in more detail in later sections.

The pilot experiment involved subjects listening to a target sound, then matching that sound by controlling the synthesiser with either the MetaMuse interface or one of the parametric interfaces. The test was run semi-formally, with the expectation that refinement would be required before the system was ready for formal experiments.

The primary result of the pilot was that it showed that the target-matching approach is a valid way to test controllability. Other revelations led to several modifications to the testing procedure and infrastructure. Most important of these was the identification of a number of reliability issues exposed during the pilot, which resulted in some components of the evaluation system being re-implemented more robustly. Also, it became obvious that one of the parametric interfaces offered significantly less control than MetaMuse or the other two GUIs. This interface was dropped from further testing.

Another result of the pilot test was the realization that the accuracy metric was poorly defined. The pilot had been using Euclidean distance in synthesiser parameter-space to determine subjects' accuracy in acquiring the target. However, it became apparent from subject feedback that this metric was inconsistent. This problem motivated the second experiment, which tested the just-noticeable difference for the various synthesiser parameters.

The just-noticeable difference for a parameter is the minimum change in that parameter's value that produces a perceptible difference in the sound. By determining which parameters have the greatest impact on the perceived sound, the metric can be tuned to measure the subjects' perceived accuracy. Testing JND involved constructing a new interface which allowed incremental changes of individual synthesiser parameters. Subjects were asked to listen to two sounds and indicate whether they perceived a difference between them. By changing a parameter varying amounts from various base values and repeating for all parameters, a map of auditory sensitivity against parameter was made.

The most significant result of the JND experiment was that certain aspects of the sound had surprisingly low significance when varied alone. The primary one of these was the number of droplets per second: varying this parameter by large amounts produced little difference to the untrained ears of the JND experiment's subjects. Other parameters were found to vary together at about the same rate. This, coupled with the correlation between some parameters due to the three-droplet landscape layout discussed in Section 4.3.4, allowed the number of parameters used in the distance calculation to be reduced to two.

With the performance metric and other issues resolved, the system was ready for further user testing. At this stage, the timeline and required resources for additional testing were examined. Based on limited available resources, it was decided that the next experiment should test expressivity in addition to controllability. A set of questionnaires was added to the target-matching test to gain insight into the expressive nature of the system.

A second pilot was run to ensure that the modified experimental process was valid and all systemic issues were resolved. This pilot was run by several subjects who had done the first pilot, and verified the experimental process to be used in the formal user test. A number of minor issues were identified and some refinements were made. The questionnaire was modified slightly to ensure the appropriate information would be obtained, and the layout of the testing area finalised.

The formal experiment run next was the final experiment of the series. Twelve subjects were run through the procedure; technical problems resulted in one of those subjects being dropped from the final analysis. The data were then analysed to evaluate the controllability and expressivity of MetaMuse compared to the other interfaces. We hypothesised that MetaMuse is more expressive than the parametric interfaces, and that it is more controllable than the parametric interface. We found significant support for the expressivity hypothesis. While trends were evident supporting the controllability hypothesis, these results were not significant.

The expressivity hypothesis was supported by the experimental results. After using each interface, subjects were asked to rate their agreement with several statements on a scale of 1 ("Strongly Disagree") to 6 ("Strongly Agree"). For the statement "I felt creative while making rain sounds with this interface", subjects' responses averaged 5.22 for MetaMuse, compared to

3.78 for the highest-ranked parametric interface. This represents an increase of 38.2%, and was a statistically significant result.

The controllability hypothesis was loosely supported by the experimental results. Using MetaMuse, subjects acquired their targets about 7% and 12% faster than with the two parametric interfaces. Subjects' accuracy performance was about the same on all interfaces — within 2%. However, statistical analysis of variance indicated that these results were not significant.

The four experiments and their results will be discussed in greater detail in the remainder of this chapter. The next section will lay out the general approach taken to evaluating MetaMuse, motivating the following discussion of the experiments themselves.

5.2 Approach to Evaluating MetaMuse

Our goal in evaluating MetaMuse was to determine how well it allows expressive playing. This indicates the success of the system's metaphors, and by extension the metaphor-based design process. Recall from Chapter 3 our definition of expression as the act of communicating meaning or feeling. Our approach to evaluating MetaMuse, then, was centred around determining if the instrument enables a player to communicate meaning or feeling.

There are two components to evaluating the expressivity of MetaMuse. The first is to test whether users can control the instrument – without basic control over the sounds produced, expression is impossible. Proving basic control also ties in to the transparency of the mapping, indicating that users were able to grasp the connection from their input actions to the system's output sounds. The second component is to measure expressivity itself; as we shall see in the following sections this is a more difficult task. Both of these components are evaluated by comparison to some "standard" user interfaces.

5.2.1 Evaluation by Comparison

It is common in statistical studies to measure performance against a set of control conditions. With MetaMuse, no control conditions existed — there were no existing systems against which to compare performance. However, it is difficult to show meaningful results without such a comparison. We therefore created the control condition in the form of a set of graphical user interfaces that control the same synthesiser.

The traditional interface for computer programs is a point-and-click interface, also called a graphical user interface or GUI. Such an interface would act well as the control condition, as it is the standard way in which people normally control computers. In the case of MetaMuse, we created three such interfaces, with decreasing levels of abstraction. These interfaces sequentially remove the props that embody the system metaphors, then the metaphors themselves, then all amalgamation of parameters. They create a set of baselines against which to compare MetaMuse when evaluating both controllability and expressivity. These parametric interfaces will be discussed in greater detail in Section 5.3.

5.2.2 Measuring Controllability

We measured controllability using a sound matching task with pre-recorded target. This approach allows performance to be measured in a predictable way, and ensures that all users are performing comparable tasks. Other experimental approaches, such as composition tasks or free-form improvisation, allow subjects more freedom and are therefore appropriate to measuring expression. However, such tasks are much more difficult to measure, both within each subject and across subjects. For measuring controllability a more controlled task is preferable.

The primary factors considered for a target-matching task are target selection, target presentation, and performance measurement. We expound on these factors in the following sections.

Target Selection

Targets for the sound-matching task were generated by recording MetaMuse in a variety of static positions. A selection of watering can and landscape positions was used, and a 30 second recording was made in each position. Care was taken to ensue the entire spectrum of possible sounds was covered. In all, 30 samples were recorded, and these were used as targets for all the controllers. In the formal experiment, a subset of fifteen targets was used, as described in Section 5.4.

Target Presentation

The presentation of a sonic target requires separation of the target sound from the controlled sound. There are several possible approaches to creating this separation, based on two concepts: temporal separation or spatial separation. Targets can be temporally separated by presenting them at the beginning of the task only, or throughout the task with a toggle switch to alternate presentation of the target sound and the controlled sound. Targets can be spatially separated by presenting them in separate channels of, for example, stereo headphones.

We consider the latter approach to confound the matching task with the subject's ability to separate two sounds in his head, while the first approach confounds the task with the subject's memory abilities. Therefore, we chose to present the target sound alternately with the controlled sound, providing a toggle switch so that the user can select which sound is presented at any time.

Performance Measurement

The final consideration for measuring controllability in a target-matching task is actually taking the measurement. This is also the least clear-cut factor in this part of the evaluation, as there are a number of ways to do it. Several metrics are possible, including parametric distance from target, length of time to acquire target, self-assessed perceptual distance either after each trial or on review at the end of the test, self-assessed satisfaction, or perceptual distance as judged by a third party after the experiment. The measurement is further clouded by the question of when the subject has acquired the target. Target acquisition can be recognised when the subject comes within a certain parametric distance of the target, after a specified length of time, or when the subject explicitly indicates they have acquired the target.

In order to make the measurements as consistent as possible, we eliminated metrics that require personal judgement, either on the part of the subject herself or on the part of a third party. Instead, we measured those parameters that were objective: parametric distance from target and length of time to acquire target. In order to allow both to be measured, we required subjects to explicitly indicate when they had acquired each target.

The distance metric is based on a subset of the synthesiser parameters.

These parameters provide a weighted contribution to the total distance, which is measured in Euclidean coordinates. The time metric was determined by the subjects' indication of target acquisition. The remainder of this section explains the distance metric in more detail.

For the first pilot experiment, all parameters contributed equally to the Euclidean distance calculation. The results of that pilot, discussed in Section 5.4.1, motivated the JND experiment, detailed in Section 5.4.2. Based on that experiment's results, the second pilot and the final experiment used only two of the synthesiser parameters to calculate distance from the target.

The use of Euclidean coordinates for the distance metric was questionable for the first pilot. The many parameters were likely correlated in perception-space, so the independent-axes assumption of Euclidean geometry was not met. The pilot results and JND trials helped to simplify the calculation to use only two parameters. These were more clearly independent, so we felt the Euclidean distance was a reasonable metric for the final experiment.

5.2.3 Measuring Expressivity

We measured expressivity by soliciting user responses to a questionnaire between tasks. Questionnaires have the benefit of being relatively easy to administer compared to other techniques. The questionnaire was employed as a supplement to other experiments.

We considered a number of ways to measure expressivity, such as using a target-matching experiment, where the target is perceived emotive content, or using a questionnaire to glean subjects' emotional reactions to the system. These options are explored in some detail in this section.

Matching against an emotive target has similar concerns to matching a sonic target, described above: target selection, target presentation, and measurement. Possible emotive targets are photographs or pictographs with high emotive content, some examples of which are shown in Figure 5.1. Such pictures have been made available by some parties as standardised sets of emotion-evoking images. Other possible targets include video clips, abstract shapes or patterns such as those used in Rorschach tests, and, of course, sounds. Matching to a video has the additional benefit of allowing subjects to explore MetaMuse's dynamic capabilities, while complicating the measurement of their success. Matching emotional content of sonic targets has the additional complication of the sound itself, rather than its emotional content, influencing subjects' output. Presenting any of the above targets depends,

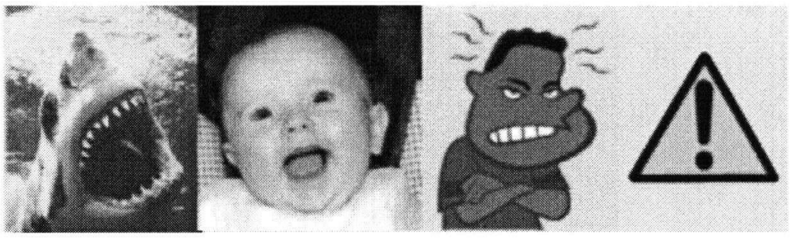


Figure 5.1: Possible emotive targets for an expressivity test. Images range from photographic on the left to abstract on the right. The leftmost pair of images evoke more visceral responses, while the rightmost image requires symbolic interpretation.

of course, on the particular type of target chosen, though visual targets do not present the channel interference problems with respect to sonic targets discussed in Section 5.2.2.

Measuring a subject's performance when matching an emotive target is non-trivial. A good first step is to classify the targets themselves by having them ranked in advance by an independent panel of judges to classify their emotional content. Ranking targets on separate scales for different basic emotions (joy, anger, etc.) would provide a baseline for the test. Then having subjects rank each target on the same scale would give an indication of the consistency of perceived emotional content across all subjects.

Once the subject had identified the emotional content of a target, he would proceed to attempt to match that emotional content using the MetaMuse system. Such a match could be restricted to unvarying sounds only, or the subject could be permitted to create a varying sound to make her match. Finally, the resultant sound could be judged for its emotional content, with its ranking compared to that of the original target, or the sound could be presented concurrently with the original target and the match between them judged. This judging of the sound produced could be performed by either the subject or an independent panel of judges. If subjects were able to consistently produce sounds that matched the emotional content of the corresponding targets, we would conclude that MetaMuse is an expressive instrument.

Unfortunately, an experiment such as the one just described is fraught with pitfalls. Measuring or ranking emotional content in any medium is a complex task, as emotions are not well understood. The high variance likely in an emotion-ranking experiment would necessitate a large subject pool to produce significant results; it would also require considerable Psychology-based resources and commitment to complete such an experiment. Such experimentation lies outside the scope of this research.

We explored less ambitious means to explore the expressivity of our system, such as the use of questionnaires. Questionnaires allow us to get a sense of subjects' appreciation for the system, and can supplement the controllability experiment described previously. Supplementing an existing experiment with a questionnaire allows us to acquire additional information from our existing subject base with minimal additional resources. The only caveat to this approach is to ensure that the questionnaire is not so lengthy that it impedes the progress of the subject through the experiment, nor tires or frustrates the subject.

Given these caveats, we decided to solicit answers to a questionnaire after subjects had done all their trials on a particular interface. This way, subjects answer questions about an interface only once. The drawback to this approach is that feedback on individual trials, such as a subject's satisfaction with his match for each target, is not available. However, it does allow us to acquire feedback on each interface as a separate system. This will provide us with a first approximation of the expressivity of the system.

5.3 Parametric Interfaces

Three graphical user interfaces were created against which to compare MetaMuse, designed to incrementally reduce the complex elements that comprise the MetaMuse interface. These parametric interfaces use sliders and buttons to provide control over the same sound space without the addition of props, metaphor, or control amalgamation. The first interface removes the props and the pouring metaphor; the second replaces the landscape with a set of sliders to mix three droplet sounds; and the third removes even the control amalgamation of the droplet, requiring users to manipulate the synthesiser parameters directly. The three interfaces are described below.

5.3.1 GUI with Landscape Metaphor

The first GUI is closest to MetaMuse in control construction, lacking only the props. The watering can has been converted into two sliders, one repre-

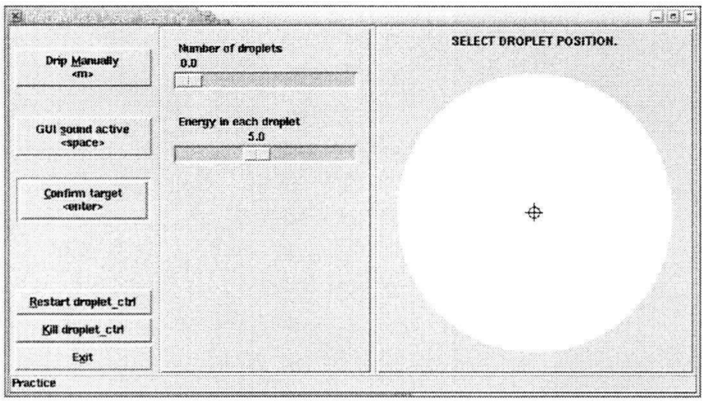


Figure 5.2: The first parametric interface uses a version of the landscape metaphor in the circle on the right but removes the props. Number of droplets and droplet energy are controlled by the two horizontal sliders.

senting its height, the other its amount of tilt. Meanwhile, the landscape is represented as a circle, with a cursor indicating the current point of action. A screenshot of this GUI is shown in Figure 5.2.

The sliders used to replicate the control of the watering can eliminate the particle model described in Section 4.3.1. In particular, in MetaMuse there is a delay between the time at which a droplet is created at the spout of the watering can and the time at which it intersects the landscape — if it does. Recall that there are also some random timing effects generated by slight differences in the initial velocities of the droplets. With the props eliminated, this GUI requires an equivalent process to provide similar control. This process is described in Appendix B. Based on subjects' lack of difficulty or comment during the user tests, the new process is indistinguishable from the MetaMuse droplet generation process.

The circle in this GUI, seen on the right in the figure, behaves exactly as does the landscape discussed in Section 4.3.4. In short, parameter sets for three droplets are defined, and are "placed" on the rim of the circle equidistant from one another. The parameter set for a given point within the circle is given by interpolating the parameter sets from the three droplets based on the point's distance from their locations on the rim. By dragging the cursor across the circle with the mouse, or clicking anywhere within the circle, the user can control parameters just as she could by moving the watering can over the landscape.

This GUI makes use of the landscape metaphor in the targeted circle and the pouring metaphor in the tilt and height sliders. The metaphors are not as explicitly presented as in MetaMuse – there are no props to embody and suggest these interaction metaphors. However, the sliders are labeled "Number of droplets" and "Energy in each droplet", which does imply the water metaphor. No specific attempt was made with this interface to obviate the metaphors, though they are present in the way the controls are grouped. Instead, users are permitted to draw associations as they like: the two sliders can be seen as a replacement for the watering can, in which they imperfectly embody the pouring metaphor, or they can be considered simply parameters.

5.3.2 GUI with Proportional Droplet Control

The second GUI is a further step removed from MetaMuse in that the landscape metaphor is discarded. Here, three sliders are used to represent proportions of the droplets that were found on the rim of the landscape in MetaMuse and the GUI of Section 5.3.1. The user uses the interface, shown in Figure 5.3, to blend the parameters of the three droplets. With the landscape metaphor eliminated, the watering can reduced to a pair of sliders, and the control delay of the particle model gone, metaphor is not a feature of this interface.

To match the control of the landscape prop and the landscape GUI, the three sliders are constrained to always sum to one. When one slider is adjusted, the other two move proportionately to maintain that sum. This allows the user to isolate the sound of a single droplet by sliding its slider to the top – the other sliders will move to the bottom automatically. This interface removes the metaphor used in the landscape GUI, but keeps the amalgamation of control. The concept of blending three droplets allows the synthesiser parameters to be manipulated as a group.

5.3.3 Full-Parameter Control GUI

In the third GUI, the synthesis parameters are exposed in full. A slider is used to manipulate each parameter in the synthesiser, as well as the droplet frequency parameter from the particle model replacement, which was present

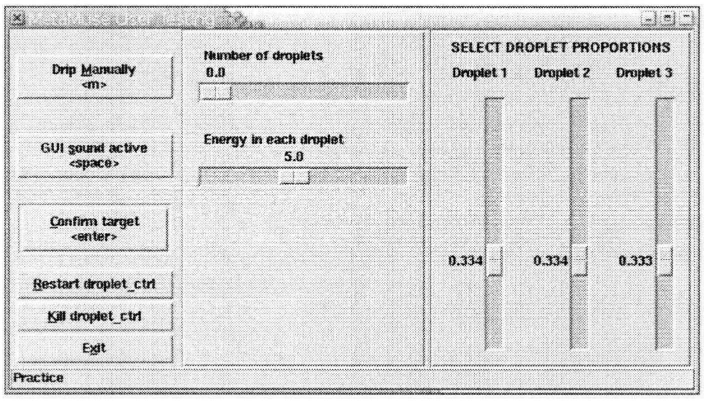


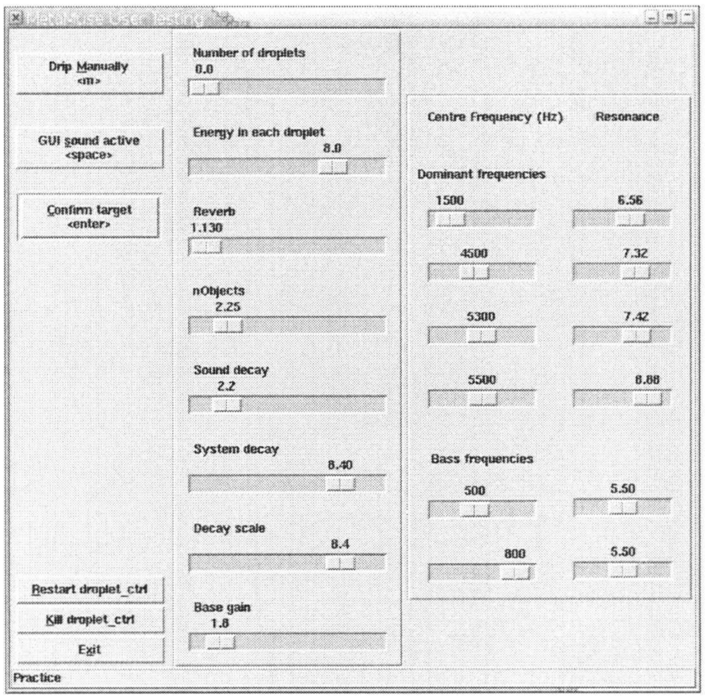
Figure 5.3: The second parametric GUI removes the metaphor but maintains aggregation of the parameters. Users can select proportions of three droplets with the three vertical sliders. Number of droplets and droplet energy are controlled by the two horizontal sliders as in the first parametric GUI.

in the previous two GUIs. The resultant GUI, with twenty sliders (some Instrument parameters are fixed to zero), can be seen in Figure 5.4.

This GUI removes metaphor entirely. It also remove the control amalgamation offered by grouping parameters into three known points and interpolating between them. Interpolating between three points in the parameter space reduced that space to a plane (see Section 4.3.4); removing that amalgamation frees the system from the plane and increases the available sound space. This interface proved to provide far too much fine control over the synthesiser. As discussed in Section 5.4.1, this GUI was eliminated from the user tests after the first pilot for this reason.

5.4 Experiments and Their Results

Having motivated the experimental approach and introduced the parametric interfaces used for comparison, we can now describe in greater detail the experiments outlined in Section 5.1.



Number of droplets 0.0
Energy in each droplet 8.0
Reverb 1.130
nObjects 2.25
Sound decay 2.2
System decay 8.40
Decay scale 8.4
Base gain 1.8

	Centre Frequency (Hz)	Resonance
Dominant frequencies	1500	6.56
	4500	7.32
	5300	7.42
	5500	8.88
Bass frequencies	500	5.50
	800	5.50

Figure 5.4: The full-parameter control GUI exposes all the parameters available in the synthesiser and allows the user to adjust them individually. Users performed significantly slower on this interface in the first pilot, prompting it to be dropped from further evaluation.

5.4.1 First Pilot Experiment: Measuring Controllability

The first experiment was an informal pilot intended to verify the experimental procedure. The choices of interface presentation order, target presentation order, and other experimental decisions were tuned based on the results of this experiment. Several other lessons were learned, including the recognition of some system deficiencies as well as a need to revise the experimental procedure to reduce learning effects. The procedure itself, and the lessons learned, are described in the following two sections.

Experimental Procedure

As discussed in Section 5.2.2, we chose a target-matching task. We had three factors in mind when setting our experimental order: experimental duration, provision for practice time, and ordering to avoid knowledge transfer between interfaces. In this section we discuss how these factors resulted in the experimental procedure used.

Our goal in setting the duration of the experiment was to have subjects work as long as possible without tiring. By having subjects do many trials, we collect as much data as possible from the minimum number of subjects. However, once a subject starts to tire, his performance likely drops, so our accuracy depends on not overworking the subjects. Our target time for the experiment was 1.5 hours. We estimated that 20 trials on each of MetaMuse and two parametric interfaces, plus practice time on each interface, would meet that goal. We followed standard experimental practices of randomizing task order between subjects and randomizing trial order between tasks.

MetaMuse and the three parametric GUIs are all newly invented interfaces, and the synthesis engine is a newly invented sound space. These facts guarantee that all users will be novices. Ideally we would bring users in for repeated practice sessions to bring them to expert skill level, but this is impractical given our resources. Instead, we aimed for comfortable familiarity, and instructed users to "practice until they felt comfortable with the interface." If users spent more than ten minutes practicing, we encouraged them to continue with the experiment to avoid fatigue. In practice this was rarely a concern.

In deciding the order of task presentation, our primary goal was to avoid knowledge transfer as much as possible. That is, we wanted users to have as little experience with interfaces B and C while testing interface A. We therefore had users practice on one interface, then do the trials for that interface, before continuing to the next interface. One example of knowledge transfer that this technique avoids is the application of metaphors learned in MetaMuse to the metaphor-free second parametric interface. As we shall see in the next section, this approach lead to other problems in our data analysis.

Results

The first pilot allowed some interesting observations to be made about the way users use MetaMuse and the other interfaces. It also exposed several problems with the system and provided insight into the experimental procedure. The systemic issues caused reliability problems during testing and prompted significant reprogramming of several software modules. The procedural issues were less severe, but required revision of the experimental process. One issue, the distance metric used, spanned both categories and will be discussed after the other problems are explained.

It was very educational to see how users explored the different interfaces. Some users approached the new interfaces very systematically, exploring the axes of movement or GUI sliders in sequence to try to understand the interfaces. Many users tried to explore the limits of MetaMuse, pouring and catching droplets in unusual ways. They would tilt the watering can backwards or invert it to see if they could pour from the filling hole, or turn the landscape upside down to see what sounds the back side made. As described in Section 4.1.1, several of these explorations weren't implemented and therefore caused behaviours that fell outside the system metaphors. However, users were universally able to recognise the boundaries of the system and return to the expected range of behaviours without difficulty. From these observations we conclude users are able to recognise those parts of the system metaphor that are not important to "normal" operation. We therefore had no concerns leaving these behaviours unimplemented for the remainder of the experiments.

One exploratory behaviour many users performed was to move the droplets along the perceived axes of the landscape, evidently assuming that the mapping decomposed on these axes. In fact it does not: the metaphor and mapping for the landscape rely on the interpolation technique described in Sections 4.1.3 and 4.3.4. Though we didn't recognise it at the time, this behaviour indicates that the system behaviour is under-determined by the system metaphors. Specifically, though the landscape metaphor suggests that the sound will change with position on the landscape, it fails to specify *how* it will change. This issue continued to be a problem throughout the experiments, but was only recognised after the formal experiment. In retrospect, the theory of transparency described in Chapter 3 predicted this shortcoming: the component of the mapping that is not addressed by the system metaphor was arbitrarily designed and resulted in the least understood part of the interface.

Several bugs in the testing system's software (as opposed to the MetaMuse software, which had no issues) were exposed by the user testing. These resulted in occasional system crashes during testing, as well as random failure to output log files. The primary cause was improper parsing of messages in the communication subsystem. Other problems identified included an off-by-one error in a data array. Once these problems were resolved the system was quite reliable: no further crashes or data loss occurred during the remainder of the experiments.

The target matching approach itself was, overall, validated by the pilot. Useful data was acquired during the pilot, though due to the data logging bug not enough was acquired for formal analysis. Some minor adjustments were made to the process, such as reducing the number of targets to 15 to correct the over-long duration of the experiments. Users reported that the landscape was too diminutive compared to the watering can, so a larger landscape was created.

A major concern from the pilot was the learning effect. The data acquired indicated that users were learning a great deal from interface to interface, so that their performance often depended on order rather than interface. This made it difficult to determine performance relative to the interface and superseded our earlier concern for knowledge transfer issues. We decided to alter the experimental order, providing subjects with practice time on all three interfaces before beginning their recorded trials. This makes it difficult to consider the parametric interfaces to be completely lacking in metaphor, as users are able to transfer skills and abstractions between interfaces. For example, the pouring metaphor learned in MetaMuse may be transferred to the two sliders used to control number of droplets and droplet energy. However, learning effects make the statistical analysis impossible barring a much larger sample set, so this restriction must be accepted.

One unsurprising result of the pilot was that the third parametric interface, which offered full control over twenty parameters, was so fine-grained that users' performance with it was consistently much poorer than with the other interfaces. The performance difference was so marked that the low-level interface was eliminated from further consideration and was not used in subsequent experiments.

The accuracy metric was an issue that spans both experimental and systemic categorization. As discussed in Section 5.2.2, the original accuracy metric used all synthesiser parameters, equally weighted, in a Euclidean distance calculation to determine subjects' distance from target. In the pilot, many subjects commented that the performance feedback they received seemed both inaccurate and inconsistent. In particular, sounds audibly quite different from their target sounds would occasionally be rated as very close, while those nearly indistinguishable were sometimes rated as distant. It became apparent that additional experimentation would be required to determine the link between the perceptual space and the parameter space.

5.4.2 Just-Noticeable Difference Experiment

With the first pilot indicating that the accuracy metric was faulty, an experiment was performed to determine the just-noticeable difference of the various parameters. The purpose of the JND experiment was to ascertain the perceptive effect of each synthesiser parameter. Subjects were given two sounds with a possible parametric difference between them, and were asked to indicate if they sounded different. The results showed that some parameters had significantly less effect on the perceived sound than did others, and the distance metric was changed to take these differences into account.

Experimental Procedure

The basic approach of this experiment was to present the subject with two sounds and ask them to judge whether they differed. Pairs of sounds were created by choosing one of several starting points in parameter space, then varying a single parameter some amount. In some cases, the sounds were purposefully identical; subjects were instructed that some pairs were the same and some different, but not the proportions thereof. The experiment was run using a modified version of the the parametric GUIs described in Section 5.3.

Results

The results of the JND test were surprising: some parameters had very little bearing on the perceived sound. In particular, subjects had difficulty distinguishing between sounds with different droplet rates, except at frequencies so low that individual droplets could be distinguished. Droplet frequency and several other non-significant parameters were therefore dropped from the accuracy calculation.

We then revisited the mapping to explore the correlations between the remaining sounds. In Section 4.3.2 we described how the sound-space was made navigable by restricting motion to a plane in that space. That restriction meant that some sets of parameters were directly or inversely correlated. By removing the less-significant parameters of these correlated sets, we reduced redundancy in the accuracy calculation.

After removing non-significant and redundant parameters, only two parameters remained. These were combined in a calculation similar to the original Euclidean distance calculation to produce an accuracy metric that was used for the remainder of the experiments.

5.4.3 Second Pilot Experiment: Comparing Controllability and Expressivity

With the issues of the first pilot resolved, and the new metric based on the JND results incorporated into the code, we performed a second pilot to validate our changes. Before doing so, however, we added a questionnaire component to the experiment to collect expressivity data. The second pilot verified our changes before we moved on to the formal user test.

In Section 5.2.3, we discussed possible ways to measure expressivity. The more comprehensive experiments we explored in that section required resources beyond our means, so we had decided to use a questionnaire. Before we began this pilot, we evaluated our time-line and decided that a separate experiment involving the questionnaire would not be possible, so we decided to add the questionnaire to this experiment.

We constructed a questionnaire with components before, during, and after the experiment. Before the experiment, basic demographic questions relating to qualities such as gender and previous musical experience were asked on a paper questionnaire. After the set of trials for each interface, questions were asked pertaining to the subjects' experience with that interface, again on paper. Finally, at the end of the experiment an oral exit interview was performed, covering a predetermined set of topics. The second pilot was run on a small set of subjects who had also participated in the first pilot.

The pilot validated the experimental procedure and verified the changes we had made after the first two experiments. The system was in good working order, and the experiment acquired the data we desired in a manageable time-frame. We also determined that the questionnaire worked well in the context of the existing experiment; some questions were tweaked and a few demographic questions added to fine-tune the questionnaire.

With the experiment prepared and verified, we were now ready to undertake the formal test.

5.4.4 Formal User Experiment

After the second pilot, the experimental process was finalised with the changes discussed in the previous section. The experiment was run on twelve subjects, though the data for one subject was discarded due to technical problems. The final set of data consisted of time and accuracy measurements, and users' responses to the questionnaire. The data contained non-significant results for the controllability part of the experiment, and both significant and non-significant results for the expressivity part of the experiment.

We performed analysis of variance (ANOVA) on the data. Our controllability hypothesis is: MetaMuse is more controllable than a typical GUI controller. We formalised this into the following null hypotheses:

1. Subjects' distance from target will be no different using MetaMuse than using either GUI.

2. Subjects' target acquisition time will be no different using MetaMuse than using either GUI.

The top and bottom value of each measurement was discarded, and the remaining values averaged. These results can be seen in Figure 5.5. The ANOVA tables for these null hypotheses are shown in Figures 5.6 and 5.7. The results show a trend indicating that the MetaMuse interface is somewhat faster than the other interfaces — 7% to 12% faster — but has about the same accuracy. However, these results fail to show significance (F=0.693 for distance; F=2.9068 for time; F=3.49 needed for $p<0.05$). Therefore we are unable to reject the null hypotheses.

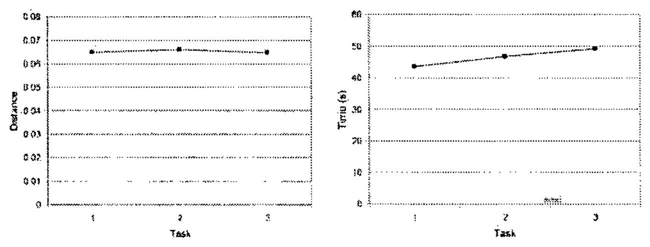


Figure 5.5: Mean distance and mean time for the middle nine subjects, where MetaMuse is Task 1. Lower is better in both tables. Distance from target was similar across all interfaces, but time to acquire target was better for MetaMuse.

ANOVA for distance

Source	SS	df	MS	F
Levels	1.965E-05	2	9.826E-06	0.0693
Subjects	6.468E-03	10	6.468E-04	
Residual	2.836E-03	20	1.418E-04	
TOTAL	9.324E-03	32		

Figure 5.6: The ANOVA table for distance. The F value is 0.0693; 3.49 or greater is required for $p<0.05$. The distance result, therefore, shows no significance.

ANOVA for Time

Source	SS	df	MS	F
Levels	262.8	2	131.4	2.9068
Subjects	7611.2	10	761.1	
Residual	903.9	20	45.2	
TOTAL	8777.9	32		

Figure 5.7: The ANOVA table for time. The F value is 2.9068; 3.49 or greater is required for $p<0.05$. The time result, therefore, shows no significance.

MetaMuse's improvement for statement	vs. GUI 1	vs. GUI 2
Enjoyable?	36.1%	28.9%
Adequate control?	30.3%	2.38%
Difficult to understand?	25.0%	14.3%
Creative?	38.2%	46.8%
Satisfied with performance?	0%	14.7%

Table 5.1: Percent that average response for MetaMuse showed improvement over those for the other two interfaces. In every case but one, users preferred MetaMuse. Note that, except for creativity, these results are not significant. They indicate trends, but do not allow us to draw formal conclusions.

We performed ANOVA on the five task-specific questionnaire answers. Our null hypotheses took the form: subjects' responses to question N will be no different for MetaMuse than for the other GUIs. The questions asked after each trial were answered on an integer scale, so responses could be averaged and ANOVA applied. As in the controllability case, the top and bottom values for each case were discarded before averaging. The averages for the five questions are shown in Figure 5.8, and the ANOVA tables in Figure 5.9.

The only question to show significance was "I felt creative using this interface." On a scale of 1 ("strongly disagree") to 6 ("strongly agree"), subjects agreed with this statement with an average score of 5.22 for the MetaMuse interface, compared to 3.78 and 3.56 for the two parametric interfaces. These results are significant ($F=6.008$, $p<0.01$). In other words, whereas subjects were on average neutral or slightly positive about the parametric interfaces, they were convincingly positive about feeling creative on the MetaMuse interface. Therefore we can reject the null hypothesis for this question and conclude that subjects felt considerably more creative using MetaMuse.

The other questions did not show significant results. We do see trends indicating a preference for MetaMuse on all questions, as shown in Table 5.1.

Subjects were also given an informal interview at the end of the experiment, and many comments made during these interviews support our findings. One subject reported that "the prop interface was way more expressive" in the rhythms that could be made. This subject also indicated "definitely the prop interface had a big advantage in . . . the continuum, being able to control simultaneously all [five] parameters." This feedback indicates that the subject preferred MetaMuse both for expressivity and for controllability.

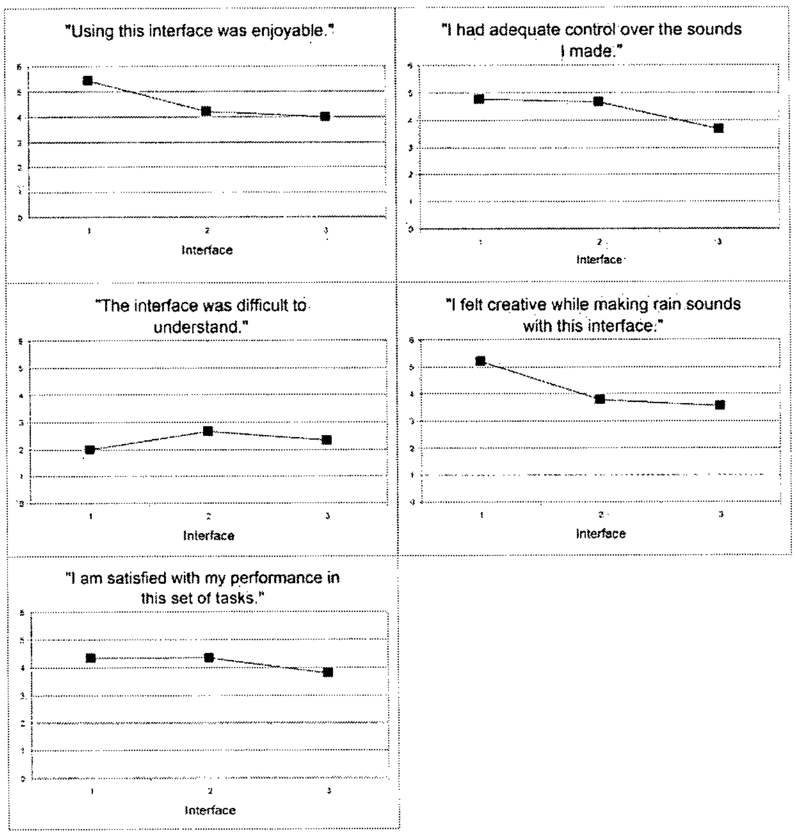


Figure 5.8: Mean questionnaire responses for the middle nine subjects, where MetaMuse is Interface 1. Possible responses ranged from 1 ("disagree strongly") to 6 ("agree strongly"). In most cases, higher is better; in the case of the question on difficulty lower is better. MetaMuse generally performed better than the other interfaces. In particular, for the statement "I felt creative using this interface," the middle nine user responses averaged 5.22 for MetaMuse compared to 3.78 and 3.56 for the other interfaces.

ANOVA: enjoyable

Source	SS	df	MS	F
Levels	10.424	2	5.212	2.679
Subjects	12.909	10	1.291	
Residual	38.909	20	1.945	
TOTAL	62.242	32		

ANOVA: adequate control

Source	SS	df	MS	F
Levels	5.091	2	2.545	1.647
Subjects	13.636	10	1.364	
Residual	30.909	20	1.545	
TOTAL	49.636	32		

ANOVA: difficult to understand

Source	SS	df	MS	F
Levels	2.364	2	1.182	0.747
Subjects	38.182	10	3.818	
Residual	31.636	20	1.582	
TOTAL	72.182	32		

ANOVA: felt creative

Source	SS	df	MS	F
Levels	17.515	2	8.758	6.008
Subjects	19.576	10	1.958	
Residual	29.152	20	1.458	
TOTAL	66.242	32		

ANOVA: satisfied with performance

Source	SS	df	MS	F
Levels	1.697	2	0.848	0.645
Subjects	17.515	10	1.752	
Residual	26.303	20	1.315	
TOTAL	45.515	32		

Figure 5.9: The ANOVA table for the questionnaire responses. An F value of 3.49 is required for $p<0.05$ significance. The only significant result is for the statement "I felt creative using this interface" ($F=6.008$, $p<0.01$).

Another subject indicated that MetaMuse "just felt more intuitive" and said it was the easiest interface to use. The subject's use of the word "intuitive" is telling. For this subject, the system metaphor shone through clearly, making MetaMuse easy to understand and therefore easy to use.

Another subject offered similar sentiments, saying, "using the prop, the sounds sound like what you would expect." Again, the use of metaphor has made the device easy to understand. This subject referred to their own literature: "you learn as a child how much you're going to pour; it just makes sense."

This feedback confirms the conclusion that MetaMuse is more expressive than the other interfaces and affirms that it is the use of metaphor that makes it so.

CHAPTER 6

CONCLUSIONS AND FUTURE WORK

In this thesis we presented metaphor as a tool for designing and playing expressive instruments. We introduced the concept of the transparency of a device mapping as a facilitator for expressivity, and metaphor as an approach to improving transparency. The MetaMuse system was presented to demonstrate the use of metaphor to increase expressivity. Finally, MetaMuse was evaluated in a user study to determine its controllability and expressivity.

The five major contributions of this thesis are:

1. The theory of transparency, which provides a framework to predict and evaluate the expressivity of musical devices.
2. The investigation of metaphor as a facilitator for designing expressive devices, and for learning and playing them, by increasing transparency.
3. Demonstration of the above concepts in the form of MetaMuse, a novel expressive device based on metaphor. MetaMuse and its evaluation show that transparency and metaphor are useful constructs.
4. A novel form of synthesis combining techniques from real-time stochastic synthesis and granular synthesis.
5. The Polhemus library, which provides simplified multi-threaded access to the Polhemus Fastrak and creates a standard framework that extends to other serial devices.

6.1 Transparency and Metaphor

We introduced transparency to provide an indication of the ease with which the mapping of a device's input to its output is understood by its player. This understanding facilitates expressivity. It can be improved by the application of metaphor, which refers the player to a literature of common knowledge.

Metaphor is applicable to the design of new devices, in that it can obviate the appropriate controller for a synthesiser or vice versa. It is also applicable to performance with such devices, where it helps the player and audience understand the instrument mapping. Both of these applications facilitate expressivity.

6.2 MetaMuse

We presented MetaMuse as a device based on metaphor. MetaMuse uses three metaphors from the literature of water — pouring, rainfall, and landscape. These metaphors guided the design of the device from the inception of the rain-sound synthesiser to the matching of the prop-based control paradigm and the mapping that connects them.

6.3 Evaluation

MetaMuse was evaluated in a series of user tests to evaluate the success of the metaphor-oriented design approach. Two qualities were measured:

Expressivity to validate the system as a demonstration of the transparency theory in practice, measured by questionnaire; and

Controllability to verify that the device provided users with the basic ability to navigate to specific sounds, measured by performance data.

The results of the experiments were positive. Users indicated that the MetaMuse interface was more creative to use than the other two interfaces. It was rated at 5.2 on a six-point scale, where the comparator interfaces were rated 3.8 and 3.6. This result was significant ($F=6.008$, $p<0.01$).

Other areas of the expressivity component did not show significance, but did show trends that preferred MetaMuse as generally more enjoyable, controllable, understandable, and satisfying than the other interfaces. Similarly, the controllability component did not show statistical significance, but it did indicate a trend that MetaMuse is as controllable as, or more controllable than, the parametric GUIs to which it was compared.

Users' feedback during the informal interviews also support our results. One user said that MetaMuse "just felt more intuitive" than the other interfaces, confirming the value of the system metaphor. Another was more explicit: "using the prop, the sounds sound like what you would expect." Clearly the metaphor-based design approach, and the embodiment of those metaphors in the interface, provide a distinct advantage.

The results of our work demonstrate that using metaphor is a valid approach to increasing the expressivity of an instrument. The theoretical framework of transparency is a valuable contribution that provides the context for this work. We believe that this direction of research remains fertile, and present suggestions for future work in Section 6.5.

6.4 Publications Based on Thesis Contributions

Contributions from this thesis have also been published in several peer-reviewed publications, including the ACM Special Interest Group on Computer Human Interaction, 2002 [20], New Interfaces for Musical Expression, 2002 [21], and Organised Sound: an International Journal of Music and Technology, 2003 [17].

6.5 Future Work

Additional research can be done in three areas: further testing of MetaMuse, further development of the system, or development of new systems based on transparency and metaphor.

6.5.1 Further experimentation

There are several ways in which the experimentation described in the previous chapter could be extended. Both the controllability and expressivity experiments could be modified to improve their significance and offer stronger conclusions; different experiments could be performed to further delve into the expressivity of the system; and the expressivity with respect to audiences could also be tested.

The primary drawback to the controllability experiment was the lack of significance. This issue could be remedied simply by running additional subjects through the same controllability trial to gather more data. Ideally, such an experiment would provide solid evidence to support the trends identified in Section 5.4.4.

While the expressivity component of the experiment provided a valuable conclusion, it did not meet its potential. Several other questions were asked in the questionnaire, but only one resulted in a significant conclusion. Additionally, the question that was significant considered creativity, which is peripheral to the specific question of expressivity. The questionnaire could be modified to provide more appropriate questions, then run on more subjects to improve significance.

A further extension to the measurement of expressivity was introduced in Section 5.2.3. Measuring expressivity would be better accomplished using targets designed to evoke emotions. More generally, expressivity is better studied in an established framework. More research into the psychology of expression and emotion is required to design an appropriate test plan.

Finally, in Chapter 3 we defined expressivity as the act of communicating meaning or feeling. In our experiments with MetaMuse we have focussed on the design and playing of expressive devices, but have not considered expression from the audience perspective. It is valid to test whether the player feels expressive while using the device, but it would also be instructive to determine what meaning or feeling the audience perceives during a performance. Such an experiment would require significant investment for a player to become highly proficient for a performance, but would provide stronger conclusions about the efficacy of metaphor as a design tool.

6.5.2 Further development

Further research could be performed with incremental changes to the MetaMuse system. Improvements can be made to the metaphor embodiment of the landscape prop. However, the greatest benefits could be made by improving the synthesiser.

As discussed in Section 5.4.4, the landscape metaphor does not fully constrain the sound of the synthesiser. In particular, while the metaphor implies that the sound will change as the droplets move across the landscape, it does not suggest how this might happen. Exacerbating this problem is the lack of discernible logic to the landscape mapping: there is no guiding principle to help the user understand how the sound changes. Consequently users are forced to use trial and error to find specific sounds on the landscape.

This problem could be resolved in one of two ways. First, the landscape could be given a simple logical mapping, such as perpendicular axes of, say, pitch and timbre. Then it is clear that a specific perceptual parameter is varied based on a specific movement. However, this may restrict the sound space, as there are not two clear perceptual parameters to be presented in this way.

The other solution would be to extend the metaphor in some way. One way to do so would be to remove the abstraction of the landscape and present certain areas as specific surface elements — rocks, trees, grass, or water. By careful mapping, droplets that match each surface type would be mapped to that type. If care were taken with the original mapping, a consensus of the "correct" sound for a visible surface element might be attained. However, the sound would still be under-constrained within a region. How should the sound of water on grass change, for example, as the droplets move from the centre of that region towards the rocky region? More investigation needs to be done into this problem.

The synthesiser itself would benefit greatly from further work. While the individual droplets sound reasonably accurate, their variation could be expanded. Most of the droplets in the sound space have a sharp sound as though they're hitting a hard surface. Increasing the variety of sounds available would add expressive depth to the device. Additional improvement could be done to the multi-droplet sound.

As discussed in Section 4.3.2, the synthesiser deviated from natural rain sounds as the droplet rate increased. While heuristics were introduced to improve the sound, it still fell short of expectations. One way to improve the sound would be to optimise the Instrument code to attain better performance. This would allow a larger number of droplets to be synthesised simultaneously. A second way would be to explore other heuristics, and a third way would be to explore synthesis techniques that provide the aggregate sound directly, such as that developed recently by van den Doel, discussed in Section 2.4.2.

Research beyond these methods requires investigation into new systems using metaphor. We would especially like to see this research extended to musical systems, as we received comments from users suggesting that music would be a more accessible medium. Finally, we believe that these ideas are transferable to other fields of human interaction, including human-human,

human-computer, and human-machine interaction, and would welcome such research.

6.6 Conclusion

This thesis provides a framework for understanding and predicting the expressive nature of devices. We introduce *transparency*, or the ease with which a device's mapping can be understood, as the main pillar of this framework. Transparency is a predictor for *expressivity*, the ease with which meaning or emotion can be communicated through the device. We explore the role of metaphor for improving the amount of expression possible with a device, and examine its use as both a design aid and an affordance during use.

To validate this theory we built MetaMuse, a controller for a rain-sound synthesiser that employs water metaphors to obviate its use. MetaMuse uses physical props to embody three metaphors: pouring, rainfall, and landscape. These metaphors acted as a guide during MetaMuse's design, and aid users in the prediction of system operation and sound output.

We performed a series of experiments comparing MetaMuse to two other interfaces to test our claim that metaphor can be used to improve expressivity. The tests measured controllability with speed and accuracy of sound target acquisition, and expressivity with qualities such as creativity. Accuracy was similar across all interfaces, and speed slightly improved for MetaMuse, but the controllability results lacked significance. Results for expressivity were significant, however: users preferred MetaMuse as more creative than two more traditional controllers ($F=6.0$, $p<0.01$). From these results we conclude that metaphor is a worthy design approach to creating expressive devices.

We believe that there is considerable fruit to be borne in continued research into metaphor as a facilitator for expressivity. The results of our work show a clear benefit to metaphor's use in this way in MetaMuse. We expect this benefit is scalable to more complex systems than that presented herein, and to highly expressive computer-based musical instruments in particular. We look forward to continued exploration of metaphor leading to the emergence of new expressive instruments supporting virtuosic playing.

BIBLIOGRAPHY

[1] Ronald M. Baeker, Jonathan Grudin, William Buxton, and Saul Greenberg. *Readings in Human-Computer Interaction: Toward the Year 2000.* Morgan Kaufmann Publishers, 1995.

[2] C. Bahn and D. Trueman. interface: electronic chamber ensemble. In *ACM SIGCHI Workshop on New Interfaces for Musical Expression*, page electronic proceedings, Apr 2001.

[3] Ravin Balakrishnan and Ken Hinckley. The role of kinesthetic reference frames in two-handed input performance. In *Proceedings of the 12th annual ACM symposium on User interface software and technology*, pages 171–178. ACM Press, 1999.

[4] Eric A. Bier, Maureen C. Stone, Ken Pier, William Buxton, and Tony D. DeRose. Toolglass and magic lenses: the see-through interface. In *Conference on Human Factors in Computing Systems*, pages 73–80. ACM Press, 1994.

[5] Tina Blaine and Sidney S. Fels. Collaborative intefaces for musical expression. *Journal of New Music Research*, 32(4):411–428, Dec 2003.

[6] Tina Blaine and Tim Perkis. The jam-o-drum interactive music system: A study in interaction design. In *Proceedings of ACM Symposium on Designing Interactive Systems (DIS)*, pages 165–173, 2000.

[7] William A.S. Buxton. A three-state model of graphical input. In *Human-Computer Interaction - INTERACT '90*, pages 449–456. Elsevier Science Publishers B.V. (North-Holland), 1990.

[8] C. Cadoz. Instrumental gesture and musical composition. In *Proceedings of International Computer Music Conference*, 1988.

[9] Stuart K. Card, Jock D. Mackinlay, and George G. Robertson. The design space of input devices. In *Proceedings of ACM CHI'90 Conference on Human Factors in Computing Systems*, pages 117–124. ACM Press, 1990.

[10] Perry Cook. Principles for designing computer music controllers. In *ACM CHI Workshop in New Interfaces for Musical Expression (NIME)*, April 2001.

[11] Immersion Corporation Cyberglove. http://www.immersion.com/3d/products/cyber_glove.php, 2005.

[12] Institute de Recherche et Coordination Acoustique/Musique (IRCAM). http://www.ircam.fr/produits/logiciels/jmax-e.html, 2002.

[13] ZOOM RhythmTrak RT-323 drum machine. http://www.zoom.co.jp/english/models/rt323/pdmodel.html, 2005.

[14] Polhemus FASTRAK. http://www.polhemus.com/fastrak.htm, 2005.

[15] S. S. Fels and G. E. Hinton. Glove-talkII: A neural network interface which maps gestures to parallel formant speech synthesizer controls. *IEEE Transactions on Neural Networks*, 9:205–212, 1998.

[16] S. S. Fels and K. Mase. Iamascope: A graphical musical instrument. *Computers and Graphics*, 2:277–286, 1999.

[17] Sidney S. Fels, Ashley Gadd, and Axel Mulder. Mapping transparency through metaphor: Towards more expressive musical instruments. *Organised Sound: an International Journal of Music and Technology*, 7(2):109–126, 2003.

[18] Sidney S. Fels and Florian Vogt. Tooka: Explorations of two person instruments. In *2nd International Conference on New Interfaces for Musical Expression (NIME02)*, pages 116–121, May 2002.

[19] F.G. Fowler and H.W. Fowler. *The Pocket Oxford Dictionary of Current English 4e.* Oxford University Press, 1957.

[20] A. Gadd and S. S. Fels. MetaMuse: a novel control metaphor for granular synthesis. In *Proceedings of the ACM Special Interest Group on Computer Human Interaction (SIGCHI'02)*, Apr 2002.

[21] A. Gadd and S. S. Fels. MetaMuse: Metaphors for expressive instruments. In *Proceedings of New Interfaces for Musical Expression (NIME'02)*, May 2002.

[22] C. Goudeseune, G. Garnett, and T. Johnson. Resonant processing of instrumental sound controller by spatial position. In *ACM SIGCHI Workshop on New Interfaces for Musical Expression*, page electronic proceedings, Apr 2001. Available online at http://www.csl.sony.co.jp/person/poup/research/chi2000wshp/papers/goudeseune.pdf.

[23] Yves Guiard. Asymmetric division of labor in human skilled bimanual action: The kinematic chain as a model. *Journal of Motor Behaviour*, 19:486–517, 1987.

[24] Michael Gurevich and Stephan von Muehlen. The accordiatron: A midi controller for interactive music. In *ACM CHI Workshop in New Interfaces for Musical Expression (NIME)*, April 2001.

[25] Ken Hinckley, Randy Pausch, John C. Goble, and Neal F. Kassell. Passive real-world interface props for neurosurgical visualization. In *Conference on Human Factors in Computing Systems*, pages 452–458. ACM Press, 1994.

[26] Ken Hinckley, Randy Pausch, John C. Goble, and Neal F. Kassell. A survey of design issues in spatial input. In *Proceedings of the 7th annual ACM symposium on User interface software and technology*, pages 213–222. ACM Press, 1994.

[27] Ken Hinckley, Randy Pausch, Dennis Proffitt, and Neal F. Kassell. Two-handed virtual manipulation. *ACM Transactions on Computer-Human Interaction*, 5:3:260–302, September 1998.

[28] Ken Hinckley, Randy Pausch, Dennis Proffitt, James Patten, and Neal Kassell. Cooperative bimanual action. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 27–34. ACM Press, 1997.

[29] William Hudson. Why metaphor is a double-edged sword. *interactions*, 7:3:11–15, May/June 2000.

[30] A. Hunt, M. Wanderley, and R. Kirk. Towards a model for instrumental mapping in expert musical interaction. In *Proceedings of the International Computer Music Conference (ICMC2000)*, Berlin, Germany, 2000.

[31] Andy Hunt, Marcelo M. Wanderley, and Matthew Paradis. The importance of parameter mapping in electronic instrument design. In *Proceedings of New Interfaces for Musical Expression (NIME '02)*, pages 149–154. University of Limerick, Department of Computer Science and Information Systems, 2002.

[32] H. Ishii and B. Ullmer. Tangible bits: Towards seamless interfaces between people, bits and atoms. In *Proceedings of Conference on Human Factors in Computing Systems (CHI '97)*, pages 234–241, Atlanta, GA, march 1997. ACM.

[33] S. Jorda. New musical interfaces and new music-making paradigms. In *New Instruments for Musical Expression Workshop*, Seattle, WA, 2001.

[34] Sergi Jorda. New musical interfaces and new music-making paradigms. In *ACM CHI Workshop in New Interfaces for Musical Expression (NIME)*, April 2001.

[35] Paul Kabbash, William Buxton, and Abigail Sellen. Two-handed input in a compound task. In *Proceedings of CHI '94: Human Factors in Computing Systems*, pages 417–423. ACM Press, April 1994.

[36] Wendy Leeds-Hurwitz. *Semiotics and Communication: Signs, Codes, Cultures.* Lawrence Earlbaum Associates, 1993.

[37] Kim Halskov Madsen. Magic by metaphors. In *Proceedings of DARE 2000 on Designing augmented reality environments*, pages 167–169. ACM Press, 2000.

[38] A. N. Marx. Using metaphor effectively in user interface design. In *Proceedings of Computer-Human Interaction (CHI '94)*, Boston, MA, 1994. ACM.

[39] Kenji Mase and Tomoko Yonezawa. Body, clothes, water and toys. In *Proceedings of New Interfaces for Musical Expression (NIME '02).* University of Limerick, Department of Computer Science and Information Systems, 2002.

[40] Graeme McCaig and Sidney S. Fels. Playing on heart-strings: Experiences with the 2Hearts system. In *2nd International Conference on New Interfaces for Musical Expression (NIME02)*, pages 54–59, May 2002.

[41] F. R. Moore. The dysfunctions of MIDI. *Computer Music Journal*, 12(1):19–28, Spring 1988.

[42] A. Mulder and S. S. Fels. Sound sculpting: Manipulating sound through virtual sculpting. In *Proceedings of the Western Computer Graphics Symposium'98*, Whistler, Canada, April 1998.

[43] D. A. Norman. *The Invisible Computer.* MIT Press, Cambridge, MA, 1998.

[44] Donald Norman. *The Design of Everyday Things.* Currency/Doubleday, 1990.

[45] Roberto Oboe and Giovanni De Poli. Multi-instrument virtual keyboard – the MIKEY project. In *Proceedings of New Interfaces for Musical Expression (NIME '02).* University of Limerick, Department of Computer Science and Information Systems, 2002.

[46] Sile O'Modhrain. Foreword. In *Proceedings of New Interfaces for Musical Expression (NIME '02)*, pages vii–viii. University of Limerick, Department of Computer Science and Information Systems, 2002.

[47] Sile O'Modhrain and Georg Essl. Pebblebox and crumblebag: Tactile interfaces for granular synthesis. In *Proceedings of New Interfaces for Musical Expression (NIME '02).* University of Limerick, Department of Computer Science and Information Systems, 2002.

[48] Nicola Orio, Norbert Schnell, and Marcelo M. Wanderley. Input devices for musical expression: Borrowing tools from hci. In *ACM CHI Workshop in New Interfaces for Musical Expression (NIME)*, April 2001.

[49] J. Paradiso. Electronic music interfaces: new ways to play. In *IEEE Spectrum Magazine*, volume 34:12, pages 18–30, 1997.

[50] J. Paradiso, K. Hsiao, and A. Benbasat. Musical trinkets: New pieces to play. In *SIGGRAPH 2000 Conference Abstracts and Applications*, page 90. ACM Press, 2000.

[51] J. Paradiso, K. Hsiao, and E. Hu. Interactive music for instrumented dancing shoes. In *Proceedings of the 1999 International Computer Music Conference*, pages 453–456, 1999.

[52] Joseph A. Paradiso, Kai yuh Hsiao, and Ari Benbasat. Tangible music interfaces using passive magnetic tags. In *ACM CHI Workshop in New Interfaces for Musical Expression (NIME)*, April 2001.

[53] Maxwell J. Roberts and Riccardo Russo. *A Student's Guide to Analysis of Variance*. Routledge, 1999.

[54] Dominic Robson. PLAY!: Sound toys for the non musical. In *ACM CHI Workshop in New Interfaces for Musical Expression (NIME)*, April 2001.

[55] D. Rokeby. Transforming mirrors: Subjectivity and control in interactive media. In *Critical Issues in Electronic Media*. SUNY Press, Albany, NY, 1995.

[56] G. Scavone and P. Cook. Synthesis toolkit in C++ (STK). In Ken Greenebaum and Ronen Barzel, editors, *Audio Anecdotes*. A.K. Peters Press, 2004.

[57] Norbert Schnell and Marc Battier. Introducing composed instruments, technical and musicological implications. In *Proceedings of New Interfaces for Musical Expression (NIME '02)*. University of Limerick, Department of Computer Science and Information Systems, 2002.

[58] Tamara Smyth and Julius O. Smith III. Creating sustained tones with the cicada's rapid sequential buckling mechanism. In *Proceedings of New Interfaces for Musical Expression (NIME '02)*. University of Limerick, Department of Computer Science and Information Systems, 2002.

[59] Laurie Spiegel. Editor's notes: an alternative to a standard taxonomy for electronic and computer instruments. *Computer Music Journal*, 16:3, 1992.

[60] D. Svanaes and W. Verplank. In search of metaphors for tangible user interfaces. In *Proceedings of DARE 2000: On Designing Augmented Reality Environments*. ACM, 2000.

[61] Leonello Tarabella and Graziano Bertini. Giving expression to multimedia performance. In *International Multimedia Conference: Proceedings of the 2000 ACM workshops on Multimedia*, pages 35–38. ACM Press, 2000.

[62] B. Truax. Real-time granular synthesis with a digital signal processor. *Computer Music Journal*, 12(2):14–26, 1988.

[63] D. Trueman and P. Cook. Bossa: The deconstructed violin reconstructed. In *Proceedings of the International Computer Music Conference (ICMC '99)*, Beijing, Oct 1999.

[64] Kees van den Doel. Physically-based models for liquid sounds. In *Proceedings of ICAD 04 — International Conference on Auditory Display,*, 2004.

[65] Erich M von Hornvostel and Curt Sachs. A classification of musical instruments. *Galpin Society Journal*, 14:3–29, 1961.

[66] David Wessel and Matthew Wright. Problems and prospects for intimate musical control of computers. In *ACM CHI Workshop in New Interfaces for Musical Expression (NIME)*, April 2001.

[67] Yamaha WX5 wind controller. http://www.yamaha.com/yamahavgn/CDA/ContentDetail/ModelSeriesDetail/0,6373,CNTID%253D1040%2526CTID%253D208500,00.html, 2005.

[68] T. Winkler. Making motion musical: Gestural mapping strategies for interactive computer music. In *Proceedings of the International Computer Music Conference (ICMC '95)*, pages 261–264, Banff, Canada, 1995.

APPENDIX A

USER TESTING SYSTEM DETAILS

This appendix provides additional detail on the system used to evaluate MetaMuse. It also provides specific information about the questionnaire used.

A.1 System configuration

The user testing experiments are set up with two computers: the primary one for the subject and another the experimenter. The primary computer hosts MetaMuse and the parametric interfaces, along with a small-footprint server to launch and close the interfaces. The secondary comptuter hosts a controller client that allows the experimenter to control which experiment is currently being run; it also hosts target playback.

Parametric data from the synthesiser is captured from the primary computer as the experiment runs. This data is used to measure the subjects' performance after each trial, as discussed in Chapter 5. It is also logged to allow review and statistical analysis to be performed after the experiments.

The system architecture and data logging for the experiments are described in the following sections.

A.1.1 System architecture

The configuration GUI for MetaMuse, described in Section 4.3.1, is reused as the server application on the primary computer. That section describes the Tcl client/server library that is used for communication between the configuraiton GUI and the main application. The same library is used to set up the GUI as a server which can launch the parametric GUIs as client subprocesses. Similarly, the controller client on the secondary computer connects to the configuration GUI as a client.

Commands to launch and close MetaMuse and the parametric interfaces originate at the controller and are sent to the server. This allows the experimenter to script the experiment, presenting the appropriate interface for a set of targets. The scripts specify which interface to present, along with a series of targets. The experimenter can load a script from the controller interface to start an experiment.

Target sounds are produced on the secondary computer to reduce processor load on the primary machine. This is important in light of the computational restrictions on the synthesiser, discussed in Section 4.3.3. Additionally, the subject's sound output can be recorded by the secondary computer for future analysis. This provides the ability to run experiments using post-trial judging, but for the experiments described in this research such recordings were not employed.

The target sound is routed to the primary computer by way of the sound card line in. There, it is multiplexed with the synthesiser output using a software mixer. The subject presses a button to switch between the two sounds, and visual feedback is given by the interface currently in use. In MetaMuse, the droplets displayed on screen change colour from blue to dull grey; in the compartor interfaces a button widget toggles its colour and text.

Subjects also press a button to indicate when they have reached each target to their satisfaction. Data logging and performance measurement is performed at this time and is described in the following section.

A.1.2 Data capture

As the subject performs a task, the synthesiser's data is stored for performance measurement after the trial and further analysis after the experiment. As discussed in Section 4.3.2, a set of parameters is used to instantiate each droplet the synthesiser produces. This set of parameters, along with the time at which the droplet event occurred, is used for the real-time analysis and saved to file for additional investigation.

Performance measurement involves determining the distance from the subject's sound to the target after the trial is complete. The distance is calculated in Euclidian parameter space with a subset of the total number of parameters: "nObjects" and "reverb". This subset was chosen using the just-noticeable-difference trial described in Section 5.4.2.

Since there are random effects in the droplet creation, measuring the accuracy of the last single droplet introduces random error to the performance measurement. Therefore, the last ten droplets are averaged to determine the subjects' performance. However, more than the last ten droplets are stored in memory during the trial.

To provide analysis options post-experiment, the data from the entire trial is saved if possible. Due to performance concerns, though, this data cannot be logged to a file in real time. Therefore, droplet data is stored in a buffer in memory and saved to file at each trial's completion. The buffer is pre-allocated for performance reasons, and can store data from 6000 droplets. This is long enough to store one minute of data in the worst case scenario, or an expected four to six minutes of normal use. The buffer is circular, allowing the last 6000 droplets to be stored if the trial runs longer than the buffer allows.

A.2 Questionnaire

The purpose of the questionnaire is to determine the expressive qualities of the MetaMuse interface, on its own and as compared to the parametric interfaces. It also had a secondary purpose: to gather any demographic information that may have a bearing on the results. The questionnaire is comprised of three parts: a demographic pre-experiment written questionnaire; a written questionnaire soliciting subjects' opinions on their experiences with each interface, which was administered after each task was completed; and a post-experiment verbal interview intended to draw out additional discussion in a semi-structured manner.

The three components are discussed in greater detail.

A.2.1 Demographic questionnaire

The demographic questionnaire is intended to identify possible confounding qualities of individual participants that may make them unsuitable for the experiment. Basic demographic information such as age, gender, and handedness was obtained, as was level of computer literacy. Finally, subjects were asked if they played any musical instruments or had any experience with computer-based musical instruments.

Questions with limited choices presented possible answers, and subjects were instructed to circle the appropriate choice. The questions asked, with their choices and in the order presented, are:

1. With which hand do you normally write? (Left/Right)
2. Have you used a computer before? (Yes/No)
   - (a) How often do you use a computer? (Daily/Weekly/Monthly/ Rarely)
   - (b) Have you used a mouse? (Yes/No)
   - (c) What is your primary use?
3. Do you play a musical instrument? (Yes/No)
   - (a) What type(s)?
   - (b) How do you rate your proficiency? (Novice/Intermediate/ Experienced)
4. Have you used computer-based musical instruments? (Yes/No)

Questions without provided answers had space indicated for subjects to write in their responses. Space was also left at the bottom for comments, though few were expected at this stage of the experiment.

A.2.2 Post-task questionnaire

The post-task questionnaire is designed to solicit feedback related to the user experience and, especially, the expressivity of each interface. The questionnaire was given after the user completed all trials for each interface. The same questions were asked for all interfaces. Subjects were asked to circle a number corresponding to their level of agreement with each of several statements:

1. Disagree strongly
2. Disagree
3. Disagree somewhat
4. Agree somewhat
5. Agree
6. Agree strongly

The questions, in their order of presentation, are:

1. Using this interface was enjoyable.
2. I had adequate control over the sounds I made.
3. The interface was difficult to understand.
4. I felt creative while making rain sounds with this interface.
5. I am satisfied with my performance in this set of tasks.

Once again, space was left after the questions for users to write comments.

This questionnaire is critical to gaining insight into the subjects' experiences with the interfaces. However, it had to be repeated three times – once for each interface – and by necessity takes place after an extended period of concentration. Therefore conscious effort was made to keep the questionnaire as succinct as possible. balance between relatively straightfo

A.2.3 Exit interview

To garner further feedback on the interfaces and, more generally, the experiment itself, a oral interview was conducted after each subject had completed all tasks. The interview was intentionally informal, allowing subjects to relax and discuss their experiences without having to conform to specific wording or structure that may be imposed by a formal interview. Interviews were recorded with a standard "shoebox" recorder.

To ensure that certain specific aspects of the experience were explored, the experimenter had a list of questions which acted as a guide for topics of conversation. Subjects were encouraged to talk as much or as little as they liked on each topic, and conversation was allowed to flow naturally from one topic to another when possible. Topics were generally discussed in the order below, with the earlier topics generally taking up the bulk of the interview's duration.

The guiding questions, along with some discussion on why the questions were asked and how they were introduced during the intervew, were:

- **What was the most enjoyable part of the test?** The intention of this question was to solicit which *aspect* of the test was most enjoyable, not necessarily which *interface*. This distinction was made clear when the topic was raised. The expectation was that certain parts of the test, such as the act of control or the pleasure of exploring the sounds, would be mentioned.

- **What was the easiest part of the test? The hardest?** Again, this question's intent is not to rate interfaces, but aspects of the experiment.
- **Which was your favourite interface? Why? How about your least favourite?** Unlike the previous two questions, this question specifically deals with the subjects' preferences in interface.
- **How expressive do you feel rain sounds are? Do you feel this was a valid experiment for expression?** An underlying assumption in MetaMuse is that rain sounds are expressive. This question aimed to validate that assumption.
- **How accurate was the sound produced by the rain synthesiser?** In Section 4.3.2 we discussed some of the limits of the synthesiser. This question ties in with the previous one to determine the validity of the experiment. In order to compare expressivity of interfaces, we must ensure that the interfaces themselves are producing sounds that can be considered expressive. If our target sound space is that of rainfall, we must ensure that we met that target adequately.
- **How would you suggest we improve the synthesiser?** As an extension to the previous question, we hoped to solicit ideas for possible improvements to the sound synthesiser. However, we recognised that some users would not have the background to provide meaningful input to the problem of rain sound synthesis, so the topic was approached obliquely and allowed to pass if the subject was not inclined to discuss it.
- **How would you suggest we improve the prop-based interface?** Whereas the previous question may require some technical knowledge or experience in sound or music production, this question is more related to general usability. We hoped that users would have specific suggestions about the strengths and shortcomings of the MetaMuse interface.
- **Do you have any other suggestions?** This question allowed subjects with other thoughts to voice them.

- **Do you have any questions or concerns you'd like to address?** Finally, we solicited feedback on the process. By raising this topic subjects could voice any concerns about experimental method or, generally, any aspect of the test that confused or concerned them.

APPENDIX B

SOUND SYNTHESIS

B.1 Introduction

The sound synthesis module of MetaMuse determines how the six droplet parameters – dx, dy, dz, vx, vy, and vz – plus frequency of droplets produce sound output. Interaction effects are also possible as droplets fall close together in time. The mapping is constrained by the three metaphors used in the system.

Two techniques were used to create the synthesis module. The first is a sample-based system, implemented in jMax, that uses granular synthesis to produce the sound output. It was found that the use of pre-recorded samples made it difficult to smoothly vary the sound with landscape position, so a second paradigm was explored. Based on a liberal interpretation of the concept of granular synthesis, the second technique uses a parametric synthesizer to dynamically create the individual granules. It proved to be more successful than the sample-based version and is the technique used in the evaluated system.

B.1.1 Parameter Mapping

Of the three system metaphors, two constrain the sound output of the system. The rainfall metaphor implies that each droplet creates a distinct sound. The landscape metaphor provides several constraints on how droplets produce sound, the primary of which is that the position of the droplet on the landscape, given by dx and dy, determines the quality of the sound produced. These constraints and some others are summed up in Table B.1.

Within these constraints there is latitude. For example, while the quality of a droplet changes with (dx, dy) position, how it changes is left to interpretation. As discussed in Section 5.4.1, the under-constrained nature of the metaphor was problematic in that users couldn't easily predict how the sound would change with position.

Parameter	Effect
dx, dy	quality/timbre
–	distinctness
v	energy
frequency	blurring, volume, bass

Table B.1: Overview of the mapping from droplet parameters to audible qualities. Distinctness is a result of the discrete nature of droplet initiation, so doesn't have an explicit droplet parameter.

B.1.2 Two Methods for Synthesis

The two methods used for synthesis were granular synthesis, then parametric granular synthesis. The former was first controlled by specifying 64 square regions on the landscape. Each region was assigned a pre-recorded sample, so the user could choose which sample to play. However, the discrete nature of this method was undesirable as it didn't match the continuous landscape metaphor. An interim solution was to position three droplets on the rim as discussed in Section 4.3.4 and volume-mix between them. However, the blending of three distinct rain sounds does not produce a single rain sound, so a parametric synthesiser based on STK was used instead.

B.1.3 Other Mapping Parameters

We considered many other ways in which to interpret the seven parameters. The tilt of the landscape, indicated by the off-vertical velocity of the droplets, can be used for parameter control. dz, the distance into the landscape that the droplet has penetrated during the update cycle in which it is detected, is a function of update rate and can be used to create randomness in the sound output.

These controls were tested briefly during prototyping and were found undesirable. Tilt control (over pitch, in the case of our testing) breaks the metaphor – rain is not perceived to sound different on varying slopes. Adding randomness using dz makes droplets inconsistent – droplets coming from the same controller positions may sound different, depending on the microsecond differences of their timing. This inconsistency is undesirable.

Ultimately, these variations proved undesirable in MetaMuse and were discarded. Since their implementation was trivial and they were tested on very early prototypes they will not be discussed further herein.

B.2 Granular Synthesis in jMax

The granular synthesis engine was built in jMax [12]. It is based on "voxels," or voice elements, reading samples from sample tables. As several voxels read samples in round robin fashion, multiple playback channels can be combined to create the overlapping sound of granular synthesis. Since this method is described by Truax [62] and is no longer used by MetaMuse, it will not be discussed further. However, the two mappings are of interest.

The first mapping used was a discrete mapping based on an 8x8 grid superimposed on the landscape. Each grid position was assigned a sample from the table, and each time a droplet hit that position the respective sample would play. The samples themselves were created by recording the sound of water dripping onto a wide variety of surfaces, from tissue to foliage to stone.

While the discrete sample selection technique worked as a simplistic controller, the discrete nature made it very difficult to create meaningful sounds. The controller may as well have been a 64-key keyboard. Of particular note was the awkward transition between grid positions. Since the stream of droplets had some width to it, the playback would overlap two or more unrelated samples, giving the device a very disjoint sound.

We experimented with a simplistic modification to alleviate this problem. Instead of assigning samples to grid positions, we positioned samples on three equidistant points on the rim of the landscape. When a droplet landed, its distance from each sample position would determine that sample's volume, so that the three samples were mixed together. While this proved less disjoint than the discrete mapping, the sample mixing was not a good solution. Instead of sounding like one droplet, it sounded like three. We turned to parametric synthesis to resolve this problem, ultimately dispensing with jMax entirely and moving to a solution based on the Synth Toolkit.

B.3 Use of the Synth Toolkit

The second synthesis module, and that which was ultimately used, is a real-time resonance filter bank based on the Synth Toolkit (STK). STK provides a variety of synthesis methods. Of interest to us is the Shakers class of instruments, which provides the parametric resonance model we used.

Shakers uses a set of random processes throughout the model. In particular, the excitation function is a random function, which was not suitable for our purposes. To enable expression a consistent sound must be available.

Many parameters, such as the number of resonant frequencies and their values, are encapsulated in the instruments in Shakers. For example, the Maracas instrument uses 4 different centre frequencies, while other instruments may have more or less centre frequencies of different values. To change the timbre of the instrument, STK must reinitialize it to new values. We created a new instrument that provided real-time access to these parameters. This enables MetaMuse to change the timbre of the instrument while playing.

The architecture of the parametric synthesis module has two important features. First, multiple instruments are instantiated and iterated in parallel, their outputs summed. This allows multiple droplets to sound concurrently, which is important for the granular layering effect required for the instrument. Second, the instruments are iterated in a single, separate thread. A condition variable and a protected, shared memory block are used to pass information from the physics model.

The mapping from droplet parameters to synthesis parameters follows the same principles as the sample-based synthesizer. In short, droplet velocity is mapped to filter excitation energy, while position maps to centre frequencies and timbral parameters. The mapping uses three pre-defined parameter sets on the circumference of the landscape and interpolates between them for positional mapping as discussed in Section B.2.

The system described thus far exceeded the processing power available to us when more than 10-15 instruments were instantiated. This limited the number of concurrent droplets the system could support. Some heuristic methods were used to improve the sound quality and simulate greater droplet numbers. A running estimate of the number of concurrent droplets is calculated. This value is used to adjust the overall volume of the system. It also affects the gain on a low-frequency resonator to fill in the sound. Finally, at greater droplet numbers a low-pass filter is used to blend the droplets together.

The base STK Shakers class, our modifications to it, and the system architecture, mapping, and heuristics will be discussed in greater detail in the following sections.

B.3.1 STK: a foundation for real-time synthesis

The Synth Toolkit is a freely available package implementing a variety of methods for real-time synthesis. The Toolkit consists of the source code for a hierarchy of C++ classes for synthesis methods, along with a supporting set of classes performing incidental tasks such as sound output. STK also includes a controller program that implements SKINI , a variation of MIDI providing a familiar message structure for synthesizer control. Of interest to us are the PhISEM (Physically Informed Stochastic Event Modelling) class and the supporting classes implementing reverberation and sound output. For details on the class structure, SKINI message set, and other available classes, refer to the STK documentation.

The following subsections will present the salient details of PhISEM architecture, relevant parameters, and the supporting classes used.

B.3.2 PhISEM Architecture

Figure B.1 shows the basic architecture of the PhISEM system. The main feature is a bank of resonant filters. This is iterated once for each sample, with a decaying, randomized input energy. The output is averaged over three samples to low-pass filter the sound.

The input filter bank is a stochastically varying, decaying system energy value. The system energy also is affected by an input energy parameter, which represents the excitation energy of the instrument. Both the input energy and the system energy are persistent but exponentially decaying values.

The addition of the input energy to the system energy is governed by the first of three random processes. The system energy is increased by the input energy statistically infrequently – on the order of one to ten times per thousand ticks. The random addition of the input energy to the system is what drives the instrument, and when the input dies to zero the system does likewise.

The second random process creates a random signal between the system energy and the equivalent negative value. It does so by creating a random multiplicative factor from -1.0 to 1.0. This signal is more influential in creating the stochastic process which models hard collision sounds for PhISEM, as it is directly input into the filter bank. Note that the randomized system energy does not vary the system energy parameter itself.

The third random process causes the filter bank itself to change slightly.

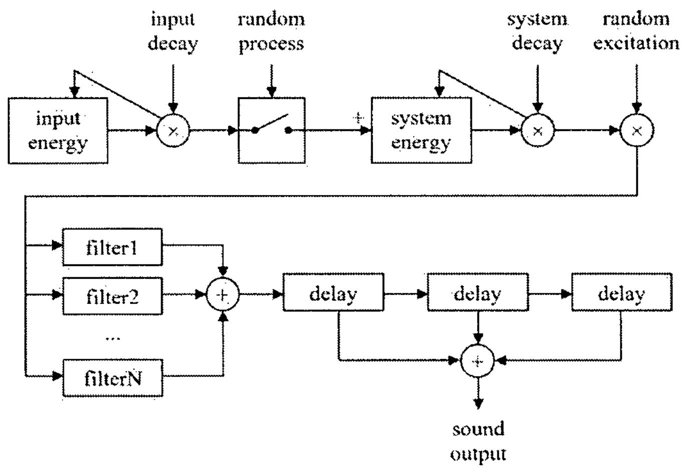


Figure B.1: The sound generation process. The input energy is set to an initial value when a droplet is initiated, and decays with each iteration. Based on a random process, the input energy is sometimes added to the system energy, which decays similarly. A random excitation process perturbs the system energy before it is input to the resonant filter bank. Finally, the filter outputs are fed into a delay line and three delays are summed to form the sound output. For MetaMuse, the random processes were replaced with pre-recorded sequences to ensure consistency across droplets.

Fixed	Mutable
system gain	input energy
filter centre frequencies	(system energy)
filter resonances	input decay
freq_rand	system decay
nfreqs	nObjects
decay_scale	resonant frequency (mod wheel)

Table B.2: Parameters in the PhISEM Instrument. Those in the left column cannot be changed at runtime; those in the right can.

Each filter has a centre frequency which is perturbed slightly whenever the input energy is added to the system.

The next section discusses the specific parameters of the PhISEM model and how they are used to predefine instruments.

PhISEM Parameters and Instrument Definitions

The PhISEM Instrument contains several parameters, some of which are fixed on initialization and others which are mutable. These parameters are listed in Table B.2.

The parameters in Table B.2 affect different components of the PhISEM system. These parameters are described as follows. Note that these parameters are described to be internally consistent with this document; some parameter names may differ from the STK documentation.

Input energy (mutable): This is the amount of energy being added to the system. Each tick, it decays by a factor *input decay.* Occasionally it is added to *system decay* – see *nObjects* for details on this random process.

System energy (mutable but internal): This is the amount of energy IN the system. While it is mutable, it cannot be directly affected; instead it is indirectly affected by *input energy.* Each tick, *system energy* decays by *system decay.* The *system energy* affects the perceived strength of the collision in the instrument.

input decay (mutable): A multiplicand for *input energy. Input decay* affects how quickly the *input energy* exponentially dies down to zero.

Typical values are 0.995 to 0.9999.

system decay (mutable): A multiplicand for *system energy. system decay* affects how quickly the *system energy* dies down to zero. Typical values are 0.95-0.97.

nObjects (mutable): Affects how efficiently *input energy* is transferred to *system energy.* Specifically, at each tick *nObjects* is compared to a random number from 0 to 1023. If the random number is less than *nObjects*, *system energy* is increased by *input energy. nObjects* typically ranges from slightly more than 1 to 512.

nfreq (fixed): Describes the number of resonant frequencies modeled. Ranges from 1 to 6 or 7.

centre_frequencies (fixed): Describe the centre frequencies of the filter's *nfreq* resonant frequencies. Typical frequencies range from a few hundred to nearly 10000Hz.

Resonances (fixed): Specify the strength with which the *nfreq centre_frequencies* resonate. Typical values are 0.995 to 0.998.

freq_rand (fixed): As discussed above, the centre frequencies are occasionally re-calculated with slight perturbations. *nObjects* determines how frequently this occurs. *freq_rand* determines how much perturbation can occur to each centre frequency. Ranges from 0 to 0.2.

Resonant frequency (mutable): Provides indirect access to the centre frequencies. Like a pitch bend, this parameter is intended for control by the mod wheel on a synth keyboard. It causes all centre frequencies to increase or decrease slightly. This parameter accepts standard MIDI values from 0 to 128.

system gain (fixed): A multiplier used when recalculating the *gains.* Can range from 0.1 to 60 or more.

gains (indirectly mutable): *nfreq* gains are multiplied by the resonator outputs before they are summed at the output of the filter bank. The *gains* depend on *nObjects* and *system_gain*, and there is no way to change an individual gain separately from the others.

decay_scale (fixed): We do not use this parameter.

The fixed parameters of an Instrument, set on initialization, affect the basic timbre of that instrument. STK defines several parameter sets for PhISEM Instruments called "Shakers". Shakers Instruments include a Maraca, a Sekere, a Casaba, bamboo wind chimes, and so on. These Instruments predefine values for *system_decay*, *input_decay*, *system_gain*, *nObjects*, *nfreq*, and *centre_frequencies* and *resonances*.

Supporting Classes for STK

Two of STK's supporting classes are used in MetaMuse: Reverb and WvOut. These implement a reverberating effect and an interface to the sound driver, respectively. The two supporting classes connect to the Instrument created by the PhISEM class as shown below. This configuration creates a pipeline wherein sound samples are created in the Instrument, reverberated in the Reverb, and output in the WvOut.

```
--------------       ----------       ---------
| Instrument |  ->   | Reverb |  ->   | WvOut |
--------------       ----------       ---------
```

The Reverb class implements a single delay line with a feedback loop. The length of the line is variable, creating an echo effect that falls behind the original sound. A mutable parameter, *reverb*, allows the amount of reverberation to be changed at runtime.

The WvOut class implements several methods for outputting the samples making up the waveform. MetaMuse makes use of the real-time output, which passes the samples through to the sound card for immediate playback. Other output methods, such as output to WAV file, are also available. WvOut sets up its output stream(s) on initialization.

Reverb and WvOut are used as-is in MetaMuse. The PhISEM class, however, is changed considerably to create a synthesizer for MetaMuse. The following section details how PhISEM is changed.

B.3.3 Modifications to STK Shakers

STK's Shakers required three changes to make it viable for use in MetaMuse. First, the randomness had to be removed from the tick function to allow for consistent droplet sounds. Second, the excitation function was replaced with a water-droplet excitation function. This pre-recorded sample series provides a more appropriate sound than the existing random excitation. Third, and most complex in its implementation, was providing real-time access to the fixed Instrument parameters such as centre frequency and resonances. This allows the Instrument to create different sounds from one droplet to the next. The three changes made will be discussed in greater detail.

Removing Randomness from STK Shakers

Randomness is introduced in three places in the PhISEM class, shown in Figure B.1. The first is the random value compared to *nObjects*, used to determine when to increase the system energy by the input energy and recalculate the filter coefficients. The second is the noise_tick function at the input to the filter bank. The third is the random value, controlled by freq_rand, that gives minor perturbations to the centre frequencies.

Of the three, the third is the simplest to remove. By ensuring freq_rand is always 0, the random component of the centre frequencies is always 0 and the centre frequencies remain constant.

The first random value, that associated with *nObjects*, is more difficult to handle. It is not possible to simply replace the random value with a constant one. If the constant were less than *nObjects*, no energy would enter the system; if greater, too much energy and the filter would saturate. To produce a repeatable random sequence, we replaced the random function with one that reads numbers from a file and returns them in order.

The function relies on a file containing a sequence of random numbers from 0 to 1, as would normally be returned by rand(). The first time it is called, the random function initializes an array of numbers and populates the array from the file. It then returns the next array value with each subsequent call.

The second random function, noise_tick, creates an excitation function for the filter. The excitation function is discussed in the following section.

Changing the Excitation Function

As discussed in the previous section, one of the random functions, rand(), used in Shakers acts as an excitation function. Since Shakers is intended to model solid objects striking one another, it is appropriate to replace this function with one that will synthesize a water droplet. We recorded a droplet landing on a non-resonant surface. We implemented a new rand() function that takes its values from those samples.

To record the excitation function of a water droplet, we required a surface that had minimal intrinsic resonance. We decided to use a light tissue paper and record the sound of a water droplet landing on it. Fortunately we already had such a recording from our grid-based sample selection discussed in Section B.2.

Once the samples were recorded, they had to be returned in correct sequence from the new rand() function. This was accomplished by keeping static variables in the rand() function. The sequence of samples was stored, as well as an index to the current sample. Each time the function is called, the current value is returned and the index incremented. The index is rolled over each time the array bounds are exceeded, though the sequence is zero-padded to allow enough time for the system energy to die out before this happens. The counter can also be reset, allowing each new droplet to reset the excitation function to the beginning.

Before it can be used, the sample array must be initialized from a file. There are two points in the code where this can be done: in the Instrument constructor or on the first call to rand(). The first is preferable to provide consistent execution time for every call to rand(), but requires a significant change to the Instrument class. In the interests of minimizing the footprint of our modifications, we initialize the sample array on the first call to rand(). Our controller program ensures rand() initializes by initiating one droplet on startup.

Accessing Internal Parameters

In Section B.3.2 we discussed the parameters that define a PhISEM Instrument and mentioned some of the predefined Instruments included in the Shakers class. Having removed the randomness from the system and created an excitation function appropriate for a droplet of water landing on a surface, we next needed to model the resonant characteristics of the surface itself. This represents a significant departure from an instrument such as a Maraca, where the same beads are striking the same surface with every shake. With MetaMuse, we require that the surface itself from droplet to droplet, in accordance with the landscape metaphor. This requires access to the parameters that, for other Instruments, are fixed: freq_rand, centre_frequencies, nfreq, nObjects, input_decay, system_decay, decay_scale, and base_gain.

Since the Instrument handling droplets works differently from the other Instruments, having different random functions and tick functions, we defined a new Instrument in the Shakers class. The Generic Instrument defines parameter values and allocates space for eight centre frequencies, allowing nfreq to vary from 1 to 8 with no reallocation concerns. The other previously-fixed parameters are also made accessible, but no further innovation is required to allow that access.

The Generic Instrument is successful – it allows changes to the resonant characteristics of the modeled surface through its parameters at runtime. However, testing showed that it is best to change the previously-fixed parameters when the filter has died down to 0 energy. Changing the parameters at high energy sometimes results in instability in the filter. This is suitable for MetaMuse; as we shall see in Section B.3.4, MetaMuse sets up each droplet's parameters, energizes the droplet, and lets it die down before repeating.

B.3.4 System Architecture

The parametric synthesizer for MetaMuse consists of multiple Generic Instruments iterated in parallel. The Instruments are "fixed" in round-robin fashion, allowing multiple droplets to overlap without saturating the filter of any one Instrument. The Instruments' outputs are summed and passed through a Reverb to a real-time WvOut. This pipeline is iterated in a separate thread from the other components of MetaMuse, with signalling taking place through a shared memory segment and a condition variable. The round-robin Instrument pool and inter-thread signalling methods will be discussed in greater detail.

Multiple Instruments and Performance Considerations

The use of multiple instruments in the synthesizer module allows multiple droplets to sound concurrently. The droplets are iterated in turn and their outputs summed. The number of Instruments to iterate is constrained from above by the amount of processing power available, and from below by the desire for many concurrent droplets. A tradeoff was made that allows the bank of Instruments to iterate in a single thread; this thread occupies one processor in a dual processor system.

Each tick of the synthesizer sends a single sound sample to the sound card. This process entails ticking each Instrument individually, then using each Instrument's output to tick its respective Reverb. Finally, the Reverb outputs are summed to provide a sample for output through the WvOut. At a sample rate of 22050 Hz, and with N instruments operating in unison, the MetaMuse system must be capable of ticking an Instrument 22050*N times a second. Clearly, then, the hardware used provides a maximum bound on N.

The lower constraint on the number of Instruments is provided by the number of droplets we wish to have sound in unison. We found that a single droplet dies out in about 100 ms. Therefore, with N Instruments we can provide a maximum droplet frequency of 10N droplets per second.

We run MetaMuse on a dual-processor P4 operating at 2GHz. With the synthesizer module running in a single thread, we were constrained by the power available from one processor; indeed, with the system appropriately tuned we find that the synthesizer thread fully occupies one processor, all other operations being handled by the other. In this setup, we were able to iterate 15 Instruments before the sound quality deteriorated due to dropped samples. We reduced this number to 10 for our final system, allowing further overhead for inter-thread communication.

10 Instruments provides a maximum Droplet frequency of 100 droplets per second. This turns out to be adequate for a moderate rain-shower. We discuss ways in which the sound is further improved in Section B.3.5.

Droplet allocator and inter-thread communication

The droplet allocator is responsible for keeping track of which Instrument is next in line to receive an incoming droplet. To aid in this task, a Droplet struct is defined containing all the necessary parameters for initiating a droplet. The new Droplet is placed in shared memory by the mapper, which then waits on a condition variable for the allocator to receive the Droplet. Once received, the allocator sets up the droplet parameters in the appropriate Instrument, increases the input energy of the Instrument to the required level, and continues ticking all the Instruments.

New Droplets are sent to each Instrument in turn. However, the allocator does not keep track of how recently an instrument was last used. If the Droplets arrive too quickly, previous droplets may be overwritten. The calling module is responsible for ensuring that droplets are not sent too frequently.

In MetaMuse this is done by throttling back the number of droplets produced when the watering can is at full tilt.

B.3.5 Improving the sound with heuristic methods

The synthesiser described thus far works reasonably well for single droplets, but sounds inaccurate when used to approximate a fuller rain shower or rain storm. We believe this to be because of the sheer magnitude of a rain storm – at any given time the listener may be hearing thousands of concurrent droplets, with millions of droplets per second falling in audible range. As discussed in Section B.3.4, MetaMuse is capable of a maximum of 10 concurrent droplets and about 100 per second. Additional effects at the large scale may include surface water, secondary splashes, cavitation, and droplets interfering with one another on impact. We cannot hope to duplicate these effects, but we can imitate them to some extent.

Three heuristic methods are used to improve the sound of the synthesiser: bass frequency fill-in, output low-pass filtering, and volume control. All three of these effects depend on the instantaneous frequency of the synthesiser. More precisely, the number of currently active Instruments is used to control the amount of each of these effects.

The three heuristic effects will be discussed in greater detail.

Bass fill-in

One of the first differences we noted between the MetaMuse output and a recording of a rain storm was that MetaMuse lacked bass. The rainstorm recording had a deep, low-frequency component that filled in the sound. MetaMuse created the impression of several separate droplets rather than one rainstorm.

We attempted to add bass to the system proportional to the instantaneous frequency. We did so by adding two centre frequencies to the four currently used. These additional resonators are set to lower frequencies — on the order of 400 to 800 Hz. They are iterated with the other resonators, but their output is multiplied by a gain of zero before the resonator outputs are summed. As the instantaneous frequency increases, the bass gain is also increased. At full frequency, the bass resonators are at a gain of 1, creating a bass rumble to the sound.

The bass fill-in helps to alleviate the sense of droplet separation the listener feels. By filling in the lower frequencies, a better sense of depth is introduced to the sound. However, the droplets are still too infrequent to completely fill in the sound; the bass is heard as a series of separate sounds rather than as the continuous rumble perceived in the rain storm recording. The following section discusses a second heuristic method — low-pass filtering — that improves on this situation.

Low-pass filtering and volume control

MetaMuse performs low-pass filtering when there are many concurrent droplets to help blend the droplet sounds together. As with the bass fill-in, the instantaneous frequency is used to determine the amount of filtering to do. The filtering itself is performed by a biquad filter that is part of the STK package.

The final heuristic method is volume. To further give the impression of more droplets, the volume is increased with the number of instantaneous droplets.

APPENDIX C

IMPLEMENTATION OF POLHEMUS LIBRARY

The Polhemus Library is a multi-threaded library for simplifying access to the Polhemus Fastrak. The Fastrak is a magnetic sensor that is attached as a peripheral on the serial port. The Polhemus Library, and the supporting Serial Port Library, provide layers of abstraction to facilitate use of the device.

The Polhemus Library wraps the basic functionality of the Fastrak into a cohesive, easy-to-use set of function calls. Similarly, the Serial Port Library provides a cohesive interface to the computer's serial port, abstracting away the platform-specific details of opening, closing, and communicating on the port. These libraries will be discussed in greater detail in the following sections.

C.1 Serial Port Library

The Serial Port Library provides a unified interface to the serial port across multiple platforms. It allows changes to the underlying port calls, such as open and close, without affecting the operation of the overlying device library. It does not provide access to advanced serial port control functions such as ioctl calls. A serialPort structure is defined in the library and contains the basic information required for operation. The use of this wrapping function is evidenced by a recent extension of the library to provide similar functionality for the Universal Serial Bus (USB), which has a different set of controlling functions. One of the benefits of this abstraction is that a USB-based Fastrak, if there was one, could be used in place of the serial port model now simply by relinking with a different Serial Port library.

The Serial Port Library is simply a wrapper for the primary functions of the serial port: open, close, read, and write. The relevant information for the port, currently a name and the file number, are stored in a structure that is used by the calling function. The contents of the structure are extracted as needed and passed with the function's arguments to the appropriate system calls. Since the library only provides access to the basic functions of open, close, read, and write, the serialPort structure is left unprotected to allow more complex control calls. However, this functionality is not used by the Polhemus library.

C.2 Polhemus Library

The Polhemus Library provides a standardized interface to the Polhemus Fastrak through the serial port. It handles the parsing of data from the Fastrak into records and provides access to those records through three modes: by polling, callbacks, and synchronous waiting. The interface is standardised in that it can be similarly applied to other serial devices. The implementation is multi-threaded, allowing synchronous or asynchronous communication. We will discuss the three input methods and their implementation in the following sections.

C.2.1 Input Methods

The three input methods are polling, callbacks, and synchronous waiting. The basic operation of each of these methods, with the Fastrak in streaming mode and using nonblocking reads for a single record, is as follows.

Polling When records are polled, no indication is given to the application when new records are received. Instead, the application simply reads a record periodically. If a new record is available, it is returned, otherwise the read call returns an error code. Polling is most useful in situations where the timing of the record retrieval is very important, such as in limited-resource applications.

Callbacks With the callback method, the application registers a callback function to be run whenever a new record is available. The function is run in a separate thread so it doesn't interrupt the parsing thread or the application itself. Callbacks allow data to be updated "behind the scenes," without interfering with the application. For example, a graphics application may use callbacks to update the positions of objects, which are then rendered by the main application thread.

Synchronous Waiting Synchronous waiting, based on the *select* system call, is used when the application wishes to wait for one or more file descriptors to be ready to read[1]. When this method is in use, the application selects on a file descriptor that points to the receive end of a pipe. The pipe is signalled whenever a record is ready to be read, allowing the application to handle records as they arrive. The select system call is often used when multiple input methods, such as the Fastrak, the keyboard, and possibly other devices, are used concurrently.

More advanced operation of these methods occurs is possible. The Fastrak can be used in non-streaming mode, with blocking reads, or with multiple records to account for multiple sensors. These concerns are discussed in the implementation sections below as each is introduced.

C.2.2 Common Implementation

The three input methods discussed above share a common implementation, with extensions for the additional requirements of each method. The common implementation is discussed in this section, followed by the extended implementation for the three methods.

The common implementation of the Polhemus library centres around a pair of structures, one used to hold record data and one which stores runtime variables, and a thread dedicated to parsing data as it arrives from the Fastrak. The first structure, called a phRecord, contains space for all data that can be received from the Polhemus in one update cycle. The second is called a Polhemus, and stores all necessary structures for the operation of the library, including thread pointers, flags, mutexes, condition variables, and a pointer to a phRecord. The parsing thread separates the data parsing from the flow of the application.

The phRecord structure

The implementation of the phRecord involves an important concept of the Polhemus library: the record set. The Polhemus Fastrak can produce up to four records in an update cycle, one for each sensor. In addition, it can produce messages relating to its status or to internal errors. To provide access to all of these in a given cycle, the phRecord structure contains space

[1]Other conditions are also usable with select. See the system manual pages.

to store four data records as well as one message. When a record is read, the calling application must specify which record or records to read from within the phRecord.

This design has ramifications in the implementation of both callbacks and the select function call. In both those input methods, an action occurs when a record is received. To allow the specification of a single action for a set of several data records, the concept of a record set is defined. Callbacks can be registered, and select call file descriptors requested, for any of the individual data or message records, or for the set of data records for all active sensors. This implementation allows one action to be performed when records are received for several sensors. Finally, when the record set is used, any message is viewed as completion of the set — this allows error messages to be passed on immediately regardless of the completion status of the data records.

The implementation of individual and set callbacks and file descriptors is discussed in further detail in the callback and select system call sections below, respectively.

The Polhemus structure

The Polhemus structure maintains all the runtime information required by the library. This includes the underlying serialPort pointer, the thread pointer for the parsing thread, and various flags, mutexes, and condition variables. It also includes additional data elements required for callbacks and the select system call; these elements are discussed in the appropriate sections below.

A Polhemus variable is created upon initialization of the Polhemus Fastrak using phInitialize. A pointer to the variable is returned by phInitialize, and is then passed to all other functions called by the application. This allows all the Polhemus data to be kept in one place. Also, providing the application with all the Polhemus data allows nonstandard operations to be performed. For example, the application can configure the Fastrak to provide data in inches rather than centimeters, or to provide a different set of Euler angles for the orientation, by calling spWrite, the serial port write function, on the Polhemus' Port variable. The only restriction to this configuration is that the number of bytes per record, and the number of records in a set, cannot be changed by the application.

Conceptually, the most important element in the Polhemus structure is the thread pointer for the parsing thread. Without the use of a separate thread to parse the Fastrak data as it arrives on the serial port, the application program would be required to pause periodically to carry out that parsing. Failure for the application to do so frequently enough could result in buffer overruns on the serial port, and thence lost records. The use of a separate thread for parsing the Fastrak data frees the application from having to consider the parsing task. Instead, the operating system's thread scheduler runs the parser often enough to read all the serial data under normal system load.

The use of a separate thread necessitates measures to ensure synchronization between threads and protection of shared memory. Condition synchronization, in the form of pthread_cond_t and pthread_mutex_t variables, is used to signal the blocking read call, phGetBlocking, that a phRecord is ready. Condition synchronization is also used in the implementation of callbacks, which will be discussed below. Locks, in the form of pthread_mutex_t variables, are used to protect shared memory in the case of the parsing thread itself, the phRecord, and a status flag. All these pthread_cond_t and pthread_mutex_t variables are stored in the Polhemus structure.

The parsing thread

The parsing thread continually parses data from the serial port into records, then signals the application as appropriate when full records are accumulated. A state machine is used in the parsing algorithm, with the data being examined to determine its content at the beginning of each record. Also important in the implementation of the parsing thread are thread synchronization and thread initialization and termination.

Data is captured into a buffer, which is then parsed according to the parsing algorithm. The parsing algorithm is based on a state machine. With no data parsed, the machine is in the COMMAND state. If the first byte is '0', the incoming record is a data record and the next byte indicates to which sensor it belongs. This byte is read in the STATION state. The third byte, is a status code, read in the STATUS_CODE state, then 24 bytes of data, providing six four-digit numbers for position and orientation of the sensor, are received in the DATA state. A carriage return (CR state) and a line feed (LF state) complete the record, and the parsing algorithm returns to the COMMAND state for the beginning of the next record. If the first byte is other than '0', the incoming record is a message and is parsed in the ACCUM state until a newline is received. In any state, if the buffer is emptied the algorithm will read further data from the port to complete the current record.

The parsing thread is initialized and forked in the phInitialize function call. The pthread_t variable is stored in the Polhemus object, which is then passed to pthread_create as the argument to the thread's function. A keep-alive thread, protected by a mutex, is used to determine when the thread should stop parsing and exit.

Other functions

There are several common functions used throughout the library, independent of the method of access used. These are phInitialize, phClose, phStreamOn, phStreamOff, phRequestRecord, phGetNonBlocking, and phGetBlocking. phInitialize was discussed in the previous section; the remainder will be discussed here briefly.

phClose Signals all threads to terminate, joins with them, deallocates memory space, closes the sPort, and exits.

phStreamOn and phStreamOff Send command to turn data streaming on or off.

phRequestRecord Used when streaming is turned off, this function sends a command to the Fastrak to request a single record set.

phGetNonBlocking Gets a new parsed record; returns an error if no new record is available. The calling application includes indication of which records (one or more data records, the message record, or the full record set) to get. Records are copied from the phRecord in the Polhemus structure to a phRecord passed in by the calling application. A lock is used to protect the Polhemus' phRecord from simultaneous access. phGetNonBlocking is used in all MetaMuse functions.

phGetBlocking Gets a new parsed record; blocks until a new record is available. This function otherwise operates the same as phGetNonBlocking. However, in the current implementation of the Polhemus library, this function is not implemented.

In both phGetNonBlocking and phGetBlocking, only new, unread records are returned. A status flag is used to keep track of whether a record has been read or not; the flag is set when a record is read and cleared when the record is written by the parsing algorithm. This presents a limitation to the library: only one application can read data from a Polhemus with this library. If multiple applications try to read, the first will cause the status flag to be set, and all others will receive an error, or block, indicating no new record is available. For applications that use the Polhemus Fastrak data in multiple places, or for situations where multiple applications use the data, it must be read in only one place and shared amongst the other modules or applications from there.

C.2.3 Polling Implementation

As discussed above, the polling method is used when the application simply reads a record periodically; no notification is given to the application when new records are received. As such no specific implementation is required for use of polling. The application either uses streaming mode, set with phStreamOn, or periodically requests records using phRequestRecord, then gets records with phGetBlocking or phGetNonBlocking.

C.2.4 Select Implementation

The use of the select system call requires the application to have one or more file descriptors. The application is then signalled by being made ready to read; this causes the select call to return with an indication of which file descriptor is ready. File descriptors are created based on pipes, and are accessed by the application with the phGetFD and phFreeFD functions.

To set up select call use, the application calls phGetFD, indicating for which records file descriptors are required. File descriptors can be issued for any of the four data records, the message record, or for the full record set. Though there are no checks to prevent the application from getting file descriptors for both the full record set and for some subset, the behaviour of the library in this case is undefined. File descriptors can be released, either to convert to a different input method or to change from, for example, a full record set to some subset, using phFreeFD.

When phGetFD is called, the library creates a pipe for each file descriptor requested. Pipes are created with the pipe call from the unistd system library, which returns a pair of file descriptors: one for reading, one for writing. The readable file descriptor is returned to the application to be added to an FD_SET and used in a select call.

When the parsing thread completes a record, it checks the existence of a relevant file descriptor. If one exists, the thread writes a single byte to the pipe, which puts the readable file descriptor into the 'ready-to-read' state. The application's select call will then return. Note that the pipe is used only for signalling purposes; it does not transmit the record itself. Instead, the application calls either of phGetBlocking or phGetNonBlocking to read the record as usual; both these functions read from the necessary file descriptors to clear the byte and remove the 'ready-to-read' status.

C.2.5 Callback Implementation

The use of callbacks requires one or more functions be registered with the Polhemus library. As with the file descriptors for the select call, callbacks can be registered for any of the individual records or for the full record set (and behaviour is undefined if both are registered). The difference, however, is that a function must be provided by the application for execution as each callback. In addition to the required function definition, a separate thread is created to allow callbacks to be run without interrupting the parsing thread.

The callback function is in the form:

```
void callback\_function( void *polhemus\_pointer,
                         void *callback\_argument )
```

with the polhemus pointer providing access to the Polhemus structure, and the callback argument providing access to a structure defined by the application. The callback argument is represented as a pointer to void to allow any structure to be passed in — the application simply casts the structure to a void pointer when passing it to the library, then casts it back within the callback function itself.

When the application registers its first callback with the Polhemus library using phRegisterCallback, a new thread is forked to run the callback on demand. That thread waits on a condition variable, which is signalled each time a relevant record is received. Note that since only one thread is forked to run all possible callbacks, a lock-protected flag is used to indicate to the thread which callback or callbacks should be run. This operation is confounded, however, if the callback takes longer to run than the refresh period of the Polhemus. Rather than simply signal the callback thread for every record received, the value of the condition variable is tested. The thread is signalled only if it has not already been signalled.

The described operation required one important implementation detail. The flag indicating which callbacks are to be run could be changed during the course of the callback thread's operation. Locking it for the entire duration of the callback or callbacks, however, potentially delays the parsing thread as it attempts to update the flag with the arrival of new records. To prevent both of these issues, the flag is read once by the callback thread immediately after it has been signalled. The value is then stored as a local variable, and only those callbacks that were signalled at that time are run.

Only one callback can be registered per record or record set. Callbacks can be deregistered, either to convert to a different input method or to change from, for example, a full record set to some subset, by registering a NULL callback function. Finally, when the Polhemus is closed with the phClose function, a flag is set to indicate to the callback thread that it should terminate, and the thread is signalled. The thread then terminates and phClose joins with it.

C.2.6 Variations

The Polhemus library is intended to create a template for device interaction, as well as a Polhemus Fastrak interaction library. Libraries with a similar interaction paradigm could be created for any serial device, as well as many other devices such as frame-buffers. Two other libraries are currently in operational development for serial devices other than the Polhemus: one for the Virtual Technologies CyberGlove, a data-glove, and one for the Phidget, a USB-based general-purpose I/O device.

The CyberGlove library differs from the Polhemus library in the number of available records and the size of the record. Records from the CyberGlove have data for all sensors in one record, meaning that multiple records, and therefore multiple callbacks or file descriptors, are not required. As such, the CyberGlove library has a simpler implementation than the Polhemus library. However, other than these simplifications, the CyberGlove library is identical to the Polhemus library.

The Phidget is similar to the CyberGlove in its record structure. However, it uses the USB port rather than the standard RS232 serial port. Since the serial port operation is separated into a different library, however, all that was required to convert the CyberGlove library to use the Phidget was to change the function calls for opening, closing, reading to, and writing from the port. Otherwise the Phidget library is identical to that of the CyberGlove.