The [hid] Toolkit for Pd

Hans-Christoph Steiner

Interactive Telecommunications Program, New York University

Masters Thesis

Table of Contents

1. Abstract

The *[hid] toolkit* is a set of **Pd** objects for designing instruments to harness the real-time interaction made possible by contemporary computer music performance environments. This interaction has been too frequently tied to the keyboard-mouse-monitor model, narrowly constraining the range of possible gestures the performer can use. A multitude of gestural input devices are readily available, making it much easier utilize a broad range of gestures. Consumer Human Interface Devices (HIDs) such as joysticks, tablets, and mice are cheap, and some can be quite good musical controllers, including some that can provide non-auditory feedback. The *[hid] toolkit* provides objects for using the data from these devices and controlling the feedback, as well as objects for mapping the data from these devices to the desired output. Many musicians are using and creating gestural instruments of their own, but the creators rarely develop virtuosity, and these instruments rarely gain wide acceptance due to the expense and skill level needed to build them; this prevents the formation of a body of technique for these new instruments. The *[hid] toolkit* is built in **Pd**, which provides an ideal platform for this work, combining sound and visual synthesis and control with easy access to many external devices for interfacing with humans. **Pd** is a high level programming language, which is relatively easy for novices without major limitations for advanced users. Using consumer HIDs allows musicians to build a shared body of technique, much like video game players have developed; the *[hid] toolkit* enables sharing of instrument patches. In combination, designers of new interfaces for musical expression can design their own instruments while building upon existing skills.

2. Introduction

"*Since the first modular synthesizers, it became possible (for the first time in history) to separate the interface from the actual sound source. It makes sense to try to design new interfaces, new instruments, unbounded by the sound source.*" [1]   - Bert Bongers

With the power that even a cheap laptop can provide, the computer has gained widespread acceptance as musical tool. Composers have been creating music using computers for more than 40 years now, and even music that is played on all analog instruments is generally mixed on computers when recorded. More and more musicians are using computer-based instruments for live performance, to the extent where you can see live computer music in just about any major city in the world. There is a wide range of software available for live computer music performance, such as **Reaktor** [2], **Ableton Live** [3], **SuperCollider** [4], **Max/MSP** [5], and **Pd** [6] designed expressly for this purpose. Though these tools can provide an engaging performance environment, the live performance leaves something to be desired. The audience may be unable to tell whether the performer is actually controlling the music in real time, or just clicking a start button and reading their email. Such performance lacks physicality in the interaction and is quite limited in the range of possible gestures.

Most computer musicians are bound to the standard keyboard/mouse/monitor interaction model. To provide an engaging performance, musicians need to move beyond that interaction model. The human body is capable of a great range of gestures, large and small. Human gestures give off information in a manner similar to language. There are many cultural and some universal human gestures that are well established and easily understood. Music is about expressing certain kinds of ideas. Having the ability to use a broader range of gesture in performance means the performer has broader range of possible expression. Computer musicians should not be limited to the small set of gestures that normal computer use encompasses. In order to physically interact with the computer, input devices are needed. Many of these software environments are already capable of using data from Human Interface Devices (HIDs) such as joysticks, drawing tablets, gamepads, and mice.

Performers of live computer music generally stare at the screen intently while performing. The screen alienates the performer from the audience. What the performer is staring at is obviously important (judging by the intensity of the stare); however the details of what the performer is looking at is almost always completely out of view for the audience. This is in stark contrast to traditional musical instruments, where the instrument is in generally in plain view of the audience. Additionally, the audience is at least somewhat familiar with the mechanisms of the instrument being played. There is potential for using consumer HIDs can alleviate this alienation by allowing the performer to step away from the computer screen. HIDs such as joysticks are objects familiar to those attending computer music concerts. Using HIDs in performance therefore has the potential for making the experience much more understandable to the audience. The performer can come out from behind the computer screen, bringing back a closer connection between audience and performer.

Another element that is missing from the keyboard/mouse/monitor interaction is haptic feedback. "Haptic" is defined as relating to the sense of touch at the skin level and the sense of forces to the muscles and joints. Traditional instruments provide haptic feedback because the interface is producing the sound itself, so the vibrations can be directly felt. Practiced musicians rely heavily on this feedback, often "feeling" mistakes and correcting before hearing them. Computer music performers are obviously using non-auditory feedback during performance, the intensity of their stare at the computer screen is a measure of this. Providing haptic feedback enables the performer to step away from the computer screen and engage more with the audience.

Audio synthesis has freed instrument design from the constraints of the physical method of generating sound, thus desired interface can be mapped to any given synthesis algorithm. For example, the guitar's strings are both the interface and the sound generator, while a MIDI device can control any given synthesizer. This flexibility allows musical instrument designers to choose their physical interface without the constraints of the method of sound generation. Consequently, a multitude of means of translating gestural input from the human body are readily available. By combining such gestural devices with multimedia software, a broad range of people can now make their own computer-based gestural instruments. A new model of instrument design is emerging built upon this idea of flexible interfaces. Instrument designers are shifting from devices that are designed for a broad user base, to general building blocks that allow the individual musician to create their own instrument tailored to their performance goals.

Pd [6], also known as **Pure Data**, is a graphical programming environment for working with MIDI, sound, video, graphics, or anything that can be controlled by computer. It is a real-time system, designed for interactive processing. It provides a fertile platform for designing new instruments, providing a high level, rapid programming environment that is accessible to a wide range of people with varying backgrounds. It is a unified platform for a broad range of activities, combining realtime audio, video synthesis and manipulation, physical modeling. More options exist for data input and output including MIDI, HIDs, and general serial communications. Because **Pd** is free software that runs on most operating systems, musicians with even very limited budgets can build their own computer music instruments. Up until recently, computer music has been out of reach to all but a select few. It is now possible to build an instrument using **Pd** that costs less than most traditional musical instruments, including the cost of the computer.

The main objective of the thesis is to provide an coherent environment for performers to create instruments using gestural interfaces. The *[hid] toolkit* is a toolkit for creating instruments using HIDs (Human Interface Devices). The *[hid] toolkit* provides high-level objects for accessing the data from various HIDs and low-level objects for getting the data directly from the HIDs. It also includes objects for mapping that data to whatever output the user wants to control, and controlling haptic feedback. It is built to an integral part of **Pd**, and most of the *[hid] toolkit* objects are written in **Pd**. For these reasons, **Pd** and the *[hid] toolkit* are an ideal platform for designing computer-based gestural instruments.

3. Concept Overview

The *[hid] toolkit* is a set of **Pd** objects for using a wide variety of HIDs (Human Interface Devices) to build computer music instruments. The aim is to have a standardized and coherent set of objects that enable rapid prototyping of instrument design ideas. Also, the coherent, high-level objects should prove accessible to the novice instrument builder, providing an easy entry point for such users into an otherwise difficult realm. There are four main areas of instrument design: input, mapping, output, and feedback. **Pd** provides a wealth of options for output, the *[hid] toolkit* addresses the processes of getting input data, mapping it to the output, and generating meaningful non-auditory feedback. In order to get gesture data from the performer, an interface with the human is needed. This project is focused on physical interfaces as opposed to other kinds, such as video or motion sens_ors, because they provide more reliable and methods of capturing gesture data with higher resolution. Also, physical interfaces allow the possibility of haptic feedback.

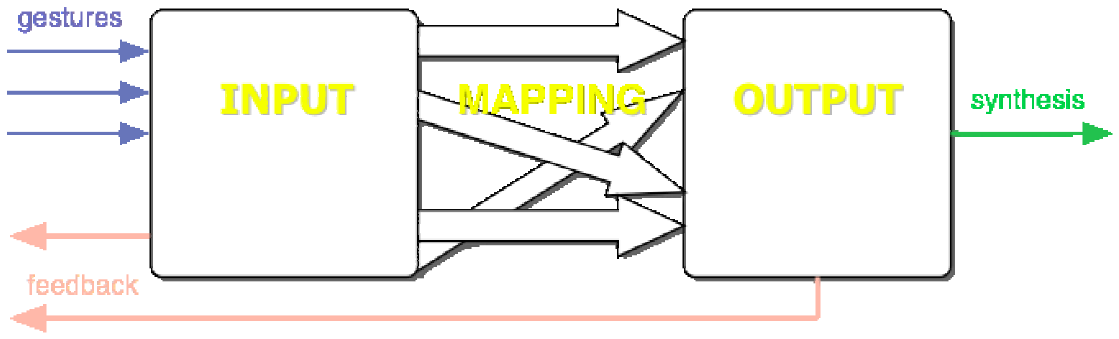

Figure 3-1: input, output, mapping, and feedback

Human Interface Devices

When talking about interacting with computers, "HID" has become the standard term for devices designed to control some aspect of a computer. A wide range of devices are classified as HIDs, including standard computer devices like mice and keyboards, as well as gaming devices like joysticks and gamepads, to devices for more specific needs like drawing tablets. There are a number of high end devices available as well like the SensAble Phantom 6DOF [47] controller, costing in the tens of thousands of dollars.

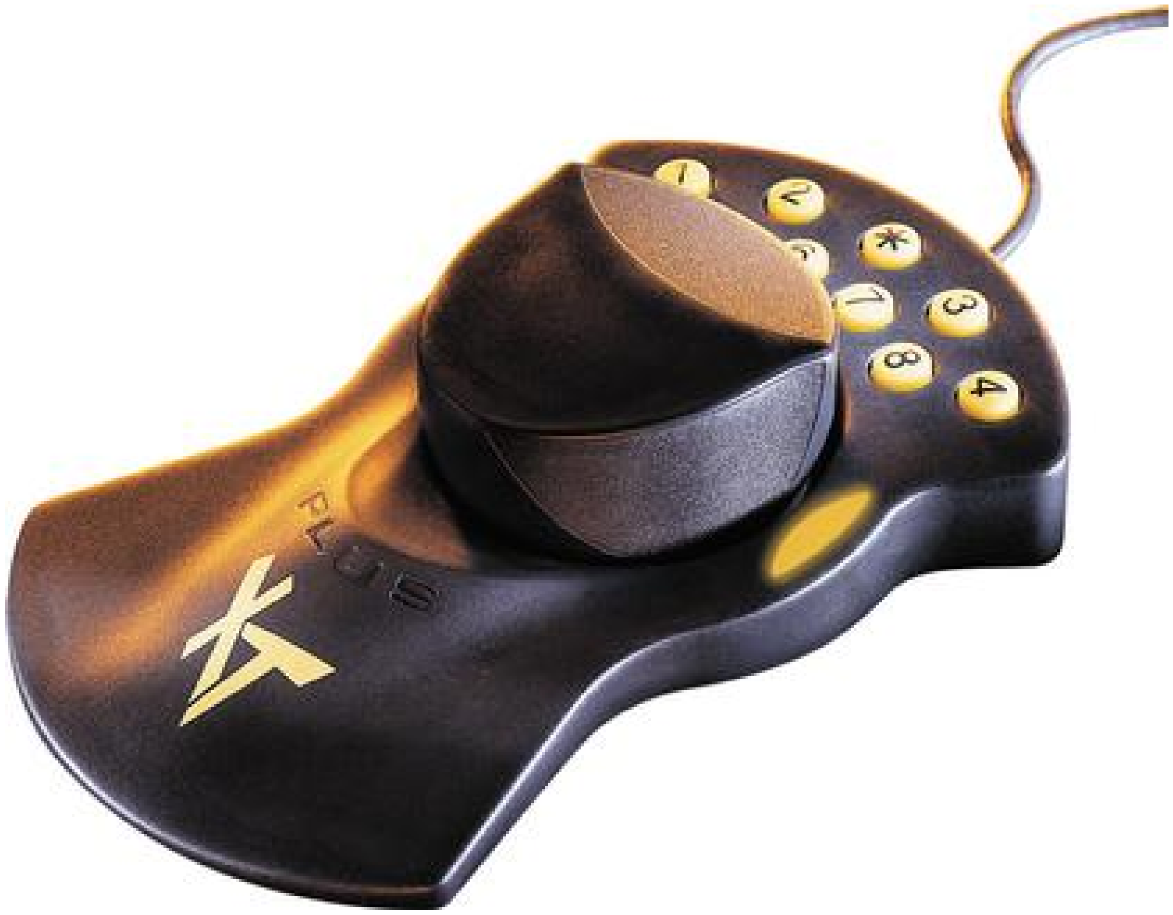

Figure 3-2: SpaceMouse

Figure 3-3: gaming joystick

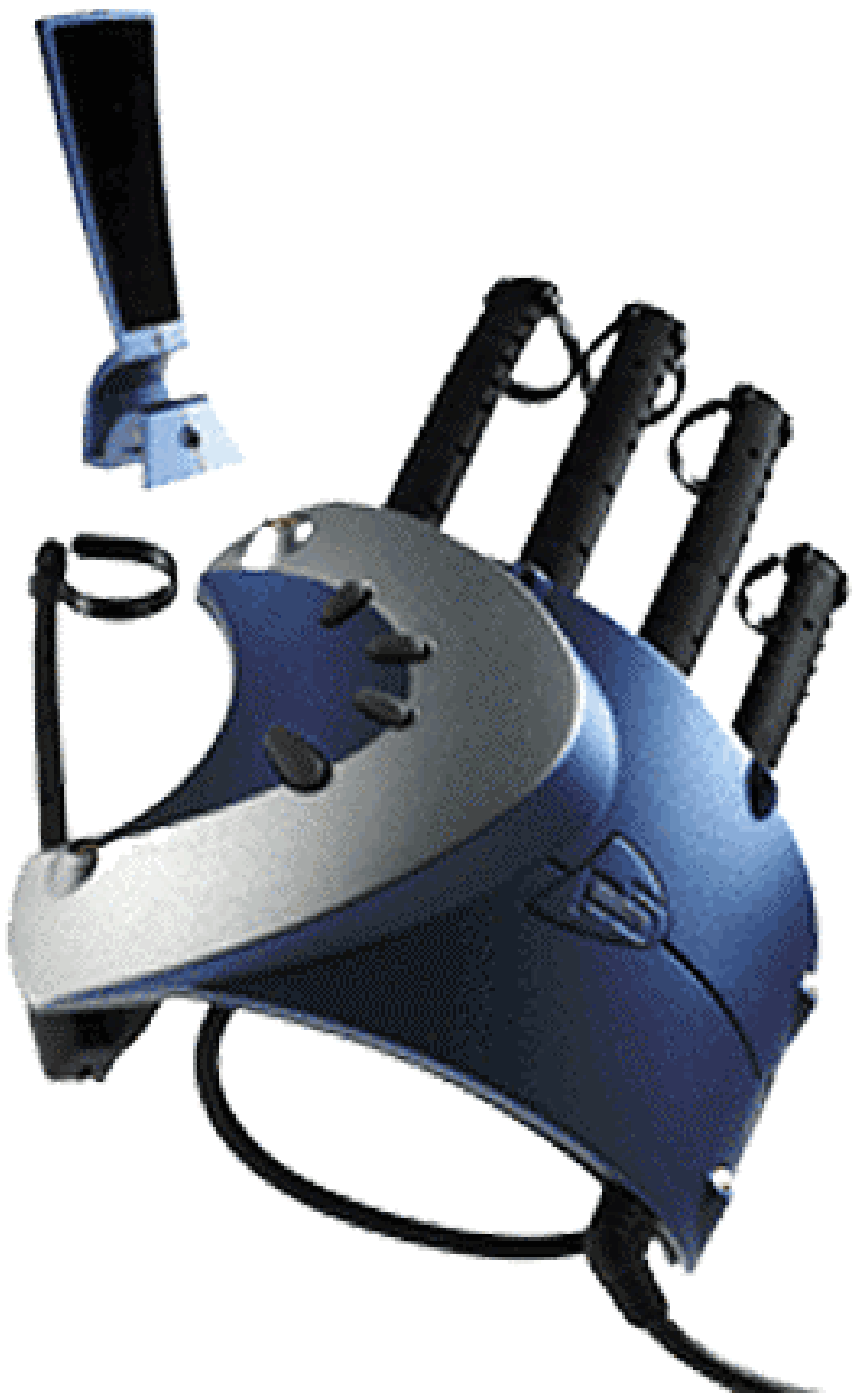

Figure 3-4: P5 Glove

HIDs usually consist of a number of common elements. They generally offer multiple axes, or dimensions. These axes can either output an absolute position, or a relative position (usually the amount of change per refresh cycle). A typical gaming joystick, for example, has 4 absolute axes: X, Y, handle-twist, and throttle, while a typical mouse has two relative axes: X and Y. The other major class of elements is buttons, which almost always have a default state of off, and can be pressed to output an "on" state. Buttons can be anything from mouse buttons to joystick triggers to keyboard keys. Another class is pseudo-axes. These are elements operate like an axis, but are generally implemented using buttons, so they only output an on or off state. Mouse wheels and joystick hat-switches are examples of pseudo-axes. They operate like axes, but output only on or off values depending on direction, because they are implemented with using buttons.

For this thesis, I chose to focus particularly on consumer level HIDs for a number of reasons. First and foremost, they are cheap and readily available. While some off-the-shelf HIDs are not up to the standards needed for musical performance, in terms of latency and resolution, many consumer HIDs perform quite well as musical controllers. Devices notable for their performance include gaming controllers such as gaming mice and joysticks; and graphics tablets. These types of devices can be used with low latency and very high resolution. For example, a good USB optical mouse can provide 4000 DPI every 5 ms. Indeed, many people already have established a high level of skill with these devices. Gamers are quite skilled at using devices such as joysticks and mice; graphic designers often are very skilled with drawing tablets such as Wacom devices.

The *[hid] toolkit* provides unified and standardized access to HIDs, so one need not learn how to use a new object for each different HID. Once the user has learned how to use one HID within **Pd**, that knowledge will be easily transferrable to other devices. It is also cross-platform so that instrument designers do not need to know the details of a given operating system's HID implementation in order to use HIDs, and the patches will work cross-platform. High-level objects are provided for commonly used devices. These objects are designed to provide the base set of what is normally expected from that device. The [mouse] object, for example, provides data for an X axis, Y axis, a mouse wheel, and buttons. These objects always output data in the same range, between 0 and 1, automatically adjusting to a given device's native range. **Pd**'s native numeric data type is floating point, so such data fits in better in that environment than MIDI's 7-bit integers. This base set allows for rapid prototyping and greater accessibility for beginners. For those who want to learn the details of various HID implementations, low level objects exist ([hid], [linuxhid], [darwinhid], and soon [windowshid]).

For this thesis, a coherent, usable scheme was designed to represent the range of possible event data. The specifications for *USB HID*[7], *Mac OS X HID Manager* [8], and *Microsoft DirectInput* [9] are all arcane and overcomplicated for most users. The Linux input event system [10] is cleanly organized, and the [hid] scheme was built using it as a model. In order to make for more readable patches and enable rapid prototypes, the *[hid] toolkit* uses symbolic names like "rel_x" and "btn_0" rather than numeric values to represent the possible device elements.

Mapping

In the same way digital synthesis has freed instrument design from the constraints of the interface generating the sound, instrument designers are also free to design the mapping between the interface and the synthesis, separately from the design of the input and the output. Thus any arbitrary interface can be mapped to any given synthesis algorithm; indeed the mapping can also be designed to suit the goals of the designer[11]. HIDs almost always produce linear data but mappings in expressive instruments are rarely linear. More complex mappings usually create more engaging instruments. There are many common ideas that are frequency used when designing a mapping. For example, since humans perceive loudness and pitch on a logarithmic scale, the amplitude and frequency control data are generally mapped to a logarithmic scale as well. For a more complicated mapping of loudness, Fletcher-Munson equal-loudness contours could be used.

The *[hid] toolkit* provides a number of mapping objects for commonly used operations. These objects expect input and output data in the range of 0-1, in the same format as the input objects. Having a consistent input and output range makes it possible to easily chain [hid] objects without thinking about scaling the data for each operation. The range of 0-1 is the standard range for representing amplitude and stereo position for many computer music environments, including **Pd**. It is also very easy to scale 0-1 to other common ranges, like MIDI (0-127).

There are a number of strategies that have been used to derive mappings. The most straightforward method is looking at controls and parameters to be controlled. This often leads to direct mappings of controls to parameters, which can be a limited way of turning gestures into sound. Using the velocity or acceleration of a given control, for example, can provide for much more compelling gestural control. Rovan, Wanderley, Dubnov, and Depalle break down the strategies into three basic categories: one-to-one, one-to-many (divergent), many-to-one (convergent) [12] According to them, these methods provide a level of expressivity in the order listed, with many-to-one mappings creating more expressive instruments while generally making them more difficult to learn. A classic mapping strategy is creating a multi-dimensional "timbre space" which the musician navigates [13]. In this method, a few dimensions of timbre are designed and then mapped out into a dimensional space which the user can navigate. A. Cont, T. Coduys, and C. Henry present a different approach to mapping, using neural networks to create mappings that are based on learning gestures from the user [14]. Their software, written in **Pd**, makes designing mappings an iterative process, where the user ranks the desirability of a given gesture, leading eventually to a chosen array of performance gestures.

Before controller data can be mapped, the output from many devices needs to be smoothed or otherwise processed. On the most basic level, the range coming from the input device will have to be scaled to match the parameters being controlled. This happens automatically in the high level HID objects ([mouse], [joystick], [tablet], etc.) A number of commonly used curves are also available in the *[hid] toolkit*, including [hid_log], [hid_exponent], [hid_square], etc. Data from high resolution devices such as mice and tablets can be jerky and seemingly erratic. Numerous methods for smoothing sensor data exist. One technique is to take a running average of a set of most recent values from the stream is one technique. By making it a weighted average, you can ameliorate the added latency caused by the averaging. Another technique converts the stream to an audio signal and runs that signal through a low pass filter. These two techniques are represented by [hid_average] and [hid_lowpass] respectively.

Feedback

Computer performers need feedback that is otherwise lacking in the physical interface. Most computer performances stare intently at the screen while performing to get visual feedback. In standard computer music performance environments, the screen is the sole source of feedback besides the audio itself. The software interface provides visual feedback, usually in a very concrete manner, by displaying the status of various parameters with virtual knobs, sliders, or even just numeric values. The overall amplitude is commonly shown as well. By providing haptic feedback thru the physical interface, for example, this need to stare at the screen can be alleviated.

Each sense can be thought of as a separate channel for getting data to level of conscious processing. Each sense has a portion of the brain devoted to processing that channel's data. If data is not coming in on that channel, the eyes are closed for example, that portion of the brain is basically dormant and would not be reassigned to process other types of data. It is possible to utilize these otherwise unused channels without significantly adding to the cognitive load. There is, of course, a limit to number of streams of data that can be consciously processed; it is possible to overload the brain with too many channels of feedback. So the feedback should be designed with this in mind. If the feedback is designed so that there are strong correlations to the audio and/or other feedback channels, those streams will be chunked into one stream to be consciously processed. This would minimize the additional cognitive load while providing for a richer interaction.

"[M]uscular feedback can work on time scales far below those possible in auditory feedback." [15] Adding haptic feedback to an instrument allows the musician to accurately perform actions that would be left to guesswork if haptic feedback was not available. Each sense has its strength: the sense of touch has the shortest possible feedback loop, the vision system processes the most amount of data in a given moment. The instrument designer should keep this in mind when designing the feedback offered.

Haptic devices have become readily available and affordable. There are numerous gaming HIDs, such as joysticks, gamepads, and mice, which can provide a range of haptic 'effects' from vibrations to forces to friction. Since the motor control in these haptic controllers has been encapsulated into haptic 'effects', they are generally quite easy to control. There are also high-end 6 Degree-of- Freedom devices such as the SensAble Phantom, which can provide very detailed haptic feedback in its six degrees of freedom. But the price puts these devices far out of reach of the vast majority of people.

Since non-auditory feedback can greatly enhance the interaction of human and computer, such feedback should become a standard part of instrument design. The *[hid] toolkit* provides a number of objects for generating haptic 'effects' such as [hid_ff_periodic] or [hid_ff_spring]. They follow the same data range conventions as the rest of the input, so the input in expected to be between zero and one. Therefore, they should easily interoperate with the whole set of mapping objects.

Enabling Sharing of New Instruments

One common problem with most new interfaces for musical expression is that they are difficult and/or expensive to build. This greatly hinders the spread of the instrument unless the designer can convince a manufacturer to build it and the volume is large enough that it becomes affordable. Many people have started creating their own instruments using HIDs, but because of technical difficulties, they are difficult to share with other users. These two ideas were key motivators in the design of the *[hid] toolkit*. The common, cross-platform set of objects eases sharing of **Pd** patches. Also, using abstracted objects like [joystick] allows people to use existing **Pd** patches with devices that differ from the original designer of the patch. For example, it would not be necessary to have the exact same make and model of joystick as the **Pd** patch's designer, your joystick would just have to have the minimum set of elements needed by that patch.

A common and valid critique of many new interfaces for musical expression is, by their nature of being unique novel devices, few people know how to play them. If there are few people who play a given instrument, the ability to develop and share techniques for playing that instrument is quite limited. The goal of many designers of new interfaces for musical expression is to design an instrument that is so compelling that it becomes a common instrument, thereby allowing the possibility of building a body of shared technique. But it is exceedingly rare for a new instrument of any kind to catch on in any substantial way. The Theremin is one of the most successful designs, but it is even still very much a niche instrument, even though it has been in existance since 1919. Many such instruments are also expensive and difficult to make, further limiting the potential body of users.

We can look to the relatively recent phenomenon of "turntablism", or playing the turntable as an instrument. Turntable and DJ mixer interfaces have been long standardized, providing a common platform for turntablists. This has helped to enable the development of a body of turntablist technique, which has become quite well established over the last 2 or 3 decades. For example, "[t]he first scratch, normally done in eighth-notes or triplets in time with the music, in called the *baby-scratch*." [16] There are even classes in turntablist technique offered in many cities around the world.

Using consumer HIDs enable the building of a body of shared technique in two_ ways. First, these devices are affordable and easily available where ever computers are available. Therefore, many users could easily use an instrument designed by someone else. Second, performers can build upon the existing body of knowledge that comes from other serious users of these devices. For example, gamers have developed a body of technique for using mice, joysticks, keyboards, and other HIDs for manipulating video games. Here is a definition of a term used by many gamers: "Pawing (verb): the act of lifting a mouse and returning it to the center of your mouse pad. Useful when trying to make a sequence of fast mouse movements." [17] Graphic designers also have a similarly developed body of technique for using graphical tablets. In a more general way, a number of software packages such as Apple Motion [18], Mozilla Firefox [19] and Cocoa Gestures [20] support the mapping of gestures to actions and include a standard set of gestures.

Sometimes previously learned behavior for using various HIDs can be a hinderance to the development of an expressive interface. For example, while many mice have very high quality sensors and could be well suited to musical control, they have a particular set of gestures that are strongly associated with that device. The mouse sensor is well suited for capturing large gestures, since there is no limit to the size of the gesture that it can measure since it outputs relative position information. But the mouse is strongly associated with manipulating a standard computer interface, which is generally done with small, discrete gestures. In order to break such habits, the instrument designer could modify the package of the device to force a different position, or just hold the mouse in a different style thereby breaking the familiar hand position.

4. Project Rationale

The increasing accessibility of computer music has made live computer music performance quite widespread with many tools such as **Max/MSP** or **Pd** performance patches designed just for this purpose. Though these tools can provide an engaging performance environment, the actual performance leaves something to be desired. The audience may be unable to tell whether the performer is actually controlling the music in real time, or just clicking a start button and reading their email. Such performance also lacks the physicality of the interaction and the accompanying range of gestures. Computer musicians are just starting to break out of the keyboard-mouse-monitor interaction. Using the computer for synthesis allows the interface to be tailored to the interaction without being constrained by how the sound is actually being generated. A multitude of means of getting physical input from the human body are readily available. These, in combination with the high level, rapid programming environment of **Pd**, allow a broad range of people to make their own computer-based physical musical instruments.

I chose to build upon **Pd** for a number of reasons. First, **Pd** is free software, so its available for free, and the source is completely open, allowing me to make any modifications that I need to in this process. Second, **Pd** runs on most operating systems, so it has a very broad potential audience. Third, it is a high level programming environment that is relatively easy for novices to pick up without serious limitations for experienced users. And last, **Pd** already is capable of a very wide variety of tasks, such as sound and video synthesis and manipulation, control of robotics, and interaction logic.

Pd objects for accessing a number of HIDs already exist. There are a couple problems with the existing software. They are unorganized and inconsistent, leaving many HIDs poorly supported or completely unsupported. And most of the objects were written separately, with their own interface and options. This means that in order to use different HIDs, the user has to learn a different object for many different HIDs. Since the basic principles are the same across the range of HIDs, the interface of objects should be similarly structured. My goal is to provide access to as many HIDs, including haptic devices, into a unified, standardized, and coherent approach.

Currently, good documentation for using HIDs with **Pd** or other softwares in not available from one source. One needs to search thru software documentation, various academic papers, and many web sites to gain a good overview of all of the various HIDs that can be used as musical controllers. Therefore an essential part of this thesis is unified documentation. It should cover various methods of getting data from human gestures, including HIDs, video, standard musical controllers, etc. as well as compare the various methods for their effectiveness in different situations and capturing different types of human gestures.

5. Background and Context

Humans have been building musical instruments for tens of thousands of years and music is an essential part of every human culture. And humans everywhere have continuously experimented with building new interfaces for musical expression. Musical instruments have kept pace with the technology of the time, with instrument builders finding ways to utilize new technology of their era. The twentieth century saw an unrivaled explosion of technological development, so it only follows that musical instruments would also experience similar development. Electronics started to be used to synthesize sound. Starting with the Theremin, instrument designers took the opportunity to try new ideas for the interface, since it could now be separated from the sound generation.

The 1950's opened the era of computer music. In 1955, Lejaren Hiller and Leonard Isaacson composed the *Illiac String Quartet*, the first computer-generated piece of music. Starting in 1957, Max Mathews and Joan Miller wrote the **Music N** family of programming language for synthesizing sound digitally. Computers were remained expensive for decades after this, so computer music was limited to academics who could get time on university computer systems. In 1963, Max Mathews declared "There are no theoretical limits to the performance of the computer as a source of musical sounds." By the late 70's, computer music centers like CCRMA were building custom computers such as the Samson Box exclusively for computer synthesis.

Figure 5-1: Computer music circa 1967

By the 80's, the introduction of personal computers made computing much more accessible. The combination of the Apple Macintosh and programs such as Laurie Spiegel's *Music Mouse* [21], and Miller Puckette's *Max* for Macintosh, computer music was beginning to be within reach for all those who might be interested. By now, even a cheap, used laptop has enough processing power for live performance. There has been a corresponding explosion of "laptop music", live computer music generally played in dance clubs and similar venues. In the same vein, many people have begun building physical instruments to control the sound generating by the computer.

Computers have given composers incredible control over creating sound, but the process generally leaves a lot to be desired, leading many computer musicians to make their own physical interfaces to control sound synthesis on the computer. Michel Waisvisz's *The Hands* is a great example. Built in the early eighties, it has been used to control a variety of different sound synthesis schemes. It is a novel interface that he has played for 20 years, achieving virtuosity. It allows him to stand on stage with nothing but *The Hands* and use gestures large and small to control sound and compose in realtime. Another example is Max Mathews' *Radio Drum*. A new user can immediately create sound with it, yet it is not too limiting for the expert player. It is capable of many different uses, such as conducting a virtual orchestra or controlling numerous samples and loops in realtime.

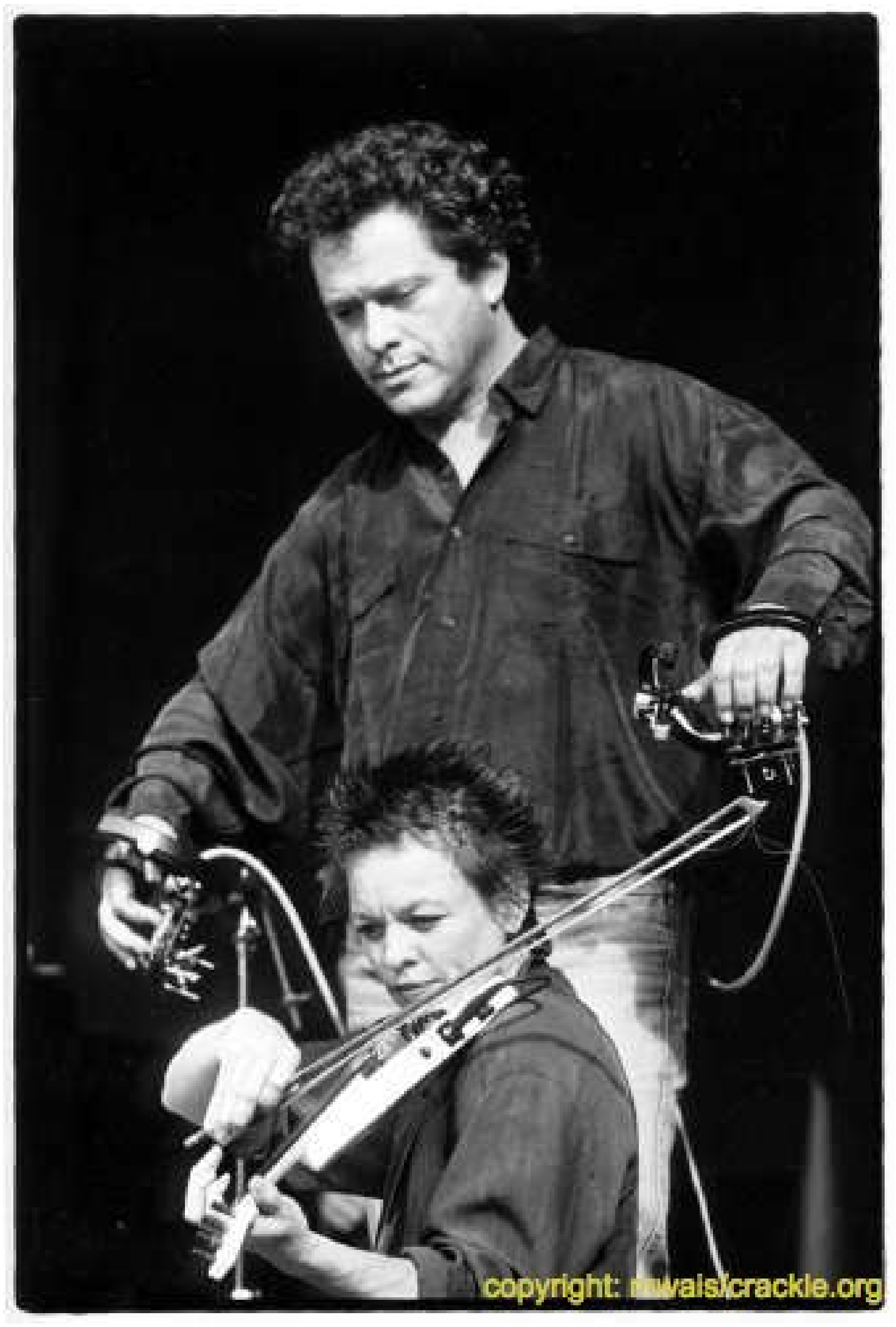

Figure 5-2: Michel Waisvisz and The Hands (with Laurie Anderson)

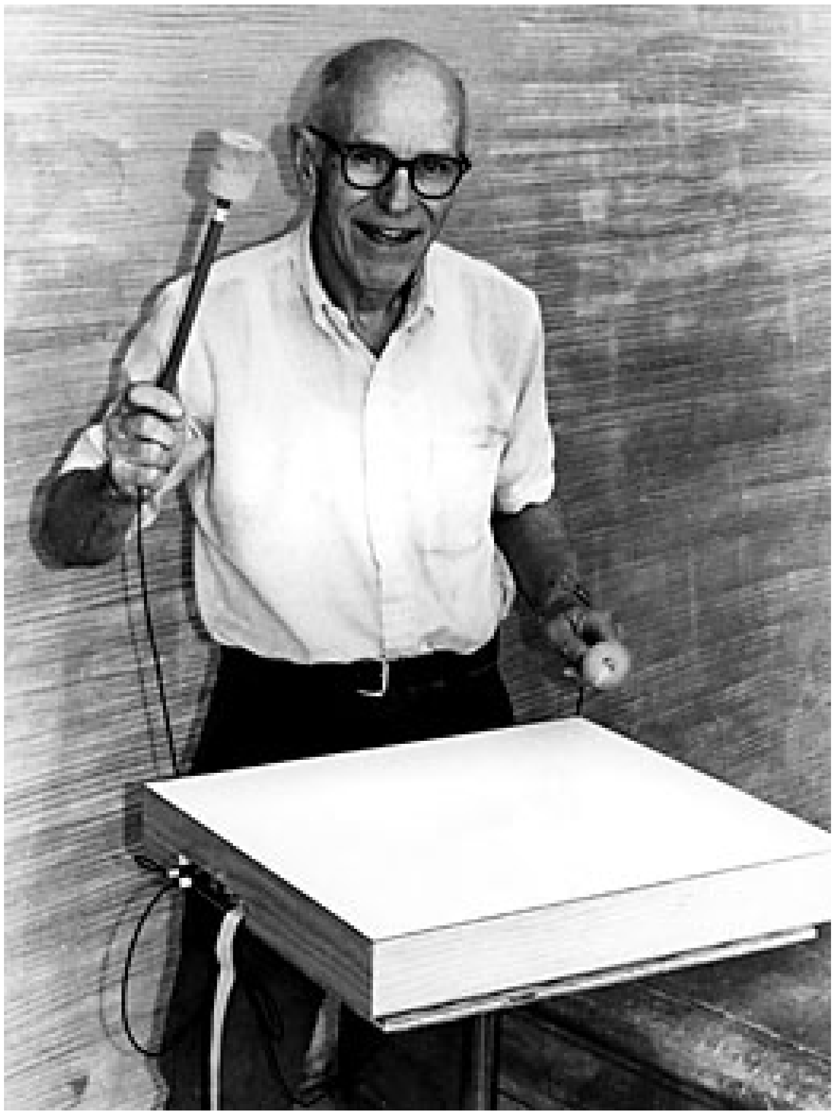

Figure 5-3: Max Mathews and his Radio Drum

These controllers were custom built and took a high level of expertise to create. Now there are many affordable, high quality musical controllers and HIDs so that people can use off-the-shelf devices rather than having to learn engineering and electronics in order to build their own instrument. There are a number of example of contemporary musicians who have mastered using a standard HIDs as a musical controller. Leon Gruenbaum's Samchillian Tip Tip Tip Cheeepeeeee [22] is built upon a standard ergonomic keyboard; playing with The Freight Elevator Quartet [23], Luke Dubois plays the Wacom tablet; Loïc Kessous has built his instrument using a Wacom tablet and a joystick [24]; Gerard Van Dongen tours with his Saitek force feedback joystick [25]; Nick Fells uses banks of MIDI sliders to control his Pd performance patches [26].

There is a conference that is dedicated to this specific field, the *New Interfaces for Musical Expression* (*NIME*) [27] conference, that has been running since 2000.

This conference is focused on research and practice related to building and designing new musical instruments of all kinds. Almost all of the instruments presented at this conference are built upon some computing processing. Besides the NIME conference, there are many people creating and playing their own instruments. Outside of the academic setting, there are many musicians who use a laptop as their primary instrument, hooking up MIDI control surfaces, HIDs, or even custom electronics to laptops in order to control sound. There are organized groups around the world where people who build their own instruments met to play together and share knowledge. There are numerous examples in many parts of the world: *Share* [28] and *FryLab* [29] in New York; *Pd Stammtisch* in Vienna; *Prutal Druth* [30] in Berlin; and *dorkbot* [31] in many cities throughout the world.

A new model of instrument design is emerging, shifting away from instruments designed for a broad user base, such as the Theremin, the MIDI keyboard the vast majority of traditional instruments. Instead many instrument builders are using systems of building blocks that allow the individual musician to create their own instrument relatively easily. This also contributes to a shift in the idea of musical instruments as a device for playing a wide range of pieces. Individual musicians can create their own instrument tailored to their performance goals, or even tailor an instrument to a specific piece or performance. One great advantage of the old model of instrument design is that musicians can develop and share a body of knowledge about how to play that instrument. This is something that has been severely lacking in the world of new interfaces for musical expression: it is rare for anyone to achieve virtuosity on these new instruments, even among the designers themselves. Using standard HIDs allows people to build a shared body of technique without sacrificing the ability to specifically tailor the instrument via the design of the mapping and the output.

My personal experience with designing new interfaces with musical expression started with my college Senior Project, *JoyStickMusicMachine* [32]. It is a program for taking the data from joysticks and mapping it to synthesis objects from the **NeXTSTEP MusicKit** [33]. The key motivation was to be able to control computer synthesis in realtime in order to break out of the keyboard/mouse/monitor interaction typical of computer music of the time. I followed on this idea with *StickMusic* [34], developing a specific instrument rather than a toolkit like *JoyStickMusicMachine* and the *[hid] toolkit*. StickMusic was creating using a force-feedback joystick and mouse, and was programmed in **Pd**. My experiences creating and performing with *StickMusic* lead directly to the creation of the *[hid] toolkit*.

First of all, I had to seek out and learn many different objects in order to use and experiment with different HIDs. These objects were usually written for one platform, so I could not run it on available computers, instead I always had to lug my PC along. Furthermore, it was quite difficult to share this patch with other interested musicians because of the details of setting it up and the technical requirements of getting it running. Through this experience, it became obvious that a unified toolkit was needed to address these issues.

An array of existing computer music platforms already allow for HID input, including **Pd**. There are numerous examples of existing software for using HIDs within music software. Within **Max/MSP**, a number of objects exist for getting data from HIDs such as [hi], [hidin] [35], [MouseState], [Insprock], [Wacom], and [MTCcentroid]. Each of these objects has a different interface, so it is necessary to learn each object to use each device. The **Max/MSP** object [hi] is a good example for coherent integration because it provides a single interface to many different kinds of HIDs. Plus [hi]'s menu system makes things quite easy to setup. **SuperCollider** [4] provides very low level access to the *Mac OS X HID Manager*, following its interface directly. This allows for great flexibility, but is a hinderance to both rapid prototyping and novices just starting out. **Pd** has a number of objects and patches for using HIDs such as [MouseState], [linuxmouse], [linuxevent], [joystick], the Gem HID objects, *kaos tools*, and **P5midiPD** [36]. But, like **Max/MSP**, they all have different interfaces, so the user must learn different objects to use different HIDs. Other computer music environments such as **Csound** [37] do not provide as broad access to HIDs as **Pd** or **Max/MSP**.

There have been a couple of attempts at building frameworks for creating mappings for musical instruments. Two notable packages come from IRCAM. "**MnM** is a set of **Max/MSP** externals based on **FTM** providing a unified framework for various techniques of classification, recognition and mapping for motion capture data, sound and music." [38] An earlier attempt from IRCAM is the ESCHER toolkit for jMax [39] which is a set of objects to address various problems of mapping. In terms of haptic feedback, **Pd** is currently the only widely available computer music environment known to the author that has the ability to control haptic feedback. **Pd** has two means of using haptic feedback devices: the **ff** lib for force feedback joysticks, and [ifeel] for haptic iFeel mice.

6. Prototype Design

Intended Audience

The intended audience for the *[hid] toolkit* ranges from novices to advanced users, from no programming experience and limited digital media experience, to in depth technical knowledge. The design was focuses on the tools necessary for advanced users to do rapid prototyping, with the belief that a well organized rapid prototyping system would also be relatively easy for beginners to learn.

Prior Work

My work designing computer music instruments started with *JoyStickMusicMachine*. Through the process of designing and implementing it, I learned a number of things about how to architect a flexible framework for building instruments using HIDs. First off, *JoyStickMusicMachine* was limited to only joysticks, it was not possible to use other kinds of HIDs. The mapping was done using pull-down menus in Windows to connect objects together. This method mapping was quite limiting because it did not allow for much processing to happen to the input data before being mapped to the synthesis objects.

The next project that I worked on that is directly related to the *[hid] toolkit* was the *StickMusic* instrument that I built in **Pd**. In *StickMusic*, I explored using haptic HIDs as controllers for real-time performance. Working in **Pd** allowed me huge flexibility in mapping and output, and I took advantage of this to play with ideas for mapping to the output and the haptic feedback. In order to get raw access to the HIDs I chose (a Saitek force-feedback joystick and a Logitech iFeel haptic mouse), I wrote the [linuxevent], [linuxmouse], [ifeel] objects for **Pd**. My experience with *StickMusic* directed my design of the event scheme and the [hid] object, as well as the mapping objects.

Event Polling

The polling of the input events was a key consideration in the design of the [hid] object, since low latency is crucial to a playable instrument. Much effort was made to reduce latency. Also, the input event polling is designed to be flexible so that it can be tailored to specific situations. There are two methods of polling available: polling automatically at regular intervals and manually, one chunk at a time. The interval time for automatic polling is easily settable from 1ms or above.

Event Naming Scheme

The overall scheme for the *[hid] toolkit* was based on the Linux input event scheme for a number of reasons, but it has some disadvantages, so the final *[hid] toolkit* scheme is a modified version of the Linux scheme. The Linux scheme has some aspects of it that are too specific, making it hard to abstract, i.e. button names for each device type, rather than just button numbers. While some parts of the scheme seem redundant, such as a "rel" event type for relative axes, with event codes for the relative axes, "rel_x", "rel_y" also labeled as relative. This redundancy provides more flexibility while directly reflecting the data as delivered from the operating system.

While creating *StickMusic*, I ran into many things that made the process quite frustrating. One example is having to look up the numeric code to remember what a given HID element was. These frustrations directed my design of the *[hid] toolkit*. I decided to use symbolic names for the elements rather than numbers, because usability was a key design concern, and most people find symbolic labels easier to remember than numeric labels. While there are some obvious disadvantages to symbolic labels in this context, such as increased CPU usage, none were severe enough to force the need for numeric labels.

It was also important to carefully devise the symbols themselves, making sure that they represented the elements well, and built upon existing schemes. Designing the button scheme highlighted this issue. *MacOS X HID Manager* [8] simply numbers the buttons, *Microsoft DirectInput* [9] works similarly since both are based on the *USB HID* specifications [7][40]. The Linux input event system [10] uses button names, like btn_left, btn_middle (mice); btn_trigger, btn_base (joysticks); btn_a, btn_select (gamepads); btn_tool_pen, btn_stylus (tablets); with a different naming scheme for each device type.
One key advantage of the button numbering scheme is that it allows buttons on one device to work in patches written for other devices. A patch written for a mouse could be triggered by the buttons of a joystick, tablet, etc. A minor disadvantage is that the user has to test the device to find the number scheme, rather than reading the label ("btn_0" vs. "btn_trigger").

Data Range

A common data format is essential in order to have the input, mapping, and feedback objects interoperate with each other. Otherwise, the instrument designer would have to have to think about converting the data ranges with each step. Even though the MIDI range of 0-127 is a loose standard in **Pd**, the *[hid] toolkit* uses the input/output range of 0-1. The computer audio standard is 0/1 (amp, pan, etc.), as well as the parameters for the **Gem** graphical environment for **Pd**. Also, 0-1 is much easier to convert to any other range. Using 0-1 for axes makes the data format the same across all of the HID elements as well: axes, buttons, and pseudo-axes all output data in the range 0-1. This opens up new possibilities for unorthodox mappings which might prove interesting.

Mapping

A number of mapping objects were created for this project, some are designed to work within the *[hid] toolkit*, while others are more general purpose. Objects such as [autoscale], [buttongate] and [keygate] can be used in any context. Objects that are designed to work within the toolkit have been named with a prefix "hid_" to make this clear. Objects such as [hid_cube], [hid_log], and [hid_average] all expect an input between 0 and 1, and their output is scaled to be within 0 and 1. This allows them to interoperate with all of the toolkit objects, but could make them less useful for general purpose applications.

Haptic Feedback

The *ff* library for **Pd** provided a number of objects for controlling haptic feedback effects in HIDs. Each of these objects sent the control messages directly to the device. To unify things and to keep as much as possible in the **Pd** realm, I chose to create haptic feedback effect objects that generated the control messages. In keeping with the rest of the *[hid] toolkit*, they were implemented them in Pd. These control messages are then sent to the [hid] object, which sends the messages to the device. Unless there was a strong reason to do otherwise, the haptic effect objects followed the interface of the *ff* objects.

7. Implementation of the Prototype

For the prototype, the bulk of the functionality is implemented on one platform, GNU/Linux. Then, to make sure that the event model that I designed for the [hid] object will work across platforms, I have a basic implementation working on MacOS X. I also researched the *Microsoft DirectInput* and *USB HID* and *USB PID* event models to make sure that the [hid] object's event model will be able to represent the range of data available.

Only the core [hid] objects ([hid], [linuxhid], [darwinhid]) are written in C, the rest of the objects are written in **Pd** itself. For these objects, it was necessary to program them in C since both the core of **Pd** and the *MacOS X HID Manager* are written in C. My intention was to implement as much of the *[hid] toolkit* in **Pd** as possible for a number of reasons. I think as a general rule, **Pd** objects should be implemented in **Pd** itself since it is a full-fledged programming environment, not merely an high-level application. Since programming in **Pd** is a different mindset that programming in procedural languages like C, I think that objects written in **Pd** will work better in **Pd**. Patches written in **Pd** are work across platforms (unless certain objects are used which are not cross-platform). Any user can the easily view the source of objects written in **Pd**, learn programming methods from them by example, and even use them as the basis for their own custom objects.

[linuxevent] and [ifeel] objects that I had previously written ended up being an easy prototyping platform, and I built the [hid] object starting with the [linuxevent] code. Even though the code changed a lot from the original [linuxevent], it proved valuable to start with working program, allowing me to test each step as it was implemented. My [ifeel] object and Gerard van Dongen's *ff* [41] library of objects provided prototypes for the haptic feedback implementation. The way that input and haptic feedback events are represented is quite different on each of the platforms. Therefore, in order to make a unified representation of events, convoluted and laborious code needs to be written. This has been implemented for the most common devices, but many remain to be implemented. Fortunately, this code can be written bit by bit, as the need arises, while the already implemented devices will be fully functional.

While I did the coding myself, there were a number of important contributions from others. The **Pd** lists [42] have been an essential resource for information, ideas, and advice. Ideas for mapping objects from Cyrille Henry and La Kitchen's set of mapping objects for sensor data, as well as Jamie Allen's mapping demo patch for **Max/MSP**. The **Pd** community has been my main pool of alpha testers, but some additional testing done here at ITP, using multiple different computers, and subjecting willing students to testing sessions.

8. User Testing

While there was no formal user testing performed for this thesis, much informal user testing was done, and influenced the design of aspects of the *[hid] toolkit*. The two forums that were utilized for this informal user testing were the **Pd** mailing lists and members of the ITP/NYU community. I also created a number of test patches in order to test ideas and their implementation. The user testing of this project started with my first attempt at a HID object for Pd, [linuxevent]. I wrote this object to provide access to all kinds of HID data to Linux Pd users. That object has been released for about one and half years. Numerous users have used [linuxevent] and reported many different bugs, as well as various problems and comments. The experiences of many users influenced the design of the *[hid] toolkit* and helped to outline the key issues preventing the level of interoperability necessary to enable instrument sharing.

The **Pd** mailing lists are the central forum for the **Pd** community. They provide a space to discuss all things related to **Pd**, from event announcements, new objects, coding ideas, bugs, problems etc. Many people post examples of their work, both to receive criticism as well as to distribute them to a broader audience. For this project, I took advantage of the **Pd** lists for informal user testing, bug testing, and discussion of concepts. Two types of users from ITP participated in the informal user testing: students with experience designing their own computer music instruments, and students with established skills using specific HIDs. Through interviews and demonstrations, I gathered information about existing software, HID techniques, and aspects of the *[hid] toolkit*. I also built a number of patched based on a number of different HIDs (mouse, joystick, tablet, keyboard) to test both the input objects and the mapping objects.

Following the open source tenet of "release early, release often", I made five releases to the **Pd** lists over the semester, starting with a rudimentary implementation. Through this process I received many bug reports which provided useful data for debugging. Also, the people who tested it confirmed that the software was working on different platforms, with different setups, and using different brands and models of HIDs. One key idea that came from user testing is to use symbolic names in the event naming scheme instead of numbers. For many users, it was necessary to constantly use lookup tables in order to remember which number was representing which HID element. Using symbolic names greatly reduced the number of lookups for some users while not increasing it for others.

The user testing process was helpful in determining that the button names should be based on sequential numbers ("btn_0", "btn_1", "btn_2", etc.) rather than descriptive names ("btn_left", "btn_trigger", etc.). While this may seem in opposition to the previous point, the meaning of most of the descriptive names was not apparent to many users and oftentimes were based on numbering schemes anyhow. Some examples of such names are: "btn_tl2" on a gamepad; "btn_base2" through "btn_base6" for a joystick; "btn_forward" and "btn_task" for a mouse. Also, the mouse button often called "btn_left" is not always the leftmost button on a mouse.

User testing also proved that the automatic range scaling of the high level input objects ([mouse], [joystick], etc.) allowed interoperability between devices of the same type, joysticks for example, even if they provided a drastically different range of data. Two joysticks were consistently used throughout the building of the *[hid] toolkit*, one with a range of 0-127 and another with a range of 0-4095. Both of these joysticks could be used with the same patch, with the same joystick position producing the same sound. The difference in data range was instead perceived as a difference in resolution of the control rather than a difference in the sound generated.

9. Conclusion

The *[hid] toolkit* provides a common platform for using HIDs within **Pd** with a unified framework for creating instruments from the HID data. A couple intrepid **Pd** users have already designed their own instruments using the *[hid] toolkit*. Included joystick- and mouse-based instruments work on numerous computers with different OS's and distinct brands and models of HIDs. In addition to the objects that are already working, this framework can be applied to a broad range of devices out for developing a complete platform for input, mapping, and feedback. Early user testing has confirmed that some key aspects of the *[hid] toolkit* make instrument design easier, both for novices and for more experienced users who want to build rapid prototypes. The toolkit also enables sharing of instrument patches. The unified, cross-platform objects for input, mapping and feedback work in a consistent manner. The high-level objects allow an instrument patch to work with HIDs of the same device type, joystick for example, but with differing specifications. Hopefully the *[hid] toolkit* will aid in the development a standard body of technique for HIDs, bringing virtuosity to the field of new interfaces for musical expression.

10. Future Work

Mapping

In terms of mapping, there many possibilities not addressed in the current version of the *[hid] toolkit*. Most of the current mapping objects cover relatively simple concepts. More complicated ideas like one-to-many and many-to-one mapping should be explored in the form of high-level objects. While these objects would be somewhat limited, they would provide a quick method to test ideas and an accessible way for novices to experiment with more complex mappings. Physical modeling offers a lot of potential over simple averaging and curve-mapping methods for processing the input data. The **pmpd** [43] library provides the basic building blocks for creating physical models within **Pd**. With **pmpd** it is possible to model physical interactions from the real world. When using a bowing action with the mouse, for example, a physical model of a violin bow on a string might prove quite appropriate, especially when combined with haptic feedback. This simulation of the violin bow action could be encapsulated in an object, expanding the possibilities of *[hid] toolkit*.

MIDI Devices and Sensors

The *[hid] toolkit* establishes a common framework for data from HIDs and general mapping operations. This framework should be extended to cover other methods used for getting gestural input. Many people who are used to working with electronics prefer to use MIDI controllers. **Pd** is fully capable of handling

MIDI, and given objects that convert the data from MIDI interfaces, users could use the *[hid] toolkit* mapping framework with MIDI devices. High-level objects for these MIDI devices following the model of [mouse], [joystick], etc. from the *[hid] toolkit*. By having a common interface, users could apply the same knowledge of how to use HID objects with *[hid] toolkit* to utilize MIDI input devices. Many people are also building instruments using sensors and other electronics. There is currently a multitude of methods of getting the data from the micro-controllers and sensor boxes, much like the multitude of interfaces and objects available for getting data from HIDs. Nothing that prevents these devices from fitting into the framework laid out with the *[hid] toolkit*. Using the same automatic calibration used in the high-level HID objects ([mouse], [tablet], [joystick], etc.), sensor data could also be representing using the range of 0-1. Having all of these means of getting input together in one coherent framework would allow instrument designers to seamlessly mix any number of these methods comparatively easily.

Visual Instruments

Many sound programming environments such as **Pd** have been extended to handle many different media, such as video, computer graphics, or even robotics. Gestural control could also make for engaging performance using any or all of these media. Instruments need not only be for music, visual instruments, for example, remain largely unexplored. Computer synthesis vastly expands the possibilities here, and performers are starting to create visual instruments, building on old ideas like the Clavecin Oculare. Live video performance is already common, and video performers are starting to use physical controllers, but almost always these controllers follow the video mixer paradigm. Just like the laptop performer stuck in the keyboard/mouse/monitor interaction, the wide range of human gesture is not well exploited when using video mixers.

Visual Feedback

Modern computer graphics capabilities can create complex visuals in real-time. Human vision is capable of processing a huge amount of data. Therefore, the possibilities of visual feedback are vast. Computer-based instruments could provide richer visual feedback than any traditional instrument. Very few new interfaces for musical expression take advantage of this, and they generally provide far less visual feedback than a traditional instrument. Explorations of visual feedback are becoming common, often in the context of audio-visual performance [44], [45]. *chdh* [46] is an excellent example of this fusion. The music and visuals are tightly linked, and the graphical manipulations are obviously linked to the sounds being produced. The audience as well as the performer can follow this visual feedback, providing a new expression of Wagner's idea of *Gesamtkunstwerk*. Currently, a strong knowledge of graphics is necessary in order to provide rich visual feedback. High-level objects for creating visual feedback would open up these possibilities to a wider range of people, and fit well into *[hid] toolkit* scheme.

11. Appendices

a) Formatting Conventions

There are two conventions used here which are derived from the **Pd** mailing lists:

[*object*]

This represents a _**Pd** object. If there is no reference after it, it is considered a part of the main distributions.

[*message(*

This represents a **Pd** message. [*message(*'s are generally discussed in relation to [*object*]'s that respond to the message.

b) Glossary

gamepad: a standard hardware device used to control video games that is usually held with two hands and consists of buttons and joysticks

gamer: a person who plays video games seriously

haptic: related to the sense of touch at the skin level and forces to the muscles and joints

HID: Human Interface Devices, such as a joystick, mouse, keyboard, gamepad, tablet, etc.

mapping: processing input data to control output and feedback

MIDI: Musical Instrument Digital Interface, an industry-standard interface used on electronic musical keyboards and PCs for computer control of musical instruments and devices

patch: a program written in a "patcher" environment like **Pd** or **Max/MSP**

Pd: *Pure Data*, an open source, multimedia software environment [6]

tablet: a hardware device used with a pen-like stylus used to control cursor movement

c) patches built with the [hid] toolkit

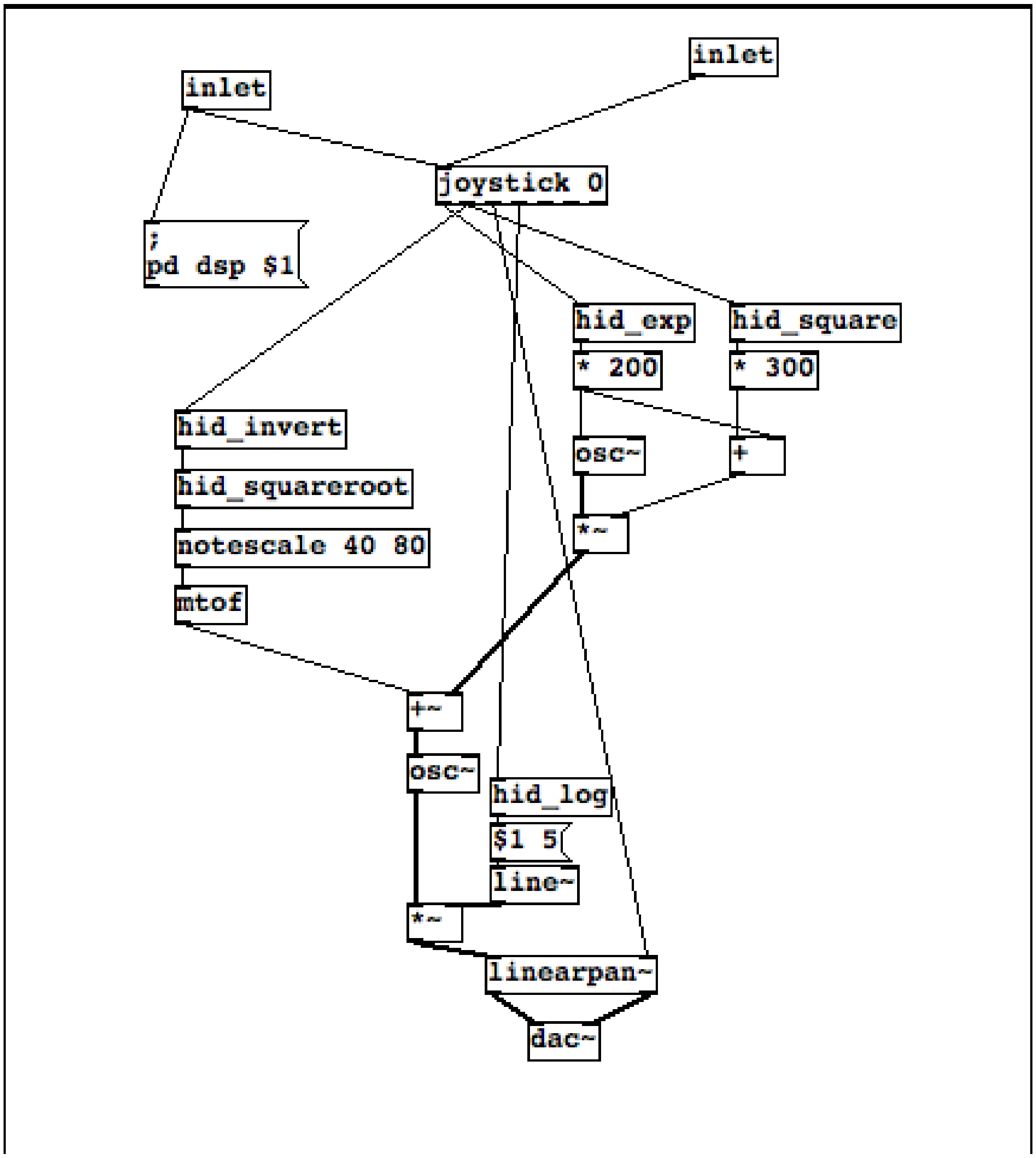

Figure 11-1: a simple joystick-based instrument patch

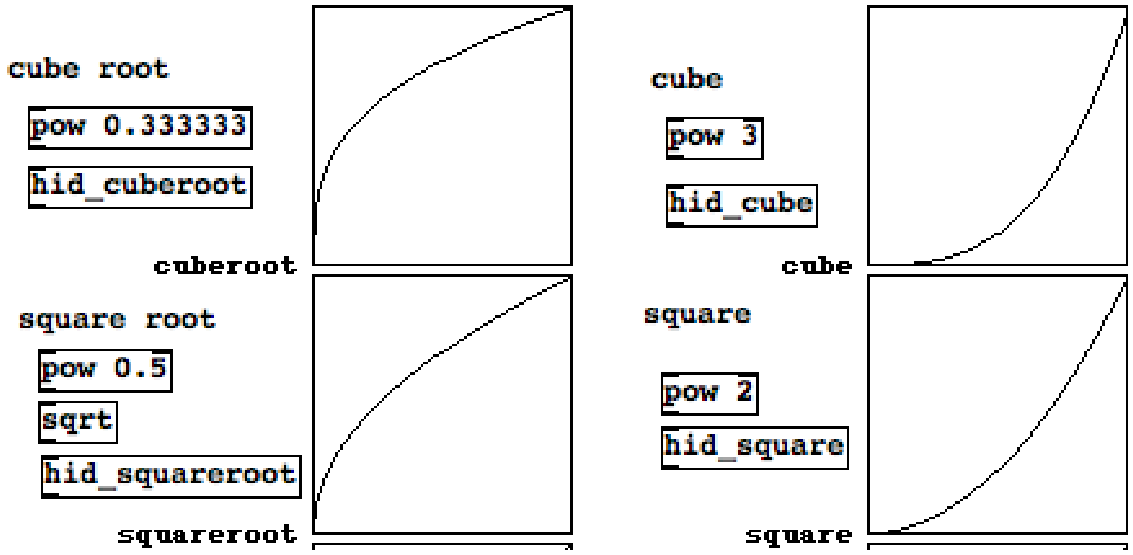

Figure 11-2: some curve objects for mapping

12. Bibliography

[1] B. Bongers, "The Use of Active Tactile and Force Feedback in Timbre Controlling Electronic Instruments", *Proceedings, International Computer Music Conference (ICMC 1993)*, 1993.

[2] Native Instruments **Reaktor**, http://www.nativeinstruments.de/index.php?reaktor_us

[3] **Ableton Live**, http://www.ableton.com/

[4] **SuperCollider**, http://www.audiosynth.com/

[5] Cycling '74 **Max/MSP**, http://www.cycling74.com/

[6] M. Puckette, "Pure Data: another integrated computer music environment", *Proceedings, International Computer Music Conference (ICMC 1996)*, 1996, pp. 269-272. also: http://puredata.org

[7] *Universal Serial Bus (USB): Device Class Definition for Human Interface Devices (HID)*, http://www.usb.org/developers/devclass_docs/HID1_11.pdf

[8] *Apple Mac OS X HID Manager*, http://developer.apple.com/documentation/DeviceDrivers/Conceptual/IOKitFundamentals/Families_Ref/chapter_11_section_7.html#//apple_ref/doc/uid/TP0000021/BABFEIAC

[9] *Microsoft DirectInput Overview*, http://msdn.microsoft.com/library/en-us/dninput/html/diov.asp

[10] *Linux Input Drivers*, http://linuxconsole.sourceforge.net/input/input.html

[11] A. Hunt, M. Wanderley, and M. Paradis, "The importance of parameter mapping in electronic instrument design", *Proceedings of the International Conference on New Interfaces for Musical Expression (NIME-02)*, 2002.

[12] J. B. Rovan, M. Wanderley, S. Dubnov, and P. Depalle, "Instrumental Gestural Mapping Strategies as Expressivity Determinants in Computer Music Performance", KANSEI - The Technology of Emotion, AIMI International Workshop, Genova, 2000. http://citeseer.ist.psu.edu/65256.html

[13] R. Vertegaal, "An evaluation of input devices for timbre space navigation", MPhil. dissertation, Department of computing, University of Bradford, 1994. http://citeseer.ist.psu.edu/vertegaal94evaluation.html

[14] A. Cont, T. Coduys, and C. Henry, "Real-time Gesture Mapping in Pd Environment using Neural Networks", *Proceedings of the International Conference on New Interfaces for Musical Expression (NIME-04)*, 2004 http://www.suac.net/NIME/NIME04/paper/NIME04_1C04.pdf

[15] M. Puckette and J. Settel "Nonobvious Roles for Electronics in Performance Enhancement." *Proceedings, International Computer Music Conference (ICMC 1993)*, 1993. http://crca.ucsd.edu/~msp/Publications/icmc93.ps

[16] K. F. Hansen, "Turntable Music", Musikklidenskapelig Årbok 2000. Dep. of music, the Norwegian University of Science and Technology, Trondheim 2000, 145-160 http://www.speech.kth.se/~hansen/turntablemusic.pdf

[17] ExtremeTech, "Review: RTR-720 Whoop-Ass Mouse", http://www.extremetech.com/article2/0,1558,1203359,00.asp

[18] **Apple Motion** gestures, http://www.apple.com/motion/advanced.html

[19] **Mozilla Firefox** gestures, https://update.mozilla.org/extensions/showlist.php?category=Mouse%20Gestures

[20] **Cocoa Gestures**, http://www.bitart.com/CocoaGestures.html

[21] L. Spiegel, **Music Mouse**, http://retiary.org/ls/programs.html

[22] Leon Gruenbaum's *Samchillian Tip Tip Tip Cheeepeeeee*, http://www.samchillian.com/

[23] Luke Dubois, http://www.cycling74.com/community/lukedubois.html

[24] L. Kessous and D. Arfib, "Bimanuality in Alternate Musical Instruments," *Proceedings of the International Conference on New Interfaces for Musical Expression (NIME-03)*, 2003. http://citeseer.ist.psu.edu/kessous03bimanuality.html

[25] Gerard Van Dongen, http://www.xs4all.nl/~gml/

[26] Nick Fells, http://www.gla.ac.uk/departments/music/staff/nick/

[27] *New Interfaces for Musical Expression* Conference, http://www.nime.org

[28] *Share @ Openair*, http://www.share.dj

[29] *FryLab*, http://www.frylab.info

[30] *Prutal Druth*, http://pd.iem.at/pdwiki/index.php?PrutalDruth

[31] *dorkbot*, http://dorkbot.org

[32] H.-C. Steiner, *JoyStickMusicMachine*, Senior Project, Bard College, 1996. http://at.or.at/hans/misc/bard/seniorproject/

[33] **NeXTSTEP MusicKit**, http://www.musickit.org

[34] H.-C. Steiner, "StickMusic: Using haptic feedback with a phase vocoder", *Proceedings of the International Conference on New Interfaces for Musical Expression (NIME-04)*, 2004. http://citeseer.ist.psu.edu/699201.html

[35] [hidin] **Max/MSP** object, http://www.akustische-kunst.org/maxmsp/dev/

[36] P5 MIDI patches for **Pd**, http://www.11h11.com/hugodini/projects/p5midipd.htm

[37] **Csound**, http://csounds.com/

[38] *MnM (Music is Not Mapping)*, http://recherche.ircam.fr/equipes/temps-reel/maxmsp/mnm.html

[39] M. Wanderley, N. Schnell, and J. Rovan, "ESCHER-modeling and performing composed instruments in real-time", *IEEE Systems, Man, and Cybernetics*, 1998. http://intl.ieeexplore.ieee.org/xpl/abs_free.jsp?arNumber=727836

[40] Universal Serial Bus (USB): Device Class Definition for Physical Interface Devices (PID), http://www.usb.org/developers/devclass_docs/pid1_01.pdf

[41] G. van Dongen, *ff* library, http://www.xs4all.nl/~gml/software.html

[42] **Pd** mailing lists, http://puredata.org/community/lists

[43] C. Henry, "pmpd : Physical modelling for Pure Data", *Proceedings, International Computer Music Conference (ICMC 2004)*, 2004, 37-41.

[44] S. Jorda, "Interactive Music Systems For Everyone: Exploring Visual Feedback As A Way For Creating More Intuitive, Efficient And Learnable Instruments", *Proceedings of the Stockholm Music Acoustics Conference (SMAC 03)*, 2003. http://citeseer.ist.psu.edu/580866.html

[45] S. Jorda, "Sonigraphical Instruments: From FMOL to the reacTable", *Proceedings of the International Conference on New Interfaces for Musical Expression (NIME-03)*, 2003. http://citeseer.ist.psu.edu/jorda03sonigraphical.html

[46] *chdh* (Cyrille and Damien Henry), http://www.chdh.net/

[47] SensAble Phantom,
http://www.sensable.com/products/phantom_ghost/phantom.asp

PICNETUSB, http://www.alecmcnamara.freeserve.co.uk/picnetusb/

D. Wessel, M. Wright, and J. Schott, "Intimate Musical Control of Computers with a Variety of Controllers and Gesture Mapping Metaphors", *Proceedings of the International Conference on New Interfaces for Musical Expression (NIME-02)*, 2002
http://www.cnmat.berkeley.edu/Research/NIME2002/NIME02WesselWrightSchottDm.html

N. Orio, N. Schnell, and M. Wanderley, "Input Devices for Musical Expression: Borrowing Tools from HCI" *New Interfaces for Musical Expression Workshop (NIME-01)* at *ACM CHI'01,* 2001. http://citeseer.ist.psu.edu/orio01input.html

A. Hunt, M. Wanderley, and M. Paradis, "The Importance of Parameter Mapping in Electronic Instrument Design", *Proceedings of the International Conference on New Interfaces for Musical Expression (NIME-02)*, 2002.
http://citeseer.ist.psu.edu/hunt02importance.html

D. Van Nort, M. Wanderley, and Philippe Depalle, "On the Choice of Mappings Based on Geometric Properties", *Proceedings of the International Conference on New Interfaces for Musical Expression (NIME-04)*, 2004.
http://citeseer.ist.psu.edu/699416.html

J. Benjamin, *Gestural Music Interface*, http://www.acm.uiuc.edu/sigchi/gmi/