[hid] toolkit: a unified framework for instrument design

Hans-Christoph Steiner
Interactive Telecommunications Program
New York University
New York, NY, USA
hans@at.or.at

ABSTRACT

The [hid] toolkit is a set of software objects for designing computer-based gestural instruments. All too frequently, computer-based performers are tied to the keyboard-mouse-monitor model, narrowly constraining the range of possible gestures. A multitude of gestural input devices are readily available, making it easy to utilize a broader range of gestures. Human Interface Devices (HIDs) such as joysticks, tablets, and gamepads are cheap and can be good musical controllers. Some even provide haptic feedback. The [hid] toolkit provides a unified, consistent framework for getting gestural data from these devices, controlling the feedback, and mapping this data to the desired output. The [hid] toolkit is built in Pd, which provides an ideal platform for this work, combining the ability to synthesize and control audio and video. The addition of easy access to gestural data allows for rapid prototypes. A usable environment also makes computer music instrument design accessible to novices.

Keywords

Instrument design, haptic feedback, gestural control, HID

1. INTRODUCTION

The [hid] toolkit is a set of Pd [13] objects for using a wide variety of HIDs (Human Interface Devices) to build computer music instruments. The main objective is to provide a standardized and coherent environment to create instruments using gestural interfaces, supporting rapid prototyping of instrument design ideas. The coherent, high-level objects also make instrument design accessible to the novice, providing an entry point into an otherwise difficult realm. Pd already provides a wealth of options for output as well as low level operators for mapping. The [hid] toolkit addresses the processes of getting input data, mapping it to the output, and generating meaningful non-auditory feedback such as vibrations. This project is focused on physical interfaces, as opposed to video or motion sensors, because they can capture gesture data more reliably and at higher resolution. Also, physical interfaces allow for haptic feedback.

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 objects for controlling haptic feedback. It is built to be an integral part of Pd, and most of the [hid] toolkit objects are written in Pd. 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 of all but a select few. It is now possible to build a computer music instrument using Pd that costs less, including the cost of the computer, than most traditional musical instruments.

NIME'05, May 26-28, 2005 Vancouver, BC, Canada

2. BACKGROUND

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. A wide range of software is available for live computer music performance; SuperCollider [4], Max/MSP [1], and Pd are designed expressly for this purpose. Many of these software environments are already capable of using data from Human Interface Devices (HIDs) such as joysticks, drawing tablets, gamepads, and mice.

Within Max/MSP, a number of objects exist for getting data from HIDs such as [hi], [hidin] [12], [MouseState], [Insprock], [Wacom], and [MTCcentroid], each with a distinct programming interface. The Max/MSP object [hi] is a good example for coherent integration because it provides a single interface to many different kinds of HIDs. SuperCollider provides very low level access to the Mac OS X HID Manager, following its interface directly. This allows for great flexibility, but the interface is far from intuitive. 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 [5]. But, like the Max/MSP objects, they all have different interfaces, making it necessary to learn each object to use each device. Since the basic principles are the same across the range of HIDs, the interface of objects should be similarly structured. Other computer music environments do not provide broad access to HIDs.

There have been a couple of attempts at building frameworks for creating musical instrument mappings. Two notable packages come from IRCAM. "MnM is a set of Max/MSP

externals... providing a unified framework for various techniques of classification, recognition and mapping for motion capture data, sound and music." [2] An earlier attempt from IRCAM is the ESCHER toolkit for jMax [18] 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. The ff library [7] supports force feedback joysticks, and the [ifeel] object supports haptic iFeel mice.

My personal experience with designing new instruments started with JoyStickMusicMachine [16], a program for mapping joystick data to synthesis objects. I followed up on this idea with StickMusic [17], developing a specific instrument rather than a toolkit. StickMusic was created using a force-feedback joystick and mouse, and was programmed in Pd. In order to get raw access to those HIDs, I wrote the [linuxevent], [linuxmouse], [ifeel] objects for Pd. Through these experiences, it became obvious that a unified toolkit was needed to provide a usable environment for instrument design.

3. OVERVIEW

3.1 Human Interface Devices

"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 devices like mice and keyboards; gaming devices like joysticks and gamepads; and devices for more specific needs, like drawing tablets. There are a number of high end devices available as well, like the SensAble Phantom 6DOF controller [3]. My goal is to provide access to as many HIDs, including haptic devices, into a unified, standardized, and coherent approach.

I chose to focus particularly on consumer HIDs for a few reasons. First and foremost, custom built hardware requires a high level of expertise to create, while HIDs are cheap and readily available. While some off-the-shelf HIDs are not up to the standards needed for musical performance, many consumer HIDs perform quite well. Devices notable for their performance include gaming mice, certain joysticks, and most graphics tablets. Also, HIDs such as joysticks are familiar to those attending computer music concerts, especially when compared to custom hardware, allowing the audience to better follow a performer's actions. Lastly, standard controllers enable the building of a body of technique, which can be shared among musicians. A number of example of contemporary musicians have mastered using a standard HIDs as musical controllers. Leon Gruenbaum's Samchillian Tip Tip Tip Cheeepeeeee [9] is built upon a standard ergonomic keyboard; Luke Dubois plays the Wacom tablet with The Freight Elevator Quartet [8]; Loc Kessous has built his instrument using a Wacom tablet and a joystick [11]; Gerard Van Dongen tours with his force feedback joystick [6].

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 with the [hid] toolkit, 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 in order to write cross-platform patches. High-level objects are provided for commonly used

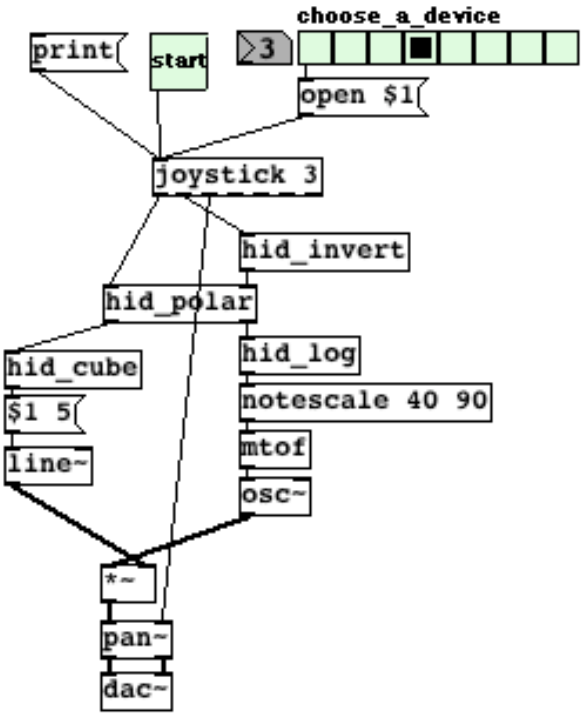

Figure 1: A patch using [joystick], [hid_invert], [hid_polar], [hid_log], [hid_cube], and [notescale].

devices, e.g. [gamepad], [tablet], etc. These objects provide data from the standard elements for that device. The [mouse] object, for example, provides data for an X axis, Y axis, a mouse wheel, and buttons. Using such abstracted objects allows people to use a given Pd patch with different devices of the same type. 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 provide the minimum set of elements needed by that patch.

3.2 Data Range

The [hid] toolkit uses the data range of 0-1 wherever possible. The standard data range for computer audio is 0-1 (amp, pan, etc.), as well with parameters controlling many other things. 0-1 is simple 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. Axis data can vary from 0-127 for many joysticks to 0-10,000 or more for tablets. Having all axes output data within 0-1 allows devices with widely varying output range to control the same patches without change. Most of the mapping objects expect input and output data in the same range. It is also very easy to scale 0-1 to other common ranges, like MIDI (0-127).

3.3 Mapping

In the same way digital synthesis has freed the physical interface from generating sound, computer music software allows the mapping to be a distinct system. 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 [10]. There are many common ideas that

are frequently used when mapping input to output. For example, since humans perceive loudness and pitch on a logarithmic scale, the amplitude and frequency control data are often mapped to a logarithmic scale. On the most basic level, the input device data must be scaled to the parameters being controlled.HIDs almost always produce linear data but mappings in expressive instruments are rarely linear. More complex mappings usually create more engaging instruments.

The [hid] toolkit provides a number of mapping objects for commonly used operations. Most are designed to work within the [hid] toolkit, these have a "hid_" prefix. Having a consistent input and output range makes it possible to chain [hid] objects without thinking about scaling the data between each operation. A number of commonly used curves are also available. [hid_log], [hid_exp], [hid_square], [hid_cube], [hid_squareroot], and [hid_cuberoot] all generate curves from their stated operation. Rovan, Wanderley, Dubnov, and Depalle break down mapping strategies into three basic categories: one-to-one, one-to-many (divergent), many-to-one (convergent) [15] According to them, these methods provide a level of expressivity in the order listed, with many-to-one mappings creating more expressive instruments. [hid_one2two] and [hid_one2three] are two objects that address this general idea. [hid_polar] and [hid_spiral] convert HID data to polar data generates polar coordinates, simplifying the creation of novel timbre spaces. Numerous methods for smoothing sensor data exist. One technique is to take a running weighted average of a set of most recent values from the stream is one technique. 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. Others objects are more general purpose. For example, [notescale] automatically scales the input to a range of discrete MIDI note values, as specified in the object's arguments. [buttongate] and [keygate] control a stream of data with a HID button or keyboard key.

3.4 Feedback

Since additional channels of feedback can greatly enhance the interaction of human and computer, such feedback should become a standard part of instrument design. For example, "[M]uscular feedback can work on time scales far below those possible in auditory feedback." [14] Adding haptic feedback to an instrument allows the musician to accurately perform actions that would otherwise be left to guesswork. Each sense has its unique strength: the sense of touch has the quickest feedback loop. The [hid] toolkit provides a number of objects for generating haptic 'effects' such as [hid_ff_periodic] or [hid_ff_spring]. The messages from these objects are then sent to the [hid] object, which sends the messages to the device. They follow the same data range conventions as the rest of the input, so they can be easily interoperate the mapping and input objects. Unless there was a strong reason to do otherwise, the haptic effect objects followed the interface of the existing ff objects for Pd.

3.5 Event Naming Scheme

A coherent, usable scheme was designed to represent the range of possible event data. The specifications for USB HID, Mac OS X HID Manager, and Microsoft DirectInput are all arcane and overcomplicated. The Linux input event system is cleanly organized, and the [hid] scheme was built using it as a model. For those who want to learn the details of various HID implementations, low level objects exist ([linuxhid], [darwinhid], and soon [windowshid]).

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. For example relative axes have a "rel" event type and "rel_x", "rel_y", etc. as event codes . This redundancy provides more flexibility while directly reflecting the data as delivered from the operating system. Symbolic names rather than numbers were chosen for the elements 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 did not needlessly deviant from existing schemes. Designing the button scheme highlighted this issue. MacOS X HID Manager simply numbers the buttons, Microsoft DirectInput works similarly since both are based on the USB HID specifications. The Linux input event system uses button names, like btn_left, btn_middle for mice; btn_trigger, btn_base for joysticks; btn_a, btn_select for gamepads; btn_tool_pen, btn_stylus for 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 joystick could be used by any other device with buttons and absolute axes, like a joystick or tablet. 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").

4. USER TESTING

While there was no formal user testing performed, much informal user testing was done, and was influential to the design of the [hid] toolkit. The two forums that were utilized for this informal user testing were the Pd mailing lists and the ITP/NYU community. 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, outlining key issues that would prevent sharing of instrument patches. 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.

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 variation

in data ranges was represented as resolution of the control rather than a difference in the sound generated.

5. CURRENT STATE

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. 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. A couple intrepid Pd users have already designed their own instruments using the [hid] toolkit. Joystick- and mouse-based patches included as examples work on numerous computers with differing OS's and models of HIDs.

6. CONCLUSIONS AND FUTURE WORK

The [hid] toolkit provides a common platform for using HIDs within Pd for creating instruments. Early user testing has confirmed that some key aspects of the [hid] toolkit simplifies instrument design, both for novices and for more experienced users. 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, joysticks for example, but with differing specifications. 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. MIDI controllers and sensors with microcontrollers are two common devices used that would fit well into the [hid] toolkit framework.

In terms of mapping, many possibilities have not been addressed in the current version of the [hid] toolkit. Most of the current mapping objects cover relatively simple concepts. More complicated ideas like many-to-one mapping should be explored in the form of high-level objects. Physical modeling offers a lot of potential over simple averaging and curve-mapping methods for processing the input data. Some common physical modeling ideas could be encapsulated in objects.

Modern computer graphics capabilities enable vast possibilities, but visual feedback for musical instruments and even visual instruments, remain largely unexplored. Computer-based instruments could provide richer visual feedback than any traditional instrument, but this idea is largely unexplored. A strong knowledge of graphics is necessary in order to create an instrument with 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.

7. ACKNOWLEDGMENTS

The Pd list has been an essential resource for information, ideas, and advice. Ideas for mapping objects came from Cyrille Henry and La Kitchen's set of mapping objects, as well as Jamie Allen's mapping demo patch for Max/MSP. The Pd community has been my main pool of alpha testers. Additional testing was done here at ITP/NYU.

8. REFERENCES

[1] Max/MSP. `http://cycling74.com`.

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

[3] Sensable phantom. `http://sensable.com/products/phantom_ghost/phantom.asp`.

[4] SuperCollider. `http://audiosynth.com`.

[5] H. Dini. P5 midi patches for pd. `http://11h11.com/hugodini/projects/p5midipd.htm`.

[6] G. V. Dongen. `http://www.xs4all.nl/~gml/`.

[7] G. V. Dongen. ff library for pd. `http://www.xs4all.nl/~gml/software.html`.

[8] R. L. DuBois. An interview with luke dubois. `http://cycling74.com/community/lukedubois.html`.

[9] L. Gruenbaum. Samchillian Tip Tip Tip Cheeepeeeee. `http://samchillian.com`.

[10] A. Hunt, M. Wanderley, and M. Paradis. The importance of parameter mapping in electronic instrument design. In *Proceedings, Conference on New Interfaces for Musical Expression (NIME-02)*, 2002. `http://hct.ece.ubc.ca/nime/2002/proceedings/paper/hunt.pdf`.

[11] L. Kessous. Bimanuality in alternate musical instruments. In *Proceedings, Conference on New Interfaces for Musical Expression (NIME-03)*, 2003. `http://citeseer.org/kessous03bimanuality.html`.

[12] O. Matthes. hidin object. `http://akustische-kunst.org/maxmsp/dev/`.

[13] M. Puckette. Pure data: another integrated computer music environment. In *Proceedings, International Computer Music Conference (ICMC 1996)*, 1996. `http://citeseer.org/164971.html`.

[14] M. Puckette and Z. Settel. Nonobvious roles for electronics in performance enhancement. In *Proceedings, International Computer Music Conference (ICMC 1993)*, 1993. `http://crca.ucsd.edu/~msp/Publications/icmc93.ps`.

[15] J. B. Rovan, M. Wanderley, S. Dubnov, and P. Depalle. Instrumental gestural mapping strategies as expressivity determinants in computer music performance. In *KANSEI - The Technology of Emotion, AIMI International Workshop*, 2000. `http://citeseer.org/65256.html`.

[16] H.-C. Steiner. Joystickmusicmachine, 1996. Senior Project, Bard College. `http://at.or.at/hans/misc/bard/seniorproject/`.

[17] H.-C. Steiner. Stickmusic: Using haptic feedback with a phase vocoder. In *Proceedings, Conference on New Interfaces for Musical Expression (NIME-04)*, 2004. `http://citeseer.org/699201.html`.

[18] M. Wanderley, N. Schnell, and J. B. 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`.