

Building your own instrument with Pd

Hans-Christoph Steiner
at.or.at
Brooklyn, NY, USA
hans@at.or.at

ABSTRACT

Computer music performance environments have evolved greatly in recent years, allowing complex control and interaction with sound in real time. But the interaction has too frequently been tied to the keyboard-mouse-monitor model, narrowly constraining the range of possible gestures the performer can use. The range of human interface devices has also greatly increased, making it much easier for the computer musician to capture a broader range of gestures. Pd 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, making it usable by people of varying technical skill. This paper provides an overview of the numerous methods of interfaces with humans using Pd.

Keywords

Instrument design, haptic feedback, gestural control, HID

1. FORMATTING CONVENTIONS

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

[object] - This represents Pd object. If there is no reference, it is considered part of the main distributions.

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

2. INTRODUCTION

With the power that even a cheap laptop can provide, the computer has gained widespread acceptance as musical tool. More and more musicians are using computers as an instrument for live performance, with many tools such as Pd performance patches, Reaktor[rea()], or Ableton Live[abl()] 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 physicality in the interaction and is quite limited in the range of possible gestures.

Digital synthesis has freed instrument design from being tied to the physical method of generating sound. Thus any arbitrary interface can be mapped to any given synthesis algorithm. This allows musical instrument designers to design their physical interface 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. A new model of instrument design is emerging from this, 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. The New Interfaces for Musical Expression (NIME)[nim()] conference is representative of such work, formed largely by musicians who are creating and playing their own instruments.

Pd is an fertile platform for such work, providing a high level, rapid programming environment that is accessible to a wide range of people with varying background. It is a unified platform for a broad range of activities, combining realtime audio, video synthesis and manipulation, physical modelling, and more with many options for data input and output including MIDI, HID's, and general serial communications. Since 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.

3. INPUT

So many 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. 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. The first question is how much the musician wants to control at a given

Permission to make digital or hard copies of all or part of this work for any purpose are granted under a Creative Commons Attribution-ShareAlike license: <http://creativecommons.org/licenses/by-sa/2.0/>

pd conference, Sept. 27th - Oct. 3rd, 2004 Graz, Austria
Copyright 2005 Copyright remains with the author(s).



Figure 1: Michel Waisvisz with *The Hands*; Max Mathews with *The Radio Drum*

time and how much a human is capable of managing at a given moment.

3.1 Dimensions of Control

Humans can only control a limited number of dimensions simultaneously. You could say that humans have limited "bandwidth". [Cook(2001)] It is possible to provide too many dimensions of control, making the instrument difficult to play, hampering expressivity. If the instrument is too simple, then it will have a limited range. Existing instruments and playing techniques are a good model of how many dimensions of control are useful. One example is the bowing motion of the violin, which can easily be emulated using a mouse, or an arrangement of buttons, like a piano, which could come in many forms.

A good place to start for examples of the level of complexity that a human can manage in a physical interface are the tried-and-true traditional instruments. From violin to piano to any instrument that humans have mastered we can see examples of functional interfaces. The violin and the piano are good examples for this because both of them offer a wide range of control over the sound produced, but in quite different manners.

The violin has 4 dimensions of control: bowing velocity, bowing pressure, location of the bowing as compared to the bridge, and the finger position on the strings. Mostly, the violinist is playing one string with a maximum of two or maybe three strings at a time, but the violinist is basically always playing one line at any given moment.

The piano model differs from the violin; playing numerous simultaneous notes is standard. A good pianist can even play two separate lines on one hand, making four simultaneous lines possible. The trade-off comes in the amount of control that the pianist has over the string as compared to the violinist. The piano has 2 dimensions of control for each string: velocity of key press and application of the damper. The pedals provide another simple dimension of control with the soft pedal. The damper and sostenuto pedal both provide control over the same damper as the finger can control, but in a different way. The final dimension of control for the piano is which key you hit, giving a different pitch. This provides 4 common dimensions of control, with the soft pedal providing a rarely used and simple 5th dimension. So in



Figure 2: A typical gaming joystick.

comparison the violin, playing one line the piano is simpler, therefore there is bandwidth left to allow playing multiple lines simultaneously.

3.2 Human Interface Devices

There are many off-the-shelf Human Interface Devices (HIDs) which can serve as musical controllers. While some of them are not up to the standards needed for musical performance, in terms of latency and resolution, many standard USB HIDs perform quite well as controllers. Devices notable for their performance include gaming controllers such as gaming mice and joysticks; and tablets. These types of devices can be used with Pd with low latency and very high resolution. For example, USB optical mice can provide 4000 DPI every 5ms.

3.2.1 Joysticks

Anyone who has played a video game is familiar with a joystick. Joysticks come in a wide variety of shapes, sizes and forms. Joysticks can have between 7 and 10 bit resolution, and anywhere from a 25Hz to 250Hz sampling rate depending on the hardware and the drivers. The basic analog joystick is inexpensive and leaves a lot to be desired as a musical controller. Modern gaming devices are of much higher quality and are better suited to musical applications. A joystick provides two or three absolute axes of measure per hand. Since a joystick's position data is absolute, the position of the joystick can be tightly mapped to different control parameters. For example, StickMusic[?] uses the axes of a joystick to control the timbre, therefore the timbre will be the same given the same position of the joystick.

There are two objects available in Pd for using joysticks: Joseph Sarlo's *[joystick]* [Sarlo()] and Hans-Christoph Steiner's *[linuxevent]* [Steiner(b)]. *[joystick]*'s outlets are dynamically created, providing an easy way to access all of the axes a given joystick has. But it has the disadvantage



Figure 3: A high-performance gaming mouse.

of having different numbers of outlets with different joysticks, making it harder for a given Pd patch to work with multiple joysticks. `[linuxevent]` provides access to all supported buttons, hat switches, and axes through a fixed number of outlets. This data is provided out of 4 outlets: time, the timestamp of the data; type, the type of data, i.e. axis, button, etc.; code, the specific instance of the type, i.e. X-axis, trigger button; and value, the data for that specific instance. When using `[linuxevent]`, the value data will need to be routed based on the type and the code

`[linuxjoystick]` is a simplified object based on `[linuxevent]`. It has a fixed amount of outlets, so that your patch doesn't need to change with different joysticks, with the disadvantage being that this object doesn't support axes beyond X, Y, twist, throttle, and 1 hat switch. You can make a custom object for your joystick using an abstraction based on `[linuxevent]` (see the Pd help patch for an example).

3.2.2 Mice

The mouse has a lot of untapped potential as a musical controller though they are generally viewed as a pointing device relegated to moving a pointer around a screen. But the instrument designer need not use the mouse in its traditional role. Modern mice can be high precision devices that work well in a variety of situations. Optical USB mice can have down to 5 ms of latency with up to 4000 dpi resolution per cycle, making them the most precise of the relatively cheap HID's. Also, mice come in a wide variety of forms, from standard mice with balls or optical sensors, to trackballs, or trackpads. If it works as a standard mouse it can be used with the standard mouse objects.

The mouse is a little more tricky to use than other HID's because, like the keyboard, it is usually in constant use by the OS. This means that mouse button clicks can mistakenly land on buttons and other GUI widgets with unintended effects. There are a number of ways to avoid this. The most basic and rudimentary method is to maximize a blank Pd window over the whole screen. Another basic workaround is to avoid using the mouse buttons at all in the instrument. Currently then best solution is to run the patch in `-nogui`



Figure 4: A keyboard customized for gaming.

mode, then there is no GUI for the mouse pointer to interact with. Ideally, there would be a way to steal the mouse from the GUI from within Pd so that it can use the mouse exclusively.

There are a number of objects available for getting data from the mouse. `[linuxmouse]` or `[linuxevent]` are currently the best choices in general because they provide the most flexible access to the mouse data. `[linuxmouse]` provides low level access to the mouse data. It outputs the relative change in position in screen pixels from the previous polling cycle. Using the relative data means that the mouse data is not limited by the size of the screen. With other objects, when the mouse pointer hits the edge of the screen, the data in that dimension stops changing, since its outputting absolute pixel coordinates. `[linuxmouse]` outputs the relative change in position each cycle regardless of where the mouse pointer is. The units are still screen pixels, but the size of the area where the mouse can move is unlimited. Set the mouse at the highest tracking speed for the highest resolution. This will make the pixels per centimeter the mouse moves much higher.

`[MouseState]` outputs the absolute coordinates of the mouse pointer on the screen in terms of pixels. This object is a clone of the Max/MSP object. Since Max/MSP originally ran on the MacOS only, this object was tailored for that environment, meaning that it only tracks the state of the first mouse button. All of the other mouse buttons are totally ignored by this object. `[gemmouse]` provides the location of the mouse pointer within the Gem[gem()] window, which can provide a solution for the problem of the OS also using the mouse at the same time. But since the resolution is tied to the resolution of the Gem window, `[gemmouse]` provides equal or less resolution than `[MouseState]`. `[serialmouse]` provides access to the data from serial mice. They have a 25-50Hz refresh rate, which limits it as a serious musical controller.

One example of an instrument built using the `[linuxmouse]` object is StickMusic, and instrument created using a haptic (meaning related to the sense of touch) joystick and a haptic mouse. In StickMusic, the mouse is used as if bowing a string. The X-axis velocity is mapped to the amplitude of the output with Y-axis position mapped to the pitch. Using velocity over position works well with the mouse since the mouse provides only relative location information.

3.2.3 Keyboards

Basically every computer has a keyboard, which provides

a large array of buttons designed around the human hand and can detect multiple simultaneous key presses. The keyboard can be a compelling musical controller along the lines of the piano, but using it can trap the musician in the keyboard/mouse/monitor interaction model. The keyboard does not need to be used in the standard way, it could be a standalone instrument strapped onto the body like a guitar. It is a cheap and useful controller, if care is taken to escape the standard keyboard interaction.

Pd provides a number of objects for accessing the keyboard data. `[key]`, `[keyup]` and `[keyname]` report key presses as long as a Pd window has focus. They are standard Pd objects and work on all platforms. `[key]` outputs the system-specific key number on key down; `[keyup]` outputs the system-specific key number on key up; `[keyname]` outputs the key name on the right outlet, and the state of that key whenever a key is pressed or released. `[gemkeyboard]` and `[gemkeyname]` report key presses when the Gem window has focus. `[gemkeyboard]` outputs the system-specific key number on key down while `[gemkeyname]` outputs the key name on the right outlet, and the state of that key whenever a key is pressed or released. It is also possible to get all of the keyboard data using `[linuxevent]`. With this object, Pd will always get the key press data regardless of whether Pd or Gem has focus since it reads directly from the keyboard device.

As with the mouse, the OS and the Pd application also use the data from the keyboard, leading to the same problems. The same workarounds that work for the mouse apply when using the keyboard and a controller. Also, when Pd is running in `-nogui` mode, it will never have focus since there would be no GUI component running, therefore all the current keyboard objects but `[linuxevent]` would not report any data in `-nogui` mode.

3.2.4 Tablets

Digital tablets such as the Wacom products are quite attractive for use as a musical controller because they are very high resolution with sampling rate generally around 100Hz. The Wacoms provide very accurate absolute position data in the X and Y axis, as well as two dimensions of pen tilt, pen tip pressure, and button state. More than one positioning device can be used simultaneously, including multiple pens or in combination with a mouse-like device.

There are two options when it comes to using a tablet with Pd: `[gemtablet]` and `[linuxevent]`. `[gemtablet]` is quite similar to `[gemmouse]` and therefore has the same limitations in that the data is limited to the size of the Gem window. For use as a musical instrument, `[linuxevent]` provides access to the tablet data at its full resolution.

Since tablets generally serve as a system pointing device like a mouse, using tablets with Pd can cause the same problems as when using the mouse. The workarounds are the same as well, but with some operating systems it might be possible to stop the tablet from acting as a system pointing device thereby restricting access to the tablet data to Pd.

3.3 Other Devices

There are a wide variety of HID devices that don't fit neatly into the above categories, but nonetheless could be interesting musical controllers, devices such as steering wheels, gamepads, the Griffin PowerMate `[gri()]` USB knob or the SpaceOrb `[spa()]`. Many of these input devices are supported



Figure 5: PowerMate USB knob and SpaceOrb



Figure 6: A force-feedback gamepad and steering wheel

by the Linux input event system, which means that they are supported by the `[linuxevent]` object. `[gemorb]` supports the SpaceOrb, a six degree of freedom ball controller which attaches to the computer using a serial port. The SpaceOrb is a cheap and interesting controller, but in the long run, its low resolution limits its use as a musical controller.

3.4 Sensors and Electronics

There is a huge variety of sensors, switches, buttons, displays, and electronic devices readily available, from force-sensitive resistors to accelerometers to infrared proximity sensors. Building from individual parts allows the designer to tailor the controller closely to their desires.

3.5 Sensor Boxes

Recently, there has been a surge in the development of various sensor boxes which allow users to easily get data from various sensors into their computers. There are three main methods that these sensor boxes get data into the computer: serial, MIDI, and USB. The MIDI interfaces such as the Doepfer Contact-to-MIDI CTM64 `[doe()]` are the easiest way to use arbitrary devices for input. Such boards convert electric signals to MIDI, making them very easy to use in a musical setting.

For sensor boxes that use serial or USB to connect to the computer, how they are interfaced depends on what protocol is used over the serial or USB connection. Generally, there would need to be specific Pd objects written in order to use them with Pd, but there might be ways around this, such as using command line tools and the `[shell]` object. The MakingThings' Teleo `[tel()]` is an example of a USB interface with specific objects needed to use it. Currently there are only objects for Max/MSP, but there is talk of Pd support.



Figure 7: The Doepfer Pocket Fader

3.6 Microcontrollers

Microcontrollers such as the Microchip PIC [pic()] or the Atmel AVR [atm()] have become a popular method of getting sensor data into computers. They are cheap and run fast enough to track the output of an array of sensors. The downside is that a solid knowledge of electronics is needed to create reliable instruments. Also, many microcontrollers are too slow to provide good resolution.

Using [comport] for serial communications is currently the only functional method of accessing a microcontroller in Pd. There are successful instruments built with microcontrollers that use MIDI, OpenSoundControl (OSC), and custom protocols designed for the task at hand. The Stranglophone and Pierrophone[Sharon(2004)] use accelerometers and sliders to get data from the user, and then outputs standard MIDI data via a serial port. This MIDI data is used by Pd to control a synthesizer. In the Sound Shell [Raskob()], a PIC microcontroller samples infrared proximity sensors and hall effect magnetic sensors and outputs data serially using MIDI.

3.7 MIDI Equipment

A wide variety of controllers use MIDI to communicate, from MIDI slider boxes to multipurpose "control surfaces" to more esoteric controllers like the Kaoss Pad [kao()]. Since the roots of Pd and the Max family of data flow languages lies in MIDI, it is very well supported. MIDI devices are generally very low latency, but the MIDI protocol itself is designed around 7-bit resolution with some 14-bit resolution devices. 7-bit is quite limited for a controller, especially compared to USB tablets and mice.

There are many variations of the mixing board, known as MIDI "control surfaces", which provide anything from rows of basic sliders to large consoles with sliders, knobs, buttons, etc. They generally are reliable and designed for musical applications, making them a natural choice for a musical controller. Nick Fells uses MIDI control surfaces in a number of different instruments. His pieces Words on the streets are these and Still Life[Fells(2002)] are two examples. They use the Peavey PC1600x control surface, mapping each slider to various parameters to be directly controlled in realtime.

The Kaoss Pad is built around a two dimensional touch controller. It also has a couple arrays of buttons as well as a couple switches and knobs, which output MIDI data. Using this MIDI output in Pd, you can map all of the parts of the controller however you see fit. Kaos Tools [kao()] provide a

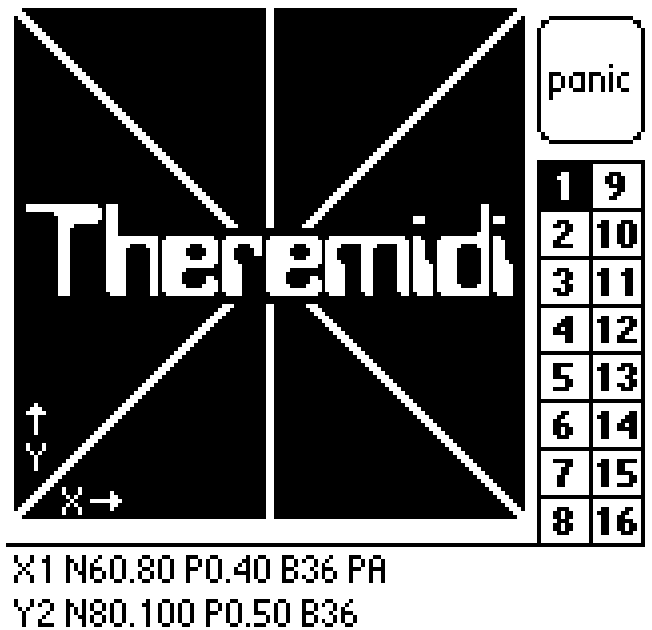


Figure 8: Theremidi, a touchscreen MIDI instrument

set of objects that streamline the use of the Kaoss Pad in Pd. The Palm Pilot in combination with the program Theremidi [the()] can provide a cheap alternative to the Kaoss Pad. Theremidi uses the Palm Pilot touchscreen as an X/Y MIDI controller, much like the Kaoss Pad.

3.8 Video

Computers have gotten to the point that heavy video and graphics processing is quite affordable. This opens up the visual dimension for the musician in a whole new way. Motion and color tracking using video processing allows all sorts of interactions that previously would have been quite expensive and difficult to implement. By extracting data from live video streams, a wide range of gestures can be captured and mapped how the instrument designer sees fit.

Video is a noisy medium. Lighting and postprocessing can be important in obtaining usable data values. Most current video cameras run at slow frame rates, and are therefore not low latency. When using NTSC (60hz field rate) or PAL (50hz field rate) video capture devices, there is a built-in 16/20 msec delay from action to detection, not including processing time. Most webcams run at a framerate of 30 Hz, quite slow for use as a musical controller. But webcams using USB 2.0 or IEEE-1394 ("Firewire") can provide much higher frame rates to reduce this latency, some as high as 120Hz.

There are three key methods of tracking gestures with video: color, motion, and shape. The most common one is using motion detection. With Gem, you can use [pix_movement] in combination with [pix_blob]; PDP provides [pdp_mgrid], which is grid-based motion tracking; GridFlow provides motion detection by subtracting previous frame from current frame using [@-]. For color and shape tracking, PDP provides [pdp_ctrack] and [pdp_shape] respectively. Another option is to process the visual data using outside software and feed that data into

Pd. `reactTable` [Kaltenbrunner et al.(2004)Kaltenbrunner, Geiger, and Jordà] takes that approach, using OSC to communicate between the two pieces of software.

4. NON-AUDITORY FEEDBACK

In using computer to make music, we have gained a large degree of control over the sounds we can make and coordinate. But with almost all computer music experiences, there is something lacking that the traditional musical experience is rich in: non-auditory feedback. Mostly this feedback is in the form of haptic sensations such as vibration, but the traditional musician can also see many of the workings of the sound generation in action, i.e. the movement of a guitar string. These two key methods of providing feedback, haptic and visual, are available within the Pd environment, which enable Pd to provide rich, instantaneous feedback. In effect, using non-auditory feedback opens up unused input channels to the brain.

4.1 Haptics

Adding haptic feedback brings back the tactile feedback loop that is a fundamental aspect of playing traditional instruments [O'Modhrain(2000)]. Haptic interfaces have thus far mostly been used to control synthesis methods that have direct analogies to the physical world [Nichols(2002)] [Cadoz et al.(2003)Cadoz, Luciani, Florens, and Castagné] [Howard et al.(2003)Howard, Rimell, and Hunt]. Scanned synthesis, designed with haptics in mind, relies on metaphors from the physical world [Wright et al.(2001)Wright, Freed, Lee, Madden, and Momeni] to synthesize sound. `StickMusic` [?] uses haptics to provide feedback for a phase vocoder, which has no analogy to the physical world.

While somewhat limited as musical controllers, using off-the-shelf haptic gaming devices allows the user to rapidly ramp up and build a functional instrument that employs haptic feedback. Haptic joysticks offer many different types of feedback, from forces to friction to vibration. Haptic mice have a motor which can create pulses that are perceived as either individual events, a stream of pulses, or an audible vibration, depending on their frequency. It is also possible to construct simple haptic devices such as `The Plank` [Verplank et al.(2002)Verplank, Gurevich, and Mathews].

The `ff` [Dongen()] library provides objects to control the feedback in most gaming haptic controllers, with different objects for each of the supported haptic effects, such as `[ff-spring]` and `[ff-periodic]`. `[ifeel]` [Steiner(a)] controls the pulse motor in iFeel mice. Currently, these objects only work on GNU/Linux. If you want to build your own haptic devices, you will need to figure out how to control them from Pd. `[comport]` in combination with a microcontroller provides the easiest method.

4.2 Visuals

Contrary to haptic feedback which so far provides a simple imitation of the real world, visual computing opens up the realm of visual feedback far beyond what is offered in traditional instruments. While visual feedback is often important in traditional instruments, most of the activity is on a scale that is too small for human vision to detect. Using computer-generated visuals, the computer can provide complex feedback through a channel that is often underutilized in musical performance. The graphical synthesis and processing afforded by `Gem`[gem()], `PDP`[Schouten()], and

`GridFlow`[Bouchard()] Pd allows this work to be done entirely within Pd.

`reactTable` is an interesting example of visual feedback as applied in a realtime performance environment. It has various widgets which represent building blocks along the same lines as objects in Pd. When the user makes a connection between widgets, a visual connection is also made. This visual connection has different colors and textures depending on what kind of connection it is.

5. MAPPINGS

Digital synthesis has freed instrument design from being tied to the physical method of generating sound. 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 [Hunt et al.(2002)Hunt, Wanderley, and Paradis]. There are a number of strategies that have been used to derive mappings. The most straightforward method is thinking in terms of controls and parameters that should be controlled. But this often ends up leading 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. Another classic mapping strategy is creating a multi-dimensional "timbre space" which the musician navigates [Vertegaal(1994)]. In this method, a few dimensions of timbre of chosen and then mapped out into a dimensional space which the user can navigate. A. Cont, T. Coduys, and C. Henry present a novel approach to mapping, using neural networks to create mappings that are based on learning gestures from the user. [Cont et al.(2004)Cont, Coduys, and Henry] 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 scaled, smoothed, or otherwise processed in order to provide good control. On the most basic level, the range coming from the input device will have to be scaled to match the parameters being controlled. Data from high resolution devices such as mice and tablets can be jerky and seemingly erratic. Using an `[average]` object on the data stream, you can smooth the data stream. By making it a weighted average using the `[weight]` message, you can ameliorate the added latency caused by the averaging. When working with sensors and electronics, often the output of the sensors must be smoothed before it can be properly sampled since the resolution of the microprocessor is limited. This must then be done using electronically, using an integrator, for example.

OSC provides a framework for abstracting the mapping process, which can help clarify the problems of mapping [Wright et al.(2001)Wright, Freed, Lee, Madden, and Momeni]. For example, the output of the controller and the input of the synthesizer can be mapped using a descriptive OSC name space, allowing the instrument designer to more easily focus on the mapping without having to think about the implementation details of the controller or the synthesizer.

6. FUTURE WORK

Currently, GNU/Linux is the preferred platform for building instruments for Pd because of HID and force feedback objects only run on GNU/Linux. But both Windows and MacOS X have solid support for HID and force feedback, so the Pd objects just need to be written. Ideally, these objects would be written to use the same interface as existing objects so that instrument patches would could easily be used on as many platforms as possible. Since my current platform of choice is MacOS X, I plan on writing a set of cross-platform objects to support HID and haptic gaming devices and incorporate them into a HID toolkit for Pd.

In terms of mapping, there is a lot of potential in using physical modeling instead of simple averaging methods for processing the data from the input devices. The pmpd[Henry(2004)] library provides the basic building blocks for creating physical models within Pd.

Now that video and graphics have become a standard part of Pd, it seems that the time is ripe for the exploration of physical instruments for controlling visual synthesis. Most current video editing setups are physical interfaces, but are mostly keyboard/mouse based and not tailored for capturing gestures in order to synthesize visuals.

7. REFERENCES

- Ableton live. URL <http://www.ableton.com/index.php?main=live>.
- Atmel AVR. URL <http://atmel.com/products/avr/>.
- Doepfer CTM-64. URL <http://doepfer.de/ctm.htm>.
- Gem. URL <http://gem.iem.at>.
- Griffin PowerMate. URL <http://www.griffintechology.com/products/powermate/>.
- Korg Kaoss Pad. URL http://www.korg.com/gear/info.asp?A_PROD_NO=KP2.
- New interfaces for musical expression conference. URL <http://www.nime.org>.
- Microchip PIC. URL http://microchip.com/stellent/idcplg?IdcService=SS_GET_PAGE&nodeId=74.
- Native instruments reaktor. URL http://www.nativeinstruments.de/index.php?reaktor_us.
- SpaceTec SpaceOrb 360. URL <http://www.3dgamers.com/articles/more/37/>.
- MakingThings Teleo. URL <http://makingthings.com/teleo.htm>.
- Theremidi. URL <http://www.versiontracker.com/dyn/moreinfo/palm/3118>.
- M. Bouchard. GridFlow. URL <http://artengine.ca/gridflow/>.
- C. Cadoz, A. Luciani, J.-L. Florens, and N. Castagné. Acroe - ica artistic creation and computer interactive multisensory simulation force feedback gesture transducers. In *Proc. of the Conference on New Interfaces for Musical Expression (NIME03)*, 2003.
- A. Cont, T. Coduys, and C. Henry. Real-time gesture mapping in pd environment using neural networks. In *Proc. of the Conference on New Interfaces for Musical Expression (NIME04)*, 2004.
- P. Cook. Principles for designing computer music controllers. In *Proc. for Workshop in New Interfaces for Musical Expression*, Seattle, WA, USA, 2001. URL <http://citeseer.ist.psu.edu/544138.html>.
- G. V. Dongen. ff library for pd. URL <http://www.xs4all.nl/~gml/software.html>.
- N. Fells. On space, listening and interaction: Words on the streets are these and still life, 2002.
- C. Henry. pmpd: Physical Modelling for Pure Data. In *Proc. of the 2004 International Computer Music Conference*, pages 37–41, Miami, Florida, USA, 2004.
- D. Howard, S. Rimell, and A. Hunt. Force feedback gesture controlled physical modelling synthesis. In *Proc. of the Conference on New Interfaces for Musical Expression (NIME03)*, 2003.
- A. Hunt, M. Wanderley, and M. Paradis. The importance of parameter mapping in electronic instrument design. In *Proc. of the Conference on New Interfaces for Musical Expression (NIME02)*, 2002. URL <http://hct.ece.ubc.ca/nime/2002/proceedings/paper/hunt.pdf>.
- M. Kaltenbrunner, G. Geiger, and S. Jordà. Dynamic patches for live musical performance. In *Proc. of the Conference on New Interfaces for Musical Expression (NIME04)*, Hamamatsu, Japan, 2004.
- C. Nichols. The vbow: Development of a virtual violin bow. In *Proc. of the Conference on New Interfaces for Musical Expression (NIME02)*. NIME, 2002.
- S. O'Modhrain. *Playing By Feel: Incorporating Haptic Feedback into Computer-Based Musical Instruments*. PhD thesis, Stanford University, 2000. URL <http://ccrma-www.stanford.edu/~sile/thesis.html>.
- E. Raskob. Sound shell. URL <http://lowfrequency.org/itp/NIME/concept.html>.
- J. Sarlo. joystick pd object. URL <http://crca.ucsd.edu/~jsarlo/pd>.
- T. Schouten. Pure Data Packet (pdp). URL <http://zwizwa.fartit.com/pd/pdp/overview.html>.
- M. E. Sharon. The stranglophone: Enhancing expressiveness in live electronic music. In *Proc. of the Conference on New Interfaces for Musical Expression (NIME04)*, Hamamatsu, Japan, 2004.
- H.-C. Steiner. [ifeel] object for Pd, a. URL <http://at.or.at/hans/pd/hid.html>.
- H.-C. Steiner. [linuxevent] object for Pd, b. URL <http://at.or.at/hans/pd/hid.html>.
- B. Verplank, M. Gurevich, and M. Mathews. The plank: Designing a simple haptic controller. In *Proc. of the Conference on New Interfaces for Musical Expression (NIME02)*, 2002.
- R. Vertegaal. An evaluation of input devices for timbre space navigation. Master's thesis, Department of Computing, University of Bradford, 1994. URL <http://citeseer.org/vertegaal94evaluation.html>.
- M. Wright, A. Freed, A. Lee, T. Madden, and A. Momeni. Managing complexity with explicit mapping of gestures to sound control with osc managing complexity with explicit mapping of gestures to sound control with osc. In *Proc., International Computer Music Conference (ICMC)*, 2001.