Progress report on the Mapping Library for Pd

Hans-Christoph Steiner
IDMI/Polytechnic University
Brooklyn, New York, USA
hans@at.or.at

Cyrille Henry
chdh
Groupe de travail de l'Association Française d'Informatique Musicale
Paris, France
ch@chdh.net

ABSTRACT

In the context of building musical instruments using computers, mapping control data to sound generation has been discussed for decades. Yet unlike the standard unit generators for synthesis and effects, there is no standard catalog of mapping methods. The Mapping Library for Pd is a fledgling library of mapping primitives with the aim of cataloging existing mapping methods. Also included are techniques for conditioning sensor data to make it usable in the context of instrument design. As part of the process of generating a catalog of fundamental building blocks for mapping, we are creating a software library of mapping methods based on both research and real world projects. The ultimate goal is to derive a set of unit generators for mapping, plotting out the most basic building blocks for creating instrument mappings. This paper presents some foundations and recent exploration related to shaping curves and physical modeling functions.

0.1 Keywords

instrument building, mapping, physical modeling, shaping curves, sensor data

1. INTRODUCTION

> Everything should be made as simple as possible
> - but no simpler. - Albert Einstein

When building musical instruments using human interface devices and computers, an essential part of the process is deriving a mapping of the control data to the process to be controlled. In this case, that process will most likely be audio signal processing on a computer. This is the type of mapping that this paper focuses on, though it need not be limited to the realm of musical instruments, but could also be directly applied to tracking dancers, building visual instruments, or other similar pursuits. Mapping for instruments has been discussed for decades. There have been a huge range of ideas touted, and many instruments built and tried. Shared elements and ideas are repeated, and re-implemented again and again. The foundations of audio have been long since codified with the standard unit generators, with few audio software packages disregarding them. They have become a basic language to express ideas in the realm of audio signal processing, whether in software or hardware. There is no catalog of fundamental mapping algorithms, and little work has been done to build the foundations into software. With no standard framework, instrument designers are constantly reinventing the wheel, re-implementing mapping algorithms whenever creating a new instrument. Many mapping operations are very common, so it makes sense to have a software library that includes these operations. Even for uncommon and more complex operations, having a library of mapping methods allows the instrument designer to rapidly test a wide variety of mapping ideas without having to implement them, and even derive some inspiration from scanning the contents of a mapping library and rapidly interchanging elements.

We are currently exploring the fundamental building blocks of mapping methods in the form of a software library for Pd. Through both research and practice, a wide range of objects have already been developed and are readily available for download as part of the 'Pd-extended' installers. The ultimate goal is to create a catalog of all of the fundamental mapping methods, and to try to reduce basically all mapping methods to a sum of these parts. This is inline with the unit generators that define the realm of audio synthesis and processing. We believe that it would not be appropriate for top-down design to dominate since there are so many possible methods, and the field of mapping for instrument design is still very much in development. Instead our process is akin to scientific observation of natural phenomena. As we generate more and more mapping primitives, patterns arise in the collection. Then we tailor the design of these emergent patterns.

We have been generating objects for as many different mapping fundamentals as possible. We believe that through this process of exploration, categories of techniques will become apparent. There are two main routes through which we create new objects for this library: first, by using the Mapping Library in real world situations such as dance performances and instrument building; second, by researching the existing literature and creating well defined mappings using the primitives that we have. These processes rapidly highlight what is missing and what is useful.

We have focused on two key areas in the last year: curve functions and physical modeling. Curve functions are an indispensible building block in any mapping since linear relationships are highly unlikely to provide an expressive mapping. Physical models allow the instrument designer to both emulate existing models such as the violin, or use physical

Pd Convention, August, 2007 Montreal, Quebec, Canada

processes to add a more natural and expressive feel to the situation. Curving functions can require relatively complex math, but the instrument designer need not fully understand the details of the math in order to effetively use the curve objects. For most people interested in instrument design, the math behind physical modeling is even more complex than for curving functions. Encapsulating all of these ideas into software objects makes mapping much more accessible, and provides a rapid prototyping environment for those designers who are well versed in the math behind these methods. Perhaps most importantly, having these methods encapsulated as software enables the ability to play with the objects in order to derive a mapping with much the same spirit that a musician plays with an instrument in order to learn it.

2. PREVIOUS WORK

Steiner started his work on the topic of mapping with the `[hid]` toolkit [12], which mainly focused on streamlining the process of getting data from game controllers for creating instruments. The `[hid]`[1] object already provides access to a wide range of devices. In addition, there is work underway on supporting sensorboxes in a way that follows standards laid out by the `[hid]` toolkit objects. The other key part of that work is the objects for mapping data. Henry started his work on mapping software when he developed a collection of Pd objects for sensor processing techniques. He has broadened his focus to include more related topics and explored these ideas further in his own performance with the group *chdh* [1] and in work with other artists.

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." [5] An earlier attempt from IRCAM is the ESCHER toolkit for jMax [13] which is a set of objects to address various problems of mapping. Goudeseune presented his mapping technique based on high dimensional interpolation he calls "simplicial interpolation" [7] . While this is an interesting technique that shows promise, it only addresses a specific approach to mapping and is not broken up into more generally useful modules. Both of these are complex systems which show promising results, but they seem to address the opposite end of the spectrum that this paper is addressing. They are built from complicated objects and take a mathematical approach to mapping. While mathematics are an integral part of mapping, the user need not experience it in that way.

3. DESIGN IDEAS

3.1 Encapsulation in Software

When talking about mapping, we are almost always talking about computer software, anything from custom C code to Pd patches to application preferences. Therefore it makes sense to implement a catalog of mapping methods as a software library. Mapping is generally represented firmly within the realm of mathematics. By abstracting the math into software objects, mapping can be approached as a system of logic. Software derives its vast power from the encapsulation of ideas and the reuse of code. Many complex mapping algorithms can be encapsulated into software objects, opening up new opportunities for exploration. One need not understand much about an algorithm within an object in order to insert it into a program and play with it. This way of interacting is much more like how many musicians learn to play an instrument: they play with it and see what they can figure out. Having encapsulated software objects, mapping can be more in this spirit of play, rather than purely a separate, studied effort. Also, catalog of mapping primitives aids in teaching and standardized terminology allows more fluid discussion and exchange of mapping ideas. This paper covers explicitly defined mapping methods, where the instrument designer directly controls each aspect of the mapping. Other papers cover methods involving generative techniques or neural networks[6]. While such systems might be based on the same mapping primitives as other methods of mapping, it is difficult to derive the mapping since it is only represented internally to the map-generating process.

[1] a word in square brackets denotes a Pd object

3.2 Mapping Primitives

There is still a lack of a set of commonly agreed upon primitives for the building of mappings. There are many great ideas about mapping, but lack reusable implementations. The goal of the Mapping Library for Pd is to first provide a set of mapping primitives, then to build more complex objects using the primitives. This modular approach has a number of advantages: the code is easy to read since its based on encapsulated ideas, code can be easily tweaked or repurposed since its all written within Pd, and as more objects get written, it becomes easier to write higher level objects. These objects are general purpose, making as few assumptions as possible for how they should be used. This idea is key to the design of Pd itself. TCP/IP was famously designed this way, and it has proven itself to be useful in ways far beyond the original creators' intentions.

In some ways, this library is a return to basics. Many interesting yet complex methods of mapping have been proposed and discussed. It seems that its too soon to be moving onto such complex methods when the basics are not clearly established. Software has become an integral part of designing new instruments, and it is rare for a new instrument these days to have absolutely no software component. Beyond writing about techniques of mapping, it should then be codified in software. Not only is software functional, but it is also a highly effective method of communicating logical ideas, arguably more effective that written language. Much how the standard audio unit generators encapsulate the mathematics used in synthesizing audio, ideas of mapping should also be laid out and encapsulated.

3.3 Data Range

As with the [hid] toolkit, the Mapping Library uses the data range of 0 to 1 wherever possible for the reasons outlined in the paper on that software[12]. Almost all of the mapping objects expect input and output data in the same range. This range is applied everywhere wherever possible, including to somethings that might disturb mathematicians, like [polar], which converts cartesian coordinates to polar coordinates. Even the angle is scaled to the range of 0 to 1. This allows other mapping objects to be used after the conversion without having to rescale the data. Though the objects are designed to work within this range, many of them

also work beyond that range. For example, [spiral] has an infinite range, with 1 representing one full rotation.

4. EXAMPLES OF MAPPING OBJECTS

There are already quite a few objects implemented, here are some selected examples: control rate filters ([iir], [fir], [lop]); basic transfer functions ([breakpoint], [quartic], [distance]); interpolation ([wave], [interpolate]); sensor data conditioning ([hysteresis], [local_min], [median_n]); testing ([test_n], [box], [stream_presense]); ranging and sizing ([resample], [upsample], [downsample]), and, dividing and joining data ranges([join]/[disjoin], [segment], [split_n]). Scaling is common operation, so there is [autoscale] which dynamically scales input data to an output range, [notescale] which scales 0 to 1 to the specified range of integer note values. Other ranging objects include [local_min], [local_max], [min_n], and [max_n]. Objects with the "_n" suffix mean that they take an numeric argument which control how many previous values that object should consider (i.e. apply the function to n elements).

The mapping objects are built from the ground up from quite primitive operations into higher level objects. For example, the [spiral] object is built using the [polar] object, which is in turn built using the [vector] object, which uses [radians->mapping]. The objects names have been carefully chosen to accurately represent the idea with a minimum of confusion. Some unusual words are used, like [disjoin], because it makes sense with its opposite operation: [join]. This is just a fledgling effort, so the names are bound to change as things develop.

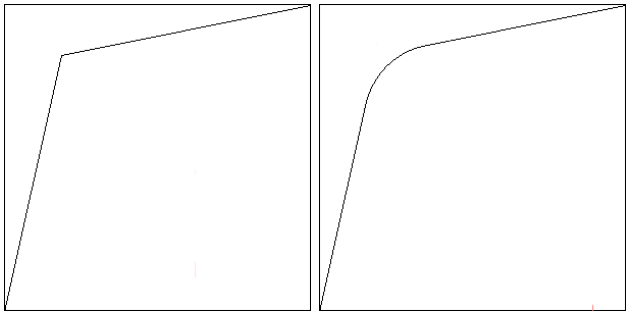

Figure 1: example output of [breakpoint] and [circular_breakpoint]

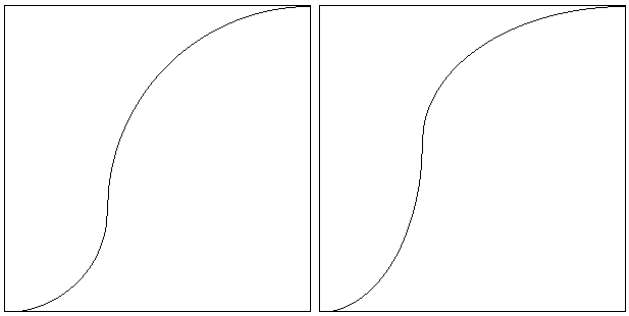

Figure 2: example output of [circular_sigmoid] and [elliptic_sigmoid]

5. CATEGORIZING CURVES

Curving functions are a fundamental aspect of mapping. Rarely does the data from a controller have a shape that produces the desired result. On the most basic level, linear controller data is usually shaped using a logarithmic curve when controlling amplitude and pitch since humans perceive both on a logarithmic scale. In many situations, more complex curves provide more satisfying results, especially when working with sensors whose data output have unusual shapes, or when trying to emulate a physical phenomenon. There are a handful of fundamental shapes that arise repeatedly in a wide range of mapping methods. At the most basic level, the shape of the simplest curves can be categorized as linear, circular, elliptical, etc. The most basic shapes of the curves can be described as breakpoint, seat, and sigmoid. [10] Sigmoid is a mathematical term for this shape, meaning "sigma-like". We also considered the term s-curve since the shape seems much closer to an S than a ς (sigma), but decided on sigmoid since it read more clearly in the object boxes.

There are some that do not seem to fit neatly into these categories, such as linear lines with elbow breakpoints or multi-point bezier curves. In the mapping library, the software objects are named for the fundamental shape of the curve. These categories are then used as keywords in the construction of the names of the software objects representing each possibility. [circular], [elliptic]. More examples are [circular_sigmoid] and [elliptic_sigmoid], which implement an sigmoid, the first with circular arcs, the second with elliptic arcs. When applicable, the depth of the curve is then a controllable parameter.

The goal with each curve object is to have a narrowly defined curve shape that can be controlled by a minimum of dimensions. The idea is that the instrument designer will mostly have a good idea of the general shape of the curve needed in a given situation. Once the curve is selected, it is easy to change the magnatude of that curve within a limited number of dimensions. Most of the curve objects have only one or two dimensions of control. Many useful curves cannot be reduced to one or two dimensions of control, so exceptions to this rule are required. An example of that is a linear breakpoint with a knee made with a circular arc. For that object to be effective, the mapping designer will want to control the x,y location of the breakpoint and the radius of the curve. When working with more complicated multi-point curves, which are rarely used in instrument mapping, it is necessary to have more control dimensions. For arbitrary curves, there is a distinct object, [curve_graph], which holds a graphical table that can be freely modified.

Keeping with the rest of the Mapping Library, all input and output data ranges are between 0 and 1, except when representing something with a data range whose limit cannot be meaningfully defined. While in the realm of math, the shape of an elliptical curve will never be a right angle, in the realm of instrument mapping, there is a relatively obvious point where the elliptical curve is perceived as a right angle. This point is then defined at 1. Then the point where the curve is perceived as linear is marked as 0.

6. PHYSICAL MODELING

From a general point of view, traditional acoustic instruments offer the musician complex interactions with the sound production mechanism. It is possible to model these interactions on the computer. We focus especially on continuous excitation-based instruments, such as the violin, clarinet etc. as this class of instruments offers a very wide range of expressive control. On a violin for example, the musician can continuously modulate the frequency, timbre and amplitude. The sound produced by this kind of instrument is not only a function of time, but is also affected by the bow pressure on the string, the position of the bow on the string, the bowing velocity, and the pressure and position of the fingers on the finger board. All these control parameters affect the sound through a complex, interrelated function. This complexity provides the experienced instrumentalist with a high level of expressivity. On the other hand, most electronic synthesis instruments have quite simple interrelationships,which limit musical expressivity. Control parameters are usually defined by the synthesis method rather than the gestural interface.

Our goal is to allow the creation of electronic instruments with high level of expressivity. This means that the mapping layer is more complex. Such a mapping layer was described by A. Momeni and C. Henry [11]. The main idea is to offer the instrumentalist control over a interrelated system defined by the desired gestures which then drives the audio synthesis.

The physical modeling mapping object collection introduces basic behaviors derived from simple mechanical systems. They provide basic objects that can be used to create a custom mapping layer. These patches are made using a mass-spring system offered by the *msd* library [3]. They must be controlled by a continuous data stream since the expectation is for the performer to control the evolution of the sound over time. The user interacts with virtual masses connected through visco-elastic links in different topologies derived from real world examples. Input of this system is usually the position of a specific mass, or a force sent to this mass. Output is typically the position of one or more masses. This collection of objects is described according to their behavior. Each class of objects offers a specific behavior, like simple attraction, repulsion, hysteresis etc. This behavior can be adjusted with "physical" parameters such as damping or rigidity of links.

These objects all incorporate inertia into the equation, thus time dependency is an inseparable aspect of the instrument. The output is not only a simple function of the input, but also a function over time, which can change even in the gestural input has not changed at that instance. We believe this creates a good platform for complex mapping between instrumentalist and audio synthesis.

7. EXAMPLES USING THE *PM* OBJECTS

- The `[pmattractor_2d]` object implements a system of 1 single mass connected to the (O,O) point. A 2-dimensional input (X and Y) influences the position of this mass. The output of this system comes from one stable point. It can be used by example to create some kind of envelope generator. This envelope can be modulated over time by the instrumentalist while the system is actively changing based on inertia and other forces.
- The `[pmrepulsor_2d]` object implements a system based on a single unstable point. The input acts like a repulsive force on the output mass. This object can be used to control synthesis using (0,0) as a "divergence".
- The *pmcloud* series of objects (1, 2 or 3-dimensional) provides different masses attracted by the position of the input mass. All the masses repulse each other so they create some kind of cloud of masses around a reference position. The output of this object is the position of all of this masses. So it can be used to generated a lot of data with little input. The relationship between all of this data can be adjusted the physical characteristics of the link between all masses.
- The `[pmmultistable]` object also provides important opportunities for expressive mappings. It provides a system of multiple stable points. Switching from one of these points to the other is achieved via a hysteresis process. Hysteresis exists in a few acoustic musical instrument. For example, changing the pitch in a flute while increasing the air flow is a process involving hysteresis.

Mixing two or more *pm* objects can create a more complex mapping layer with a wider range of results. The output of one object can be used as input for a second one, and so on, (i.e. serial connection). The objects can also be used in parallel to control different parameters of the audio synthesis starting from the same input signal.

8. EXAMPLES OF MAPPING WITH THIS LIBRARY

To explore these ideas, the mappings of two specific instruments are analyzed and reproduced using the Mapping Library for Pd. There is a huge variety of new instruments and wide range of ideas for mapping. These instruments where chosen because they have been played extensively, performed in concert, and each instrumentalist has achieved a high level of fluency with his instrument. Instruments that are regularly played will have a more honed mapping, and the spotlight of performance is excellent at drawing exposing problems.

8.1 The Ski Angular Mode

With Huott's Ski [8], he outlines four different modes for mapping the tactex pads to controlling samples: linear, polar, angular, and linear velocity. All of these can be easily implemented using exisiting mapping objects. For example, here is how to implement angular mode. First, `[autoscale]` automatically scales in the incoming data to a floating point range of 0 to 1. The scaled cartesian coordinates from the tactex pad are then fed to the `[polar]` object, which converts the data to a magnitude and angle. The angle is output in the range of 0 to 1 instead of the more usual -π to π. The allows other mapping objects to be easily chained after the `[polar]` object.

8.2 The Ski Pitched Mode

The front pads are used in a pitched mode, this layout is diagramed in figure 4 of Huott 2002. First, the pads are

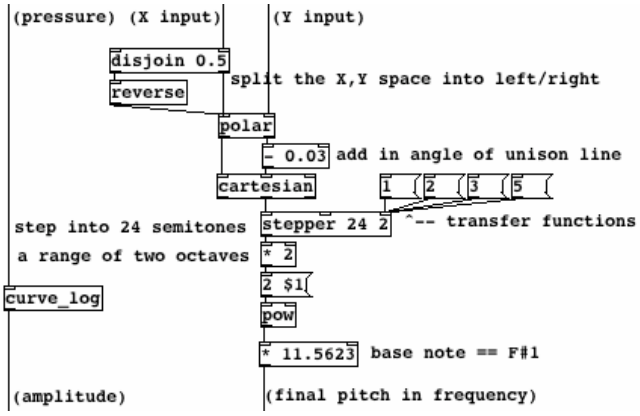

Figure 3: A Pd patch showing the mapping of The Ski's pitched mode.

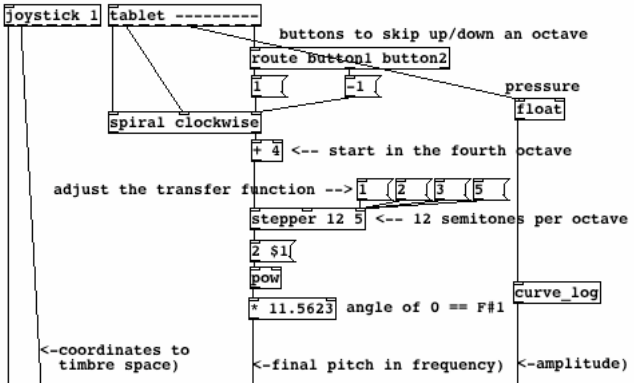

Figure 4: A Pd patch showing the mapping of the Voicer.

split into left and right sides using `[disjoin]`. The left side is reversed using `[reverse]`. The cartesian coordinates are converted to polar, then the angle of the unison line is created by subtracting 0.03 from the angle (represented from 0 to 1 not -π to π). The data is then converted back to a rotated cartesian plane, and the Y value is taken to control frequency. The range is split into 24 semitones and a x^2 transfer function is applied to allow glissando between notes. The range is multiplied by 2 to span 2 octaves, then converted to frequency with a base note of $F^{\#}1$. The pressure data is curved logarithmically to control amplitude. Figure 3 shows this patch.

8.3 The Voicer

Kessous developed his Voicer [9] using a tablet and a joystick as controllers. The tablet controls the pitch and amplitude while the joystick navigates a timbre space of vowel sounds. Pitch is derived from the polar angle, in a spiral. The `[spiral]` object does this using `[polar]` to convert to polar coordinates, then it keeps track of rotations. Since the `[spiral]` object has a "clockwise" argument in the patch, the data increases in the clockwise direction, rather than counter-clockwise as is usual with polar coordinates. `[stepper]` converts the linear angle data from

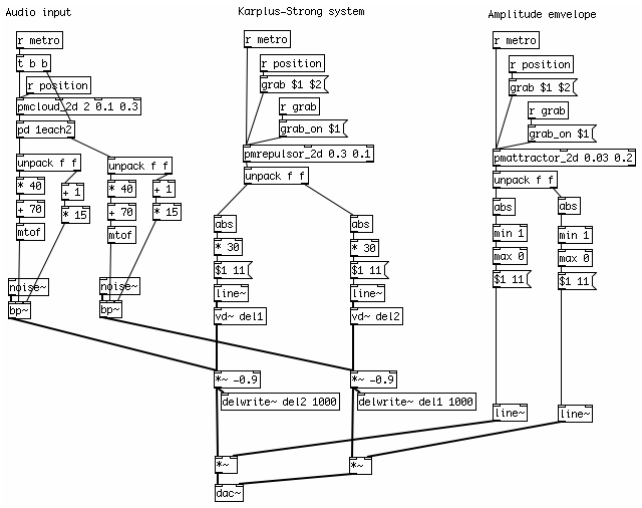

Figure 5: an example instrument using physical modeling mapping objects

`[spiral clockwise]` into a stepped line. The first argument of 12 creates 12 steps within the range of 0 to 1. Each step locally curved according to the transfer function specified by the last argument/inlet. This curved by taking the input and raising it to the power specified by the transfer function argument/inlet. The output of `[stepper]` is then converted into frequency values, with an angle of 0 equal to the beginning of the $F^{\#}4$ segment. In Arfib, et at, 2002, figure 5 displays a graph of the output of the `[stepper]` object. It is built using other Mapping Library objects: `[segment]`, `[curve_power]`, and `[desegment]`. `[segment]` is in turn built using `[disjoin]`, and `[desegment]` is built using `[join]`. For the amplitude control, the pressure in taken from the pen using the `[tablet]` object, which outputs all axes in the range of 0 to 1. The pressure data is then curved logarithmically using `[curve_log]`, to match the human perception of amplitude. Figure 4 shows this patch.

8.4 Physical Modeling

In the example patch shown in figure 5, two-dimensions of user input data control different part of the patch. First, a `[pmcloud_2d]` object is used to generated two different two-dimensional data streams. These two streams are used to control two instances of the same subtractive synthesis algorithm. This audio material is sent to separate audio delays looped in a Karplus-Strong topology. The delay times are controlled by the output of a `[repulsor_2d]` object. The (0,0) point corresponds to a singularity in the audio process (zero delay will cause a null loop), but also in the *pm* object as it acts like a repulsion of the output mass. Finally, the amplitude of the sound coming from the Karplus-Strong loop is modulated by a `pmattractor_2d]` object. When user does not send any data, the envelope goes to zero, and the sound stops. This very simple instrument offers a complex interaction with the sound synthesis thus allowing for more expressive audio exploration from the musician. Although the patch is relatively simple, it allows a complex interaction with the audio synthesis.

8.5 Sensors and Mapping

Figure 6: an dance performance using the Mapping Library to process sensor data

A real world example can be found in the most recent *Cie Contour Progressif* [2]performances, directed by Mylne Benoit. In these performances, dancers were wired with flex sensors and accelerometers. The sensors were connected to a wifi sensors box and sent to a computer over UDP. Using objects from the Mapping Library, the data from these sensors was then processed and used to control the sound and the lighting. The mapping strategy involved a number of steps. First, the sensor data was calibrated, then it was subjected to high pass filtering and differentiation to provide a more usable data stream. Some chunks of the data were broken out for various triggers. The data stream was then curved to match the dancers perception. Ultimately, this data stream then controlled different kinds of physical model filters to generate three data streams from a single input. These three streams then controlled the output amplitude, and the cutoff frequencies and Q of a bandpass filter applied to a noise source.

9. CONCLUSION

Starting with the initial effort working with ideas in the abstract, this library has been expanded and refined through both real world use and the analysis of existing instruments. This process has affirmed many of the existing ideas in the Mapping Library for Pd, and has exposed a number of weaknesses and omissions. Many interesting approaches to mapping, such as a multi-layered approach discussed by Arfib[4], Kessous, Wanderley, do not seem inherently compatible with these current objects. This is just the beginning, but this work has clearly demonstrated that there is promise to this approach to mapping. With a flexible toolbox of mapping methods, instrument designers can experiment more fluidly with mapping ideas. With a catalog of mapping methods, we can more easily discuss new mapping ideas and demonstrate them using code. This paper is another step in the process of identifying the fundamental "unit generators" for mapping. This process is strongly tied to producing a functional software library since ultimately instrument mapping is a question of software. We hope that this work will foster the conversation about what are the fundamental building blocks of instrumental mapping. A number of these primitives are showing up repeatedly while others are starting to fade as we introduce new ideas for primitives.

10. FUTURE WORK

Now that the basic foundation has been laid, more objects will be created to work towards completing a catalog of sensor processing and mapping methods. Specifically, we plan on applying this approach to more realms of mapping outside of what we have covered so far. This will lead away from a focus on music and hopefully towards a more generally useful library, contributing towards visual instruments, mapping for interactive installations, robotics, or whatever needs data mapped to controls. After a broader survey of mapping, the next phase will be to begin reducing the collection of objects to the fundamentals and applying these fundamentals to a broad range of mapping techniques.

11. REFERENCES

[1] chdh. `http://chdh.net`.

[2] Cie contour progressif. `http://mylene.benoit.free.fr`.

[3] Mass-spring-damper library. `http://nicomon.basseslumieres.org/index.php?id=7`.

[4] D. Arfib, M. Couturier, L. Kessous, and V. Verfaille. Strategies of mapping between gesture data and synthesis model parameters using perceptual spaces. *Organised Sound*, 7(2):127–144, 2002.

[5] F. Bevilacqua, R. Müller, and N. Schnell. MnM: a Max/MSP mapping toolbox. In *Proc. of the Conference on New Interfaces for Musical Expression (NIME05)*, Vancouver, Canada, 2005.

[6] 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)*, Hamamatsu, Japan, 2004.

[7] C. Goudeseune. Interpolated mappings for musical instruments. *Organised Sound*, 7(2):85–96, 2002.

[8] R. Huott. An interface for precise musical control. In *Proc. of the Conference on New Interfaces for Musical Expression (NIME02)*, Dublin, Ireland, 2002.

[9] L. Kessous. Bi-manual mapping experimentation, with angular fundamental frequency control and sound color navigation. In *Proc. of the Conference on New Interfaces for Musical Expression (NIME02)*, Dublin, Ireland, 2002.

[10] G. Levin. Signal shaping functions. In C. Reas, editor, *The Processing Book*. O'Reilly, 2007.

[11] A. Momeni and C. Henry. Dynamic independent mapping layers for concurrent control of audio and video synthesis. *Computer Music Journal*, 30(1):49–66, Spring 2006.

[12] H.-C. Steiner. [hid] toolkit: a unified framework for instrument design. In *Proc. of the Conference on New Interfaces for Musical Expression (NIME05)*, 2005.

[13] 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`.