FLEXIBLE CONTROL OF COMPOSITE PARAMETERS IN MAX/MSP

Timothy Place,[a] Trond Lossius,[b] Alexander Refsum Jensenius,[c] Nils Peters[d]

[a] Electrotap, tim@electrotap.com
[b] BEK - Bergen Center for Electronic Arts, lossius@bek.no
[c] University of Oslo, a.r.jensenius@imv.uio.no
[d] CIRMMT, McGill University, Montréal, nils.peters@mcgill.ca

ABSTRACT

Fundamental to the development of musical or artistic creative work is the ability to transform raw materials. This ability implies the facility to master many facets of the material, and to shape it with plasticity. Computer music environments typically provide points of control to manipulate material by supplying parameters with controllable values. This capability to control the values of parameters is inadequate for many artistic endeavors, and does not reflect the analogous tools and methods of artists working with physical materials.

Rather than viewing parameters in computer-based systems as single points of control, the authors posit that parameters must become more multifaceted and dynamic in order to serve the needs of artists. The authors propose an expanded notion of how to work with parameters in computer-centric environments for time-based art. A proposed partial solution to this problem is to give parameters additional properties that define their behavior. An example implementation of these ideas is presented in Jamoma.

1. INTRODUCTION

Presets and *automation* in computer music systems can be considered possible archetypes of strategies for dynamic control of a system. Presets in their purest form are a vertical-only approach; all values are instantly set to a certain state. Automation on the other hand, in its purest form is a horizontal-only approach; a fixed stream of time-tagged values progressing over a limited amount of time to control the state of one parameter, often with interpolation from one value to the next. While presets are widely used in real-time signal processing environments, the use of automation is fundamental to linear time-based media software such as digital audio workstations and video editing software.

One obvious way of expanding the flexibility of presets is by implementing a cross-fade or gradual transition to the new preset by means of interpolation. Several works have expanded this further by presenting the set of presets as points in a dataspace and developing strategies of traversing that dataspace, creating dynamic interpolations between two or more presets [1, 3, 8]. This has also been extensively used by one of the authors for developing the Hipno audio plug-ins [11].

Jamoma [1] is a system for developing high-level *modules* in the Max/MSP/Jitter environment [9]. It implements a *Model-View-Controller* (MVC) strategy, where "objects of different classes take over the operations related to the application domain (the model), the display of the application's state (the view), and the user interaction with the model and the view (the controller)." [6, p. 26]. All state management, parametric control, and automation for Jamoma is handled within the controller layer of the MVC paradigm. This forms the basis of all relationships both within a module and between different modules.

In Jamoma we are currently working towards more complex transitions of parameters in time that integrate both vertical and horizontal qualities. This is achieved through the integration of a cuelist system. This system permits instant updates to parameters, or scripting of complex transitional progressions, introducing horizontal aspects.

Previously, Jamoma offered possibilities of *ramping* to a new value over a certain amount of time by means of linear interpolation only. Recently this has been expanded by re-implementing ramps as a combination of two new libraries.

2. THE COMPOSITE PARAMETER

The parameter is the primary interface for a user manipulating the state of a module. In most systems, the parameter has a single task: to set a variable or coefficient. While it is straightforward to understand such a simple one-dimensional control, it does not offer the degree of nuance that, say, a sculptor has when working with clay.

In Jamoma, the parameter is expanded by adding *properties* and *methods* to the parameter that further refine or change its behavior [10]. These behaviors themselves can be in constant fluid motion together with the value of the parameter. Some examples of parameter properties include setting a value range, filtering out repetitions, determining the type of unit used to express values, and how automation is applied. The result is a composite parameter or *node*, made up of many constituent parts rather than representing only a single value. As such it is more like a multi-dimensional tool than a single point of control.

---

[1] http://www.jamoma.org

2.1. Properties and Methods

We have stated that a parameter may be enhanced by the addition of *properties* and *methods*. A property is an aspect of the parameter which itself has a state. For example, filtering of repetitive values can be turned on or off. A method is simply a mechanism for doing something, such as refreshing the user interface for the parameter. A method, however, does not have any value to maintain.

One interesting aspect of properties, which does not apply to methods, is that properties may themselves have properties, as illustrated in Figure 1.
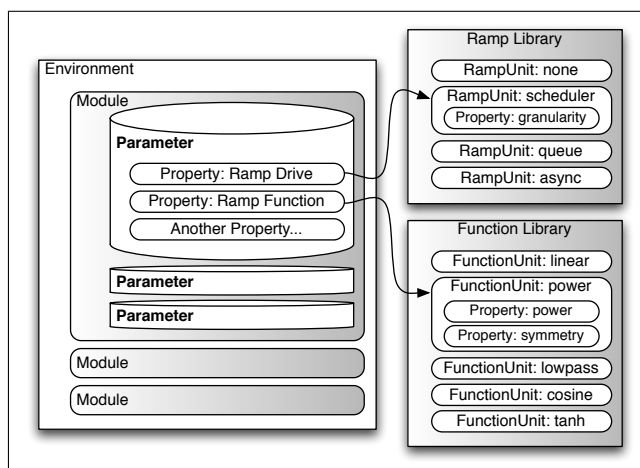

Figure 1. Parameter structure in context: Within an environment, there may be many modules. Each module may have many parameters. Each parameter may have many properties. A property may point to a dynamic entity which itself has properties, and so on.

2.2. Parameter Properties in Jamoma

Jamoma's parameter object is an implementation of the idea that properties and methods can meaningfully extend parametric control. When communicating to and from modules using the Open Sound Control protocol [13], we use the colon separator to access the properties of the parameter as proposed in [10]:

```
/path/to/parameter <value>
/path/to/parameter:/property <value>
```

Table 1 lists the currently implemented properties of the parameter object, with the path to the parameter omitted.

3. IMPLEMENTATION

In Jamoma, the parameter is implemented as a Max external called *jcom.parameter*. Within jcom.parameter, the ramping properties are implemented internally as a combination of two shared libraries called the *RampLib* and *FunctionLib*. The RambLib determines *when* a new value

Property or Method	Description
`:/value`	Value of the parameter
`:/value/stepsize`	Size of step taken `inc` and `dec`
`:/value/inc`	Increase the value
`:/value/dec`	Decrease the value
`:/value/default`	Initial value
`:/type`	Type of data (int, float, etc.)
`:/priority`	Order for recalling values from a preset
`:/ui/freeze`	Stops GUI updates to save CPU
`:/ui/refresh`	Updates the GUI
`:/ramp/drive`	Timing mechanism for ramps
`:/ramp/function`	Interpolation shape for ramps
`:/repetitions`	Filter out repeated values
`:/range/bounds`	Set a low and high range
`:/range/clip`	What to do when the range is exceeded
`:/description`	Documentation

Table 1. Selected parameter properties and methods in Jamoma

is required during a ramp, while the FunctionLib determines *what* the new value will be. Both of these are re-configurable on-the-fly during performance.

3.1. The Ramp Library

Depending on the circumstance, it may be desirable to generate new interpolated values in different ways during the ramp. Several real-time signal processing environments distinguish between audio rate and control rate signals [2, 7]. If the parameter is controlling a video processing algorithm it may be sufficient to update the value once per processed video frame [5].

The Jamoma RampLib provides a means by which to create and use *RampUnits* in Jamoma. A RampUnit is a self-contained algorithm that can slide from an existing value to a new value over a specified amount of time according to a timing mechanism. RampUnits are implemented in C++ using the TTBlue API [2]. Currently four such RampUnits are implemented:

- *none* - jumps immediately to the new value. Typically used for values where ramping is not relevant or desirable.

- *scheduler* - uses the Max internal clock to generate new values at fixed time intervals. The timing granularity can be controlled using a property.

- *queue* - ramps using the Max queue, updating values whenever the processor has free capacity to do so.

- *async* - only calculates new values when requested to do so. This is typically used in video processing modules to calculate fresh values immediately before processing the next video image or matrix.

[2] TTBlue is an object-oriented, reflective, application programming interface for C++, with an emphasis on real-time signal processing. http://code.google.com/p/ttblue

Figure 2. The possible ramping configurations in Jamoma can be represented as the intersection of a choice on each of the x and the y axes.

The RampLib can easily be extended with more Ramp-Units, and one planned extension is the implementation of audio rate ramping.

When a new ramp is started, the RampUnit internally uses a normalized ramping value increasing linearly from 0.0 to 1.0 over the duration of the ramp. Whenever the RampUnit is to provide a new value, it updates the normalized ramping value and passes it to a FunctionUnit as described in Section 3.2. The normalized value is then returned and scaled to the range defined by the start and end values for the ramp, and passed on to the module.

3.2. The Function Library

The Jamoma FunctionLib API provides normalized mappings of values $x \in [0, 1]$ to $y \in [0, 1]$ according to functions $y = f(x)$. Currently five functions are implemented:

- Linear: $y = x$

- Cosine: $y = -\frac{1}{2} \cdot cos(x \cdot \pi) + \frac{1}{2}$

- Lowpass series: $y[n] = y[n - 1] \cdot k + x[n] \cdot (1 - k)$
  The feedback coefficient k can be set as a property.

- Power function: $y = x^k$.
  The parameter k can be set as a property.

- Hyperbolic tangent: $y = c \cdot (\tanh(a \cdot (x - b)) - d)$
  The width and offset of the curve can be set as properties. These in turn set the values of the coefficients a, b, c and d.

The FunctionLib can easily be expanded by introducing new functions as C++ classes, also using the TTBlue API.

3.3. Combinations

One of the advantages of implementing ramping as a combination of two libraries, is that any RampUnit can be combined with any FunctionUnit. Currently, with 4 RampUnits and 5 FunctionUnits implemented, this provides a total of 20 options for how to perform ramping, as illustrated in Figure 2.

4. DISCUSSION AND FURTHER WORK

The proposed system for the ramping of parameter values can be understood as an extension of the well-established ADSR (attack - decay - sustain - release) envelope used in classic synthesizers to create increasingly complex developments over time. Ramps are initiated and controlled by simple OSC messages, thus combining simplicity of access with complexity and expressivity of the result.

One Jamoma module, *jmod.cuelist*, loads a text-based script of *event cues*, and is able to control all other modules [9]. A `WAIT` syntax can set the execution of a cue on hold for a specified amount of time. Thus more complex auditive events can be created by combining parallel ramps for several parameters. Simultaneously the transitional curve for each of the parameters can be made more complex by building compound curves splicing together several ramp segments, where different functions can be used for each segment, as illustrated in Figure 3.

```
########################################
CUE upAndDown
########################################
/path/to/parameter 0.
/path/to/parameter:/ramp/function linear
/path/to/parameter 1. ramp 2000
WAIT 2000
/path/to/parameter:/ramp/function cosine
/path/to/parameter 0. ramp 8000
```
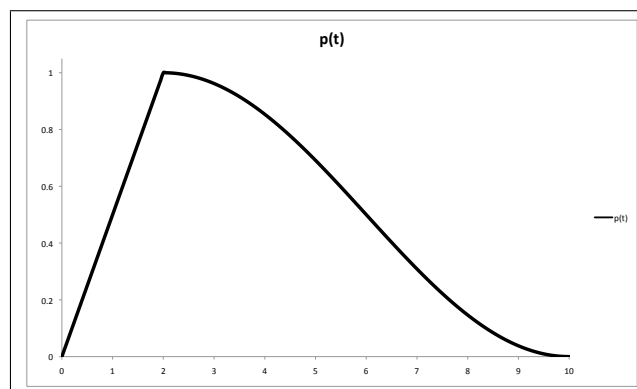

Figure 3. A simple cue script. The parameter first traverses linearly from 0.0 to 1.0 in two seconds, and then returns to 0.0 in eight seconds according to a cosine function

In the discussion thus far, ramps have been implicitly understood to be *goal-directed*. That is to say that ramps are moving towards a final destination in a fixed and limited amount of time. If this assumption is relaxed, the RampLib and FunctionLib implementations can be expanded further to provide even more possibilities for continuous movement of parameter values. For instance, low frequency oscillators can be implemented as a sequence of repeating and possibly reversed ramps. If the FunctionLib is expanded by introducing stochastic functions, and ramps are permitted to be of infinite duration, the RampLib can be used to trigger new random values when re-

quired. This leads to different types of stochastic drifts and processes in time.

Presets and automation have been mentioned in Section 1 as archetypes for strategies of dynamic control in a system. A third possible archetype for controlling a system is the use of mappings. Mappings define relationships between various components in a system that interact with each other. This creates a complex and dynamic methodology for generating not only vertical transmogrifications but gestures which develop over time [4, 12].

The FunctionLib can also be used outside the context of jcom.parameter. The *jcom.map* Max external maps values in the input range $[a, b]$ to values in the output range $[c, d]$. The FunctionLib is used to determine the curve which shapes the mapping. This curve can be changed on-the-fly by switching between any of the available FunctionUnits.

In a similar way the RampLib can be applied outside the context of jcom.parameter for other scheduled tasks.

5. ACKNOWLEDGMENTS

6. REFERENCES

[1] R. Bencina. The metasurface – applying natural neighbour interpolation to two-to-many mapping. In *Proceedings of The 2005 International Conference on New Interfaces for Musical Expression (NIME05)*, 2005.

[2] R. Boulanger, editor. *The Csound Book: Perspectives in Software Synthesis, Sound Design, Signal Processing,and Programming*. The MIT Press, 2000.

[3] P. Dahlstedt. Creating and exploring huge parameter spaces: Interactive evolution as a tool for sound generation. In *Proceedings of the 2001 International Computer Music Conference*, pages 235–242, Habana, Cuba, 2001. San Francisco: ICMA.

[4] A. Hunt, M. M. Wanderley, and M. Paradis. The importance of parameter mapping in electronic instrument design. *Journal of New Music Research*, 32(4):429–440, 2003.

[5] R. Jones and B. Nevile. Creating visual music in jitter: Approaches and techniques. *Computer Music Journal*, 29(4):55–70, 2005.

[6] G. E. Krasner and S. T. Pope. A cookbook for using the model-view controller user interface paradigm in smalltalk-80. *J. Object Oriented Program.*, 1(3):26–49, 1988.

[7] J. McCartney. Supercollider: A new real time synthesis language. In *Proceedings of International Computer Music Conference 2000. Hong Kong*, 16.

[8] A. Momeni and D. Wessel. Characterizing and controlling musical material intuitively with geometric models. In *Proceedings of the 2003 Conference on New Interfaces for Musical Expression*, pages 54–62, Montreal, Quebec, Canada, May 2003.

[9] T. Place and T. Lossius. Jamoma: A modular standard for structuring patches in max. In *Proceedings of the International Computer Music Conference*, pages 143–146, New Orleans, LA, 2006.

[10] T. Place, T. Lossius, A. R. Jensenius, N. Peters, and P. Baltazar. Addressing classes by differentiating values and properties in osc. In *Proceedings of the 2008 Conference on New Interfaces for Musical Expression*, Genova, Italy, 2008.

[11] T. Place, N. Wolek, and J. Allison. *Hipno: Getting Started*. Cycling'74 and Electrotap, 2005.

[12] D. van Nort and M. M. Wanderley. Exploring the effect of mapping trajectories on musical performance. In *Proceedings of Sound and Music Computing*, Marseille, France, 2006.

[13] M. Wright, A. Freed, and A. Momeni. OpenSound Control: State of the art 2003. In *Proceedings of NIME-03*, Montreal, 2003.

---

[3] http://imal.org/
[4] http://www.gmea.net