MorphOSC- A Toolkit for Building Sound Control GUIs with Preset Interpolation in the Processing Development Environment

Liam O'SULLIVAN
Electronic & Electrical Engineering, Trinity College Dublin
Dublin 2, Ireland
lmosulli@tcd.ie
https://github.com/LiamOSullivan/MorphOSC

Abstract

MorphOSC is a new toolkit for building graphical user interfaces for the control of sound using morphing between parameter presets. It uses the multidimensional interpolation space paradigm seen in some other systems, but hitherto unavailable as open-source software in the form presented here. The software is delivered as a class library for the Processing Development Environment and is cross-platform for desktop computers and Android mobile devices.

This paper positions the new library within the context of similar software, introduces the main features of the initial code release and details future work on the project.

Keywords

Toolkit, Processing Development Environment, Open Sound Control, User Interface, Preset Interpolation.

1 Introduction

The control of complex, dynamic sound typically involves manipulation of a large number of parameters. Complex mappings that link one-or-more input controls to one-or-more outputs have been seen to be more effective for the provision of engaging, expressive play than simple one-to-one mappings [6]. One approach to the control of multiple parameters in real time is the use of a multidimensional space superimposed on a two-dimensional graphical controller [10]. Particular settings for an ordered collection of parameters can be associated with anchor points on the controller surface and the movement of a cursor provides an interpolated output value for each parameter. The usefulness of such an approach for the provision of musical control has been noted previously in the above examples; although independent control over each output is compromised, an intuitive and 'playable' space is provided. This two-input to many-output (two-to-many) mapping can be well-suited for live performance or the exploration of timbre spaces generated when the interpolated output is sent to a synthesiser.

Although several systems provide a graphical user interface (GUI) to some implementation of such a scheme, they are usually tied to a particular application, are commercial products or are not portable to multiple platforms. To address this, a new code library is presented that facilitates rapid prototyping of interfaces utilising preset morphing- MorphOSC.

Section 2 of this paper briefly describes similar work in the form of existing GUIs that allow complex mappings through interpolated parameter spaces. The design goals for the new tool are then identified as a reaction to what is currently available. Some background to one method of parameter morphing is provided in section 3. Section 4 outlines the current library implementation, identifying key features of the software in its current state. Future areas of development are discussed in section 5 and final conclusions are made.

2 Similar work

Several software systems exist that facilitate the exploration of multidimensional parameter spaces. While some of these are sophisticated systems offering extensive functionality, it will be shown that a gap exists for the approach being outlined here due to the limitations described in each case. Previous work by the author describes a number of more general mapping interfaces [12] and will not be repeated in this paper, but salient examples of more general software controllers and specific interpolating interfaces are now presented.

2.1 Interpolating interfaces

The provision of effective control of computer music systems via preset interpolation has historically been of interest. As far back as the late 1970s, researchers at the Inaís Groupe de Recherches Musicales provided such functionality in the GUI component of the SYTER system [5]. Today, the real-time processing capability of desktop computers and even mobile devices means that interfaces can be implemented as components of a larger software system. The Max/MSP programming environment [9] is a popular example and provides many data-manipulation tools; a recent implementation of an interpolating controller for this environment is the *nodes* object. This allows many inputs to be weighted and combined to a single output based on the positions of overlapping circular graphical nodes. Similarly, the IRCAM MnM mapping toolbox, (part of the FTM external object library [3]) allows the user to build patches with existing Max/MSP GUI elements. For instance, an example patch allows the specification of two-to-many mappings using a two-dimensional controller and a set of linear sliders. The system can associate points on the controller with particular slider arrangements and value settings; moving between points provides a smooth morph between the sliders' states.

The MetaSurface is an interface for interpolating between parameter ‘snapshots’ for two-to-many mappings [1] and an example implementation is included with the AudioMulch software [2]. Still more recently, one project [8] provides a preset-interpolation interface for the SuperCollider environment [18], designed for use with a bespoke physical controller.

The above examples are part of more fully-featured programs rather than standalone controllers and/or are commercial products. They cannot be used with Android mobile devices. The ability to include subsets of output parameters in the interpolation space is provided in some cases, typically via check-boxes or a set-up dialogue. However, an interface which uses the drag-and-drop metaphor to manipulate parameter sets would provide a more interactive experience. The use of a multi-layered GUI approach would also allow more complex mapping relationships to be built and refined.

2.2 Standalone interfaces for Open Sound Control

Software controllers already exist for mobile and touch-screen devices that can output messages over a network formatted using the Open Sound Control (OSC) standard [11]. From simple applications like *andOSC* [12] to more sophisticated tools such as the popular *TouchOSC* [15], these offer real-time control of musical applications and exploit the multi-touch capability of contemporary phones and tablets (as well as additional sensor input from accelerometers etc.). Although functionality-limited free versions are available, they are not open source. Neither do they provide an interpolation surface, meaning this must be implemented on a networked computer if required. This separates the mapping configuration from the interface, inhibiting engagement and obstructing work-flow A more unified interface would facilitate greater exploration of the parameter space and dynamic mapping during performance.

2.3 Processing Development Environment

The Processing Development Environment (PDE) is an open-source initiative that attempts to make it easier for artists, designers and novice programmers to implement computer-based projects. It uses a streamlined form of the Java programming language and has evolved to become a very popular tool for creatives. Code libraries provide additional functionality such as enhanced interactivity and sound generation.

One such contributed library is the recent JunctionBox toolkit [4], which can provide interaction capability beyond the use of traditional controller widgets. Code 'sketches' written in Processing can include this library's functionality to produce OSC messages triggered by common mouse-based or multi-touch interactions (e.g. scaling, rotation etc.). As the PDE now supports rapid Android application prototyping[1], this allows easier implementation of novel OSC controllers for mobile devices. However, as the focus is on the generation of messages based on common spatial manipulations of graphical objects, it does not particularly address the production of more complex GUIs including preset-interpolation surfaces. Nevertheless, the library serves as a useful template for the provision of such functionality through a code library for the PDE.

2.4 Project goals

The design goals which emerge from the initial motivations for the project and subsequent consideration of similar work are as follows:

[1] As of version 2.0 beta 7, March 2013.

- Freely available, open source, cross-platform compatible toolkit for rapid prototyping of preset-interpolation interfaces.
- Interaction design exploiting familiar metaphors for intuitive configuration of the parameter interpolation space (e.g. drag-and-drop, layering).
- OSC-formatted output.

3 Interpolation methods

A full discussion of the various methods available for interpolation between a set of scattered data points is beyond the scope of this paper and the reader is directed to an overview from the field of cartography [7]. However, the techniques used in some examples of similar interfaces are summarised in table 1.

Software	Method
SuperCollider *PresetInterpolator*	Intersecting N-Spheres
Max/MSP *nodes*	Inverse Distance Weighting
AudioMulch/ MetaSurface	Natural Neighbour

Table 1: Interpolation methods used in some existing GUIs for musical control.

The need for real-time performance and the suitability of the software for mobile platforms prioritises the use of computationally inexpensive interpolation techniques. For the initial toolkit release, the method of Inverse Distance Weighting (IDW) was preferred.

3.1 Inverse-Distance Weighting

IDW is commonly called Shepard's Method following an early documentation of the technique [17]. In essence, it can assign values to unknown points by calculating a weighted average of the values at scattered sample points. The normalised distances from the interpolation point to the known values, $\mathbf{d_n}$, are used to scale the values of each parameter at these points, $\mathbf{p_{ni}}$, in an inverse relationship. The results are then averaged, meaning points further away have less effect on the interpolated value of a particular parameter. A general expression for the operation is therefore:

$$p_i = \frac{\sum_{n=0}^{k} p_{ni} d_n^{-1}}{\sum_{n=0}^{k} d_n^{-1}}$$

modification to the technique uses the square of the distance involved and may be more suited to the control of musical parameters, due to the non-linear nature of certain aspects of human perception and experience of the real world (e.g. inverse-square law attenuation of sound with distance). IDW considers all points on the surface, but may also be modified to only consider the nearest points and reduce the computation required for interpolating the output. Figure 1 shows an interactive Processing sketch that outputs a set of interpolated values for a three-dimensional parameter space mapped to a two-dimensional controller surface[2]. This illustrates how parameters at the interpolation point (i.e. the output) are calculated from their ordered counterparts at the 'known' sample points (i.e. the anchor points).

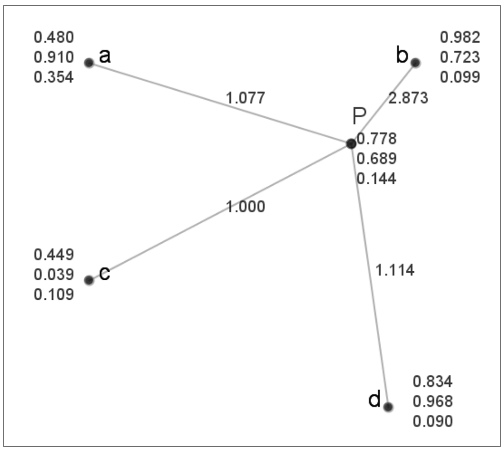

Figure 1: Inverse Distance Weighting used to interpolate values for the parameters at a point **P** from a set of scattered sample points **a, b, c** and **d**. The normalised values for the weights (inverse distances) are shown along the vector lines and three parameter values are placed at each point.

4 Implementation

The current toolkit was programmed in Java and is available as a library for the PDE. This includes example interfaces which can be loaded into the environment and modified, or exported to be run as standalone applications across multiple platforms (OSX, Windows, Linux, Android).

The toolkit builds on the functionality of other contributed libraries for Processing to allow easy

[2] The Processing code for this example is avaliable at: https://github.com/LiamOSullivan

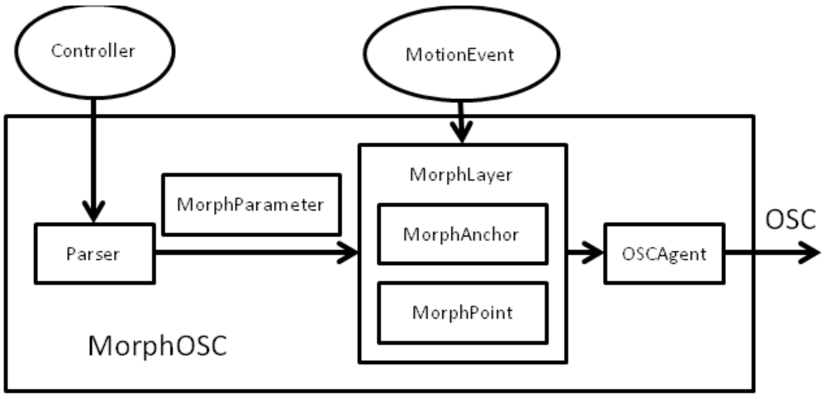

Figure 2: Overview of the core library classes (rectangular boxes) of the *MorphOSC* toolkit.

integration into the work-flow of developers and to keep the code base to a minimum. The library is design to make use of the popular ControlP5 [15] library for the provision of on-screen control widgets, while the OSC subsystem uses the oscP5 library [16] to format output appropriately.

Settings for the constructed GUIs can be stored and recalled using a preset file, formatted with extensible mark-up language (XML) for ease of portability.

Class	Description
MorphOSC	Base class, manages interaction space.
Parser	Parses subset of widget fields.
MorphLayer	Interactive GUI element. Container for (i), (ii), (iii).
(i) MorphAnchor	Holds a set of parameter values.
(ii) MorphPoint	An interpolation point.
(iii) MorphParameter	Parameter value parsed from widget.
OSCAgent	Formats outgoing messages.

Table 2: Core classes of the MorphOSC library.

The core classes that implement the MorphOSC library are listed in table 2. Figure 2 outlines their inter-relationships.

4.1 Usage

The library employs the conventions common to contributed libraries for the PDE, as shown in the example code of table 3. The base class for the library is instantiated in the usual way, by passing a reference to the parent PApplet (the encapsulating class for a Processing program). This effectively creates an interaction area at runtime with the same dimensions as the parent. Widgets are defined using the ControlP5 library to implement the interface design in the usual way. Any widgets which are to be included for morphing are then added to the MorphOSC instance using the **add()** method. This sends the element to the Parser class; a subset of the controller properties are extracted and a MorphParameter instance for each added controller is returned.

```
MorphOSC morph = new MorphOSC(this);
ControlP5 cp5 = new ControlP5(this);
Slider s = cp5.addSlider();
morph.add(s);
```

Table 3: Example Processing code. MorphOSC and ControlP5 base classes are instantiated. A slider is created and added to the MorphOSC object.

All other public classes are modified at runtime through interaction with the GUI.

4.2 Interaction Design

Manipulation of MorphOSC elements through the interaction area depends on the current mode of the interface, which can be in *Edit Mode* or *Performance Mode*.

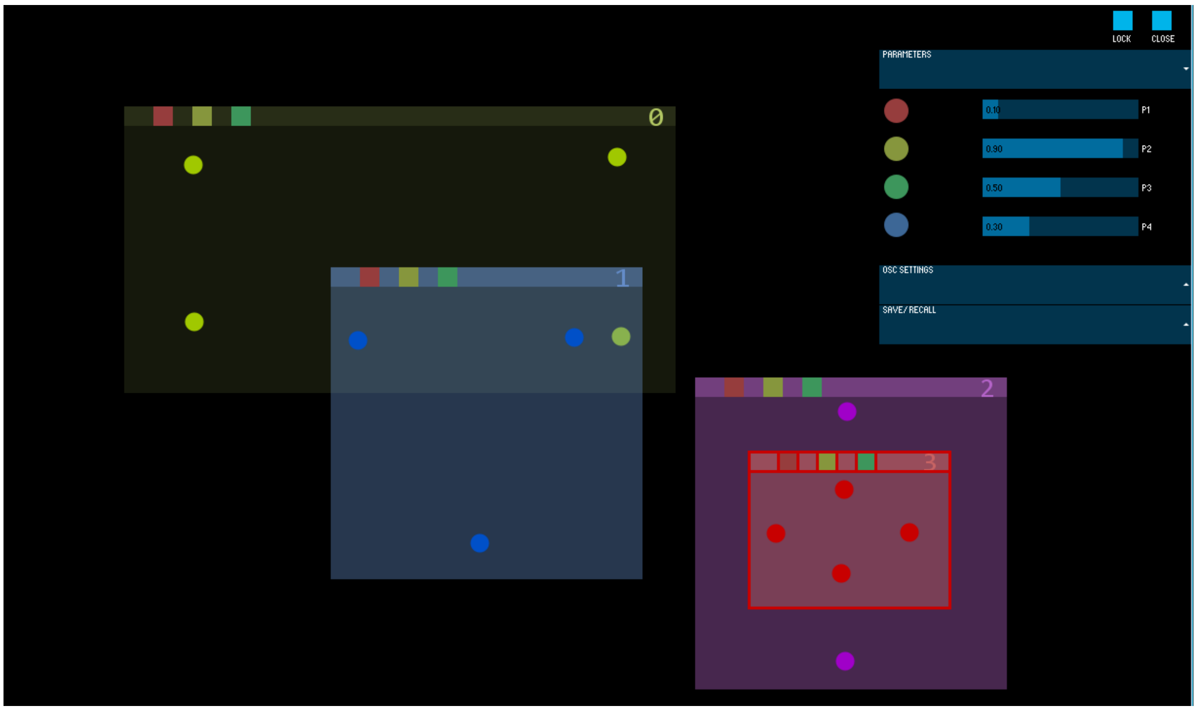

Figure 3: An example GUI created in Processing with MorphOSC and ControlP5. In the unlocked *Edit Mode*, GUI elements may be modified but interpolation output may still be auditioned in real time. MorphLayer number 3 is in focus and the various widgets for system settings are visible on the right hand side of the screen.

4.2.1 Edit mode

When unlocked in *Edit Mode*, MorphOSC elements may be created, modified and destroyed. For example, a MorphLayer can be instantiated with an event (e.g. a mouse click or a double finger-tap) in free space on the interaction area. Layers may subsequently be moved/ resized and overlapping is possible. Any widgets which have an associated MorphParameter are indicated in the GUI with a unique colour. A MorphAnchor may be added to a layer by using a drag-and-drop action from the numerical value attached to the corresponding widget. This adds the associated MorphParameter to the layer and initialises the MorphAnchor value for that parameter to the current widget value. Additional parameters may be added to existing anchors or anchors may have their values overwritten by subsequent drag-and-drop actions. Anchors may be moved about their layer to reconfigure the underlying interpolation space; finer control may be attained by moving anchors further apart, for example.

Edit mode produces interpolated values for parameters by dragging (with mouse or touch) in free space. This allows the user to audition parameter interpolation in real-time as they are manipulating them, but is not meant to provide a full performance mode.

4.2.2 Performance mode

When in the locked performance mode, MorphLayers, MorphAnchors and other instantiated classes cannot be modified other than through the specification of MorphPoints to generate interpolated parameter values. Interaction with the MorphLayers produces interpolated values for their associated MorphParameters based on the arrangement of MorphAnchors within them. MorphPoints interpolate values from all layers behind them. This means that subsets of parameters can be associated with different layers and spatial positions, providing a lot of flexibility in design of the control space.

Performance mode contains the option to hide all ControlP5 GUI elements so that the whole space is available for gestural input.

5 Conclusion and future work

This paper introduced a new toolkit to aid in the rapid development of GUIs utilising preset interpolation for the control of sound over OSC. A short review of similar work identified the need for a code library for the popular Processing environment, in order to allow cross-platform interface development. Following a brief discussion of a suitable interpolation method, the new toolkit- MorphOSC- was then introduced and key features were outlined.

The software is currently in beta version and there is much work to be done to produce a release candidate. A full evaluation of the system is required to assess stability and performance. Use of the system in a workshop setting is proposed to evaluate usability and performance is to be tested 'in the wild'.

The current system implements a simple averaging interpolation scheme through IDW, but as this can have some limitations (e.g. computation time proportional to the number of anchor points) other methods will be examined. It is envisaged that the toolkit will serve as a test bed for evaluating the effectiveness of various interpolation methods for the provision of real-time control of musical output.

This work forms part of a larger project which attempts to leverage the benefits of two-dimensional interfaces for musical control. The multi-layer paradigm is seen as a strong metaphor for the provision of intuitive interactions not currently supported in existing software.

Acknowledgements

Thanks to Andreas Schlegel for both the oscP5 and controlP5 libraries. The author appreciates the comments of the reviewers and the beta-testers for their invaluable feedback.

References

[1] Bencina, R. The Metasurface – Applying Natural Neighbour Interpolation to Two-to-Many Mapping. *Proceedings of the 2005 Conference on New Interfaces for Musical Expression (NIME'05)* (Vancouver, BC, Canada, May 26-28, 2005), 101-104.

[2] Bencina, R., AudioMulch interactive music studio. http://www.audiomulch.com/

[3] FTM & Co., IRCAM. http://ftm.ircam.fr/

[4] Fyfe, L., Tindale, A. and Carpendale, S. JunctionBox for Android: An Interaction Toolkit for Android-based Mobile Devices. *Proceedings of the Linux Audio Conference (LAC2012)*, (CCRMA, Stanford University, CA, USA. April 12-15, 2012).

[5] Geslin, Y., Digital Sound and Music Transformation Environments: A Twenty-year Experiment at the Groupe de Recherches Musicales. *Journal of New Music Research* 31(2): 99–107, 2002.

[6] Hunt, A. and Kirk, R., Mapping Strategies for Musical Performance. *Trends in Gestural Control of Music*, M. Wanderley and M. Battier, Editors, 2000.

[7] Lam, N., Spatial Interpolation Methods: A Review. *The American Cartographer*. 10(2): 129-149, 1983.

[8] Marier, M., Designing Mappings for Musical Interfaces Using Preset Interpolation. *Proceedings of the Conference on New Interfaces for Musical Expression (NIME '12)*, (May 21 – 23, 2012, University of Michigan, Ann Arbor).

[9] Max/MSP environment from Cycling 74. http://cycling74.com/products/max/

[10] Momeni, A., Wessel, D., Characterizing and controlling musical material intuitively with geometric models. *Proceedings of the 2003 conference on New Interfaces for Musical Expression* (NIME '03) (2003, National University of Singapore, Singapore), 54-62.

[11] Open Sound Control.
http://www.opensoundcontrol.org

[12] O'Sullivan, L., Furlong, D., and Boland, F. Introducing CrossMapper: Another Tool for Mapping Musical Control Parameters. *Proceedings of the Conference on New Interfaces for Musical Expression (NIME '12)*, (May 21 – 23, 2012, University of Michigan, Ann Arbor).

[13] Primevision andOSC Android application. https://play.google.com/store/apps/details?id=cc.primevision.andosc&hl=en

[14] Processing Development Environment. http://www.processing.org

[15] Schlegel, A., Sojamo ControlP5 Library for Processing.
http://www.sojamo.de/libraries/controlP5/

[16] Schlegel, A., Sojamo OscP5 Library for Processing.
http://www.sojamo.de/libraries/oscP5/

[17] Shepard, D., A two-dimensional interpolation function for irregularly-spaced data. *Proceedings of the 1968 23rd ACM national conference*, 517–524.

[18] SuperCollider. supercollider.sourceforge.net

[19] TouchOSC.
http://hexler.net/software/touchosc