1-D, 2-D AND 3-D INTERPOLATION TOOLS FOR MAX/MSP/JITTER

Todor Todoroff
Faculté Polytechnique de Mons (FPMs) - TCTS
Mons, Belgium
ARTeM asbl, Brussels, Belgium
todor.todoroff@skynet.be

Loïc Reboursière
Faculté Polytechnique de Mons (FPMs) - TCTS
Mons, Belgium
loicreboursiere@gmail.com

ABSTRACT

We propose new tools for interpolation in 1-D, 2-D and 3-D spaces. They were developed for the "Dancing Viola" project within the Numediart program and will be freely downloadable from www.numediart.org. The 1-D and 2-D representations work in Max/MSP using the *LCD* or *jit.window* objects, while the 3-D one requires Jitter. They introduce new features specially designed for interactive performances with sensor data. This includes the definition of areas centered on the interpolation points where the weights stay constant and different input messages for mouse actions and sensor inputs that offer the possibility of moving interpolation points with the mouse while receiving sensor data. This allows to tune the setup while receiving performer's data during rehearsals. But interpolation points may equally be moved and resized with Max messages, leading to a less usual way of increasing the input dimensionality by allowing sensors to move interpolation points, effectively modifying the interpolation space.

1. INTRODUCTION

Since interpolation between sets of parameters was proposed by Allouis [1] for the SYTER at GRM, several authors [6, 4, 2, 3] have developed various implementations. Interpolation has been recognized as a very intuitive way of performing various types of mapping that can very effectively be applied to various kinds of sound transformation algorithms. Most implementations are designed for mouse control, but we wanted to use data obtained by a wireless system developed at ARTeM, consisting of accelerometers and gyroscopes, for the "Dancing Viola" project, described in more details in [5]. We were missing some features that make it possible to cope with less precise control. Indeed, even with proper filters, data from accelerometers may still be a bit jittery when one wants a responsiveness that prevents the use of too strong median and/or low pass filters. We were also missing the absence of 1-D and 3-D interpolator implementations in Max/MSP. A 1-D interpolator proves to be very useful when one needs to place values at specific points of a one-dimensional sensor. Though one could argue that it is possible to achieve that result by splitting the range of that sensor equally in a certain number of portions and use a table to get the desired result, it can be a very time-consuming task. On the other hand, being able to place points exactly where desired is extremely efficient. If, for instance, one wants specific transposition intervals at several orientations of a limb, it is very straightforward to place points and assign values to them using a 1-D interpolator. And we show a system that makes it equally easy to define what kind of transition happens in between those points: steps or glissandi of various lengths. A 3-D interpolator has also an obvious interest when using a 3-D accelerometer. But those tools are not limited to sensor data control; they also provide the necessary flexibility to be controlled by sound analysis.

2. INTERPOLATION

Interpolation is an operation whereby each value of the output set of m values is the weighted sum of the n corresponding values in the n interpolated sets, with normalized weights W_{Ni}:

$$output\_val_j = \sum_{i=1}^{n} W_{Ni}\ \ val_{ij} \quad 1 \leq j \leq m \tag{1}$$

As such, the interpolator can perform the various types of mapping usually described in the literature (direct or one to one, divergent or one to many, convergent or many to one and many to many) depending on the number m of values in each set and on the dimensionality of the interpolation space, which is not to be confused with the number n of points placed within that space. We don't suggest that it should or could replace other mapping techniques, but its intuitiveness makes it a very valuable tool.

2.1. Gravitational field metaphor

Different radial basis functions (RBF) could have been used and may be added later. But we used, as Allouis [1], the metaphor of a gravity system where each of the n points can be considered as a planet which exerts an attraction force F on the cursor depending on their relative cartesian distance d, following Newton's law:

$$F_i = G\frac{m_{cursor}\ m_i}{d^2} \quad 1 \leq i \leq n \tag{2}$$

if we then consider that the weight associated to each point is proportional to this attraction force and that the mass m_i can be represented by the radius R_i of the point, we obtain the following equation:

$$W_i = \frac{R_i}{max(d_i, dmin)^{power}} \quad 1 \leq i \leq n \tag{3}$$

where *dmin* is a small number that prevents a division by zero when the cursor is located exactly at the center of the point i and $d_i = 0$. When $power = 2$, we get the Newton's law of attraction. In order to give maximum flexibility to the user, *power* can be modified with the message `power $1`, with a float argument. Changing this modifies the shape of the interpolation between points: softer in between points with low values of *power*, more abrupt with higher values.

$$W_{Ni} = \frac{W_i}{\sum_{k=1}^{n} W_k} \quad 1 \leq i \leq n \tag{4}$$

As the weights are normalized (4), we also provided the message `d_min $1`. Indeed, by limiting the highest reachable weight, one can globally define in how much, when the cursor is exactly centered on a point, its weight does or does not overshadow the other weights. Obviously, *dmin* cannot be ≤ 0 and has been limited inside the external to $10^{(\frac{-6}{power})}$. This insures that the maximum weights keep the same value at $10^6\ R_i$, regardless of the *power* factor.

2.2. Rmin

The interpolation tools are usually controlled with a mouse. As we wanted to use interpolation with sensors data input, where a precise location is quite difficult to attain by the performer, dancing on stage without visual feedback from the computer screen, we decided to define a zone of radius *Rmin* around the interpolation points where the maximum weight is always reached. This zone is a rectangle, a circle or a sphere depending on the dimensionality of the interpolation space: 1-D, 2-D or 3-D. A negative *Rmin* would on the other hand prevent the user from reaching the mathematical centre closer than the absolute value of *Rmin*. It has a similar effect as increasing the global parameter *dmin*, except that it can be individually adjusted for each point. This allows to define points that influence the values in their vicinity while never completely taking over, as their maximum weight can never be reached. Their influence decrease as their *Rmin* reaches more negative values. This gives us the final equation implemented in the external:

$$W_i = \frac{R_i}{(max(d_i - Rmin_i\ ,\ dmin))^{power}} \quad 1 \leq i \leq n \tag{5}$$

For the sake of clarity, we decided to bound the absolute value of *Rmin* by *R*, so that the outer boundary (i.e. a circle in a 2-D space) always displays *R*. To avoid the same type of shapes for *R* and *Rmin*, which could be confusing when points are superposed, we choose to display *Rmin* with a filled area and *Rmin* as a frame. The filled area includes the centre for a positive *Rmin*, showing the zone where the maximum weight is reached. It surrounds the centre at *Rmin* distance for a negative one, showing the additional distance to the mathematical center.

2.3. The Void

The idea of perturbation made possible by negative *Rmin* led to the concept of a constant field over the whole space, the void, linked to its own set of values. Within this constant field, obtained simply by attributing a user-defined constant weight to the void, the points would perturb the field with their associated sets of values. But we also defined a weight dependent on the distances between the cursor and all the other points. We offer a choice between the distance to the closest point d_{void_min} and two combined normalized distances:

$$d_{void\_sum} = \frac{\sum_{i=1}^{n} d_i}{n} \qquad and \qquad d_{void\_prod} = (\prod_{i=1}^{n} d_i)^{1/n} \tag{6}$$

We then compute the weight of the void as the sum of those two distinct contributions:

- a constant weight defined by a constant distance D_{void}, in the same way as W_i is defined by the distance from the cursor to the point i, because a distance is easier to grasp than a more abstract numerical weight value.
- a weight depending on either a minimum distance or a combined normalized distance between the points and the cursor, in relation to $Dmin_{void}$, the distance from where the influence of the void starts to take place. The intensity of this contribution is expressed with the same metaphor of mass as for the other points: with a radius, R_{void} and an exponent, *power*.

3. MAX EXTERNAL

Instead of defining a new GUI object, we decided to make a Max external that communicates with existing GUI objects: *LCD* and *jit.window* or *jit.pwindow*. The first creation argument defines the type of display: *no_display*, *LCD*, *Jitter* or *LCD_Jitter*, allowing simultaneous LCD and Jitter displays, as shown in Figure 1. The second argument defines the visualization mode: *1-D vertical*, *1-D horizontal*, *2-D* or *3-D*. All creation arguments are optional. The display and visualization modes can be redefined at any time with messages.

Colors can be defined with the RGB output of the *swatch* object. The different available color modes are shown in Figure 2 for a 2-D interpolator and, in Figure 3, for various vertical or horizontal 1-D versions, using *LCD*:

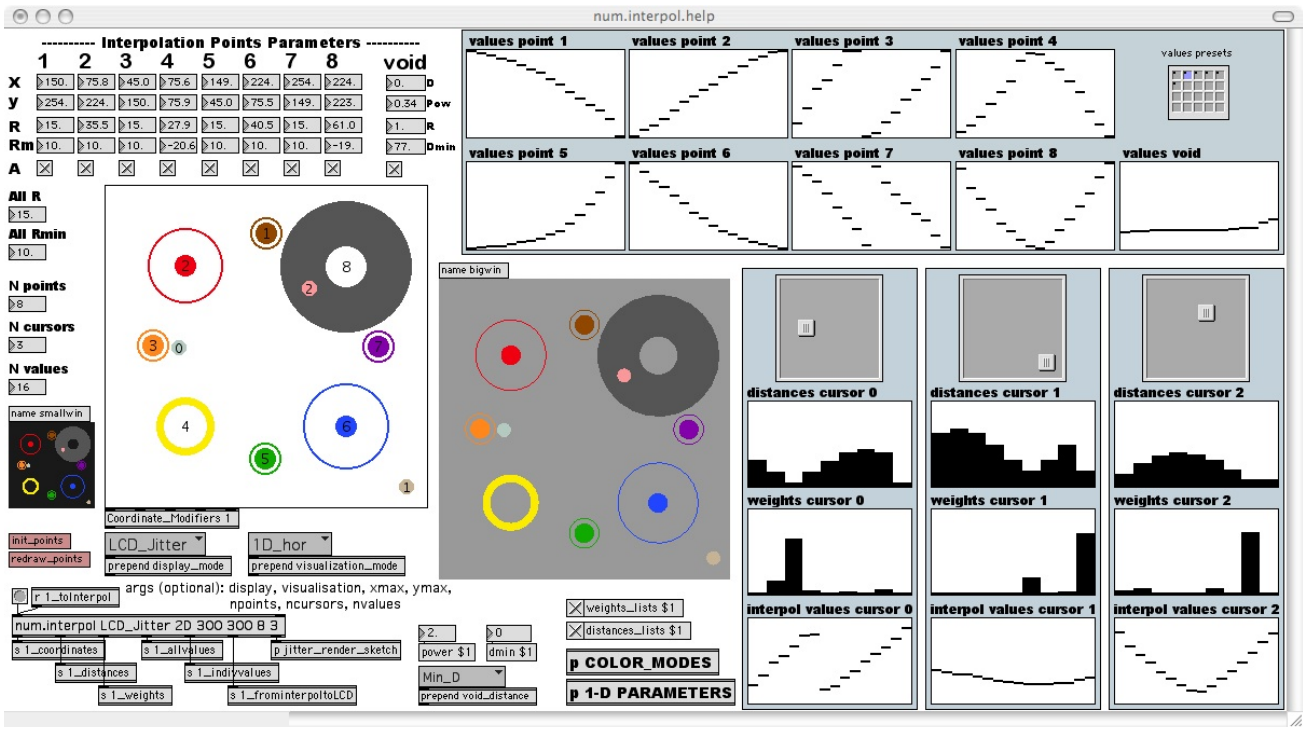

Figure 1. *Shown on one LCD and two jit.pwindow displays: eight interpolation points and three cursors with resulting distances, weights and interpolated values from the nine sets of values, including one for void.*

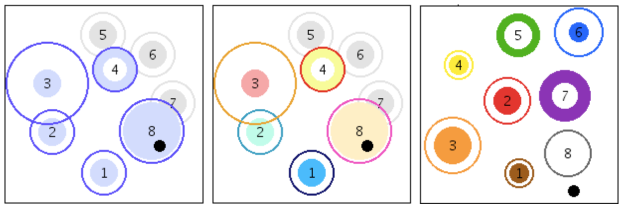

Figure 2. *Color modes: (left) user_single, (center) user_multiple, (right) resistor_code. Light grey represents inactive points and the black dot is the cursor.*

- user_single: all points i share the same R_i color and the same $Rmin_i$ color, both defined by the user.
- user_multiple: R_i and $Rmin_i$ colors can be defined individually for each point i, allowing the user to define whatever colors seem to fit the character of the sound transformation associated to each point.
- resistor_code: users familiar with the resistor code can locate the points faster, without reading the numbers.

All the parameters of the interpolation points may be defined by Max messages sent to *num.interpol*. Figure 1 shows *bpatchers* used to modify the coordinates, radiuses and activity for a 2-D interpolator. But data is also sent out at each change of point or cursor parameter. Combined with distances and weights outputs, activated by the messages *distances_lists* and *weights_lists*, it allows to use the externals for a variety of other tasks, like spatialization of sounds. Or

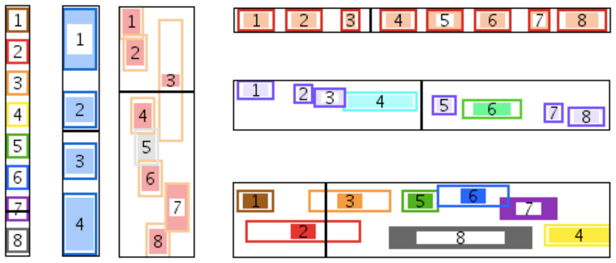

Figure 3. *Several 1-D setups in different color modes, vertical on the left and horizontal on the right. The horizontal or vertical black line shows the cursor from where distances to each interpolation point is computed.*

to send interpolate messages directly to *pattr* in case the user prefers to store data in *pattr* rather than in the object.

The data to be interpolated, i. e. the sets of values for each interpolation point can be sent to the external with the help of lists of values in the format: point#, value#, value1 [value2, value3, ...]. This allows both to set up individual values as well as to use the output of a *multislider* followed by a simple *prepend point# 1* object. The resulting interpolated output values are available either as lists directly usable by *multislider* or as individual values.

Finally, we implemented the possibility of moving several cursors, identified graphically by different numbers and colors (with similar color modes as the points), in the same interpolation space. This allows multiple users or multiple

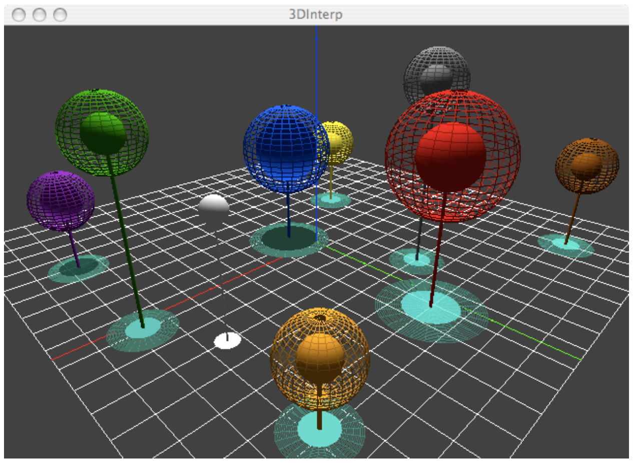

Figure 4. *3-D version in Jitter, using a poly_mode representation for R, and showing Rmin in the horizontal projection as a clear circle if positive and as a dark circle if negative. The white sphere represents the cursor position.*

sensor axes to interpolate in a polyphonic way.

3.1. LCD

The use of *num.interpol* with the *LCD* object is very straightforward and requires hardly no other Max objects. All messages to *LCD* are generated by the external and directly fed to the input of *LCD*. Sprites for each point and for each cursor are defined at initialization of the external object. They are redefined each time a point is resized or (de)activated or when the visualization mode is changed. The communication is bi-directional: the interpolation points are displayed in the GUI which also returns mouse position, mouse state (clicked or idle) and modifiers in order to move, (de)activate and resize the points. Modifiers are used to change the parameters graphically. By Default, Ctrl is used to move the points, Shift ⇧ to (de)activate the points and the combination Ctrl + Shift ⇧ to resize R and Rmin. We plan to add messages so that modifiers can be defined by the user.

3.2. Jitter - OpenGL

The use of *num.interpol* with a *jit.(p)window* for one or two dimensions is very similar, needing only a *jit.gl.render* and a *jit.gl.sketch* object. All messages needed to initialize the points, move and resize them are generated by *num.interpol*. And the mouse control to modify the points work in exactly the same way as with *LCD*.

But *num.interpol* is a bit more complicated to use for 3-D spaces as it needs not only a global *jit.gl.render* object but also several *jit.gl.gridshape* and *jit.gl.handle* objects for each point. The external generates all the messages to define all the parameters needed for those modules but abstractions are needed for each point. Figure 4 shows how we added projections on the horizontal plane and lines linking them to the spheres in order to resolve confusions in perspective views. Though functional when defining the points parameters with Max messages, we still have to implement graphical modification of points parameters in the 3-D version.

4. CONCLUSIONS

In our work on a project with a dancing viola player, equipped with gyroscopic and accelerometers sensors, we enjoyed the flexibility of this approach, combined with usual techniques of one to one mapping (input and output scaling, filtering, tables, envelopes...) and a Dynamic Time Warping gesture follower. We are currently tweaking the last details and finishing the documentation. The external and abstractions will be available on www.maxobjects.com and www.numediart.org. We plan to experiment further with the possibilities of changing the coordinates and sizes of the interpolation points in real-time in interactive installations, where they could for instance follow the positions and level of activity of visitors. We are considering a port to Pd-Gem.

5. ACKNOWLEDGMENTS

Research supported by `Numediart`, a long-term research program centered on Digital Media Arts, funded by the Région Wallonne, Belgium (grant N°716631).

6. REFERENCES

[1] J. Allouis and J. Y. Bernier, “The syter project: Sound processor design and software overview,” in *Proceedings of the ICMC*, Venice, Italy, 1982, pp. 232–240.

[2] R. Bencina, “The metasurface: applying natural neighbour interpolation to two-to-many mapping,” in *Proceedings of the NIME*, Vancouver, Canada, 2005, pp. 101–104.

[3] O. Larkin, “Int.lib - a graphical interpolator for max/msp,” in *Proceedings of the ICMC*, Copenhagen, Danemark, 2007.

[4] A. Momeni and D. Wessel, “Characterizing and controlling musical material intuitively with geometric models,” in *Proceedings of the NIME*, Montreal, Canada, 2003.

[5] T. Todoroff, F. Bettens, L. Reboursière, and W.-Y. Chu, “Extension du corps sonore - dancing viola,” in *Proceedings of the NIME ’09*, Pittsburgh, Pennsylvania, USA, 2009.

[6] T. Todoroff, C. Traube, and J. M. Ledent, “Nextstep graphical interfaces to control sound processing and spatialization instruments,” in *Proceedings of the ICMC*, Thessaloniki, Greece, 1997, pp. 325–328.