Interpolated mappings for musical instruments

CAMILLE GOUDESEUNE

Beckman Institute, University of Illinois at Urbana-Champaign, 405 N Mathews, Urbana, IL 61801, USA
E-mail: cog@uiuc.edu

Software-based musical instruments have controls for input, a sound synthesizer for output, and mappings connecting the two. An effective layout of controls considers how many degrees of freedom each has, as well as the overhead of selecting each one while performing. An isolated mapping from one control to one synthesis parameter needs an appropriate choice of proportional, integral or derivative control (the control's value, or that value's rate of change, drives the synthesis parameter's value, or *that* value's rate of change). Beyond this, a compound mapping cross-coupling several controls and synthesis parameters can surprisingly increase the performer's intuitive understanding of the instrument.

Such a compound mapping is essentially a continuous function from $\mathbf{R}^d$ to $\mathbf{R}^e$ for arbitrary integers d and e, where $1 \leq d < e$. Such a function is easily constructed by associating several sets of d control values with corresponding sets of e parameter values (i.e. sounds). This 'pointwise' mapping can then be extended through a geometric technique called simplicial interpolation to produce a continuous mapping, which can be adjusted and refined by simply moving or adding new pairs of 'points'. Furthermore, given the initial sounds in $\mathbf{R}^e$, the corresponding control settings in $\mathbf{R}^d$ can be found automatically. An open-source C++ implementation of this technique is available.

1. INTRODUCTION

Since both an instrument and a composition written for it may involve software, and since mappings are typically realised with software, mapping properly belongs to the disciplines of both instrument design and composition. Here we focus on the real-time part, the instrument.

If the performer can comprehend the mappings embedded in an instrument, obviously a more refined performance can result. This argues for static mappings over dynamic, and simple over complex (although we shall see that overly simple mappings can be sub-optimal). Explicit construction of a mapping may or may not be better than having an algorithm compute it as the solution to some set of criteria.

Whatever decisions are made about mappings in an instrument, they result in what performers call the feel of the instrument, its responsiveness and controllability, its 'consistency, continuity, and coherence' (Garnett and Goudeseune 1999). Listeners perceive the result in the range, accuracy and speed of performed gestures.

Three questions can be asked about a mapping: from what? to what? and by what means? Here we answer the first two by considering a musical instrument as something into which the performer puts gestures, and which then outputs sounds. Since adjustable mappings require software, we explore the world where software lies between the gestures and the sounds. So a mapping is from an instrument's controls, to the inputs of a sound synthesizer ('synthesis parameters'). An expanded answer to these first two questions considers the interactions between individual controls and individual synthesis parameters.

Software-based instruments can be motivated from asking what sounds a given controller might produce, or equally well from how one might perform a given family of synthesised sounds. Ryan (1992) calls instrument design the putting of 'physical handles on phantom models', discovering which controls ('handles') work well with mappings into a synthesizer (abstract 'models' of sound).

One answer to the third question is explored at length in this paper. Given an instrument's controls, we must construct a smooth mapping from their several degrees of freedom (or dimensions, geometrically speaking), to a possibly different number of scalar inputs accepted by the instrument's synthesizer. As a musically useful starting point, such a mapping can be built up from a *pointwise map*, an association of particular input values with output values: when the performer does *this*, the instrument should sound like *this*. An *interpolator* then produces reasonable intermediate outputs for intermediate inputs. Many techniques of interpolation can generate intermediate data from discretely sampled data. Simplicial interpolation, presented here, is well suited to the application domain of musical instruments.

Since it is often the case that a sound synthesizer has many more real-time inputs than a human can attend to simultaneously, we would like to reduce the complexity of a large set of parameters. A *high-dimensional interpolator* (HDI) lets the performer control a large number of parameters with a much smaller number.

A parameter which takes on a discrete set of values can often be reduced to a continuous one. If the values can be ordered from least to greatest, the parameter can directly be treated as continuous though with coarse resolution. If the values are not so orderable, as with the

Organised Sound **7**(2): 85–96 © 2002 Cambridge University Press. Printed in the United Kingdom. DOI:10.1017/S1355771802002029

https://doi.org/10.1017/S1355771802002029 Published online by Cambridge University Press

ad hoc timbres offered by the stops of a pipe organ, they may be embedded in a space of dimension greater than one by means of perceptual discrimination, situating similar values close together. (This is attempted in two dimensions by the layout of both organ stops on a console and orchestral musicians on a stage.) Some comparison (i.e. ordering) of discrete values is necessary for any generalisation or theory, and such an ordering leads directly to speaking of points in a continuous space.

2. CONTROLS AND DRIVING GRAPHS

By *controller* we mean the complete interface or set of commands available to the instrumentalist. By *control* we mean a single indivisible part of the controller. A control's *value* is its instantaneous state. We call a control *scalar* if its value is a single continuous scalar. If its value is discrete, the control is often called a *switch*.

A *dimension* is a linear continuum. Its *value* is a scalar, the dimension's realisation at an instant of time. Often a dimension is associated with a single scalar control, in which case we identify the values of the dimension and the control. Sometimes we speak interchangeably of *parameter*, *dimension* and *degree of freedom*.

A continuous control *drives* a dimension if a change in its value produces a corresponding change in the dimension's value.

Since one control may drive several dimensions, and again, several controls may drive the same dimension, we can imagine a set of points corresponding to controls, a second set of points corresponding to dimensions, and arrows from points in the first set to points in the second. We call this diagram the *driving graph* of the instrument (figures 1 and 2). Hunt, Wanderley and Kirk (2000) mention three common mapping strategies, namely 'one-to-one', 'one-to-many', and 'many-to-one'. The concept of driving graph unifies and extends these, and also makes available many results from the mathematical field of graph theory.

The *gain* of a scalar control (with respect to some dimension that it drives) describes how strongly input affects output. For the same range of input values, a high-gain control has a wider range of output values while one with low gain demands less accuracy and thus is easier to play.

The *order* of a scalar control (again, with respect to a driven dimension) is an integer describing how direct the mapping is from the value of the control to the value of the dimension. If the mapping is direct, the control has order zero. A control of order one has a direct relationship between the control's value and the *rate of change* of the dimension's value. A control of order –1 has the opposite behaviour: a direct relationship between rate of change of the control's value and the dimension's value. To speak of the order of a scalar control, we require only that the mapping is continuous; in particular, the mapping need not be linear. (For reasons of extensibility, this separation of discrete and scalar controls departs from the three-level hierarchy of discrete ('set-and-forget' auxiliary) controls, order-zero scalar controls, and order-one scalar controls presented in Vertegaal, Ungvary and Kieslinger 1996: 310). A discrete control cannot be said to have an order when it is not based on an underlying continuum.

A more general description of how input drives output is given by a constant-coefficient linear differential equation (Sheridan and Ferrell 1981: 178–80). The coefficients indicate how much gain is present at each order. But we can practically restrict ourselves to equations with only one non-zero order coefficient. In fact we prefer an alternative notation for the order of a control, borrowed from the field of *proportional-integral-derivative* or PID control (Dean and Wellman 1991: 144). If $x(t)$ is the control's value at time t and $y(t)$ is its dimension's value, then a *proportional control* can be expressed as $y(t) = \eta(x(t))$ for some mapping η: $\mathbf{R}\rightarrow\mathbf{R}$; an *integral control* has the form $y(t) = \eta(\int_0^t x(t)\mathrm{d}t)$; and a *derivative control* has form $y(t) = \eta(\frac{\mathrm{d}}{\mathrm{d}t}x(t))$. In other words, the value of a proportional control depends purely on its input, that of an integral control on the input's history, and that of a derivative control on the input's rate of change. Orders –1, 0 and 1 correspond to derivative, proportional and integral controls, respectively. (A mnemonic: –1, 0 and 1 correspond to a polynomial's change of degree when differentiated, unmodified

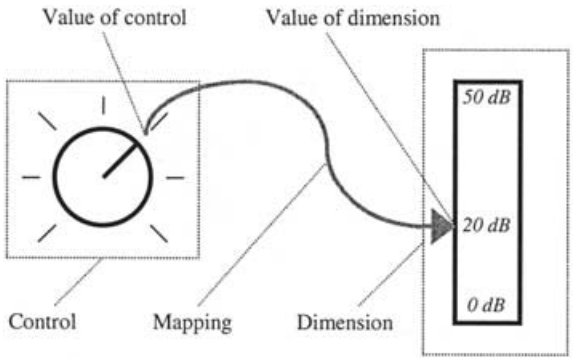


Figure 1. The simplest non-trivial driving graph of an instrument: one scalar control driving one dimension.

https://doi.org/10.1017/S1355771802002029 Published online by Cambridge University Press

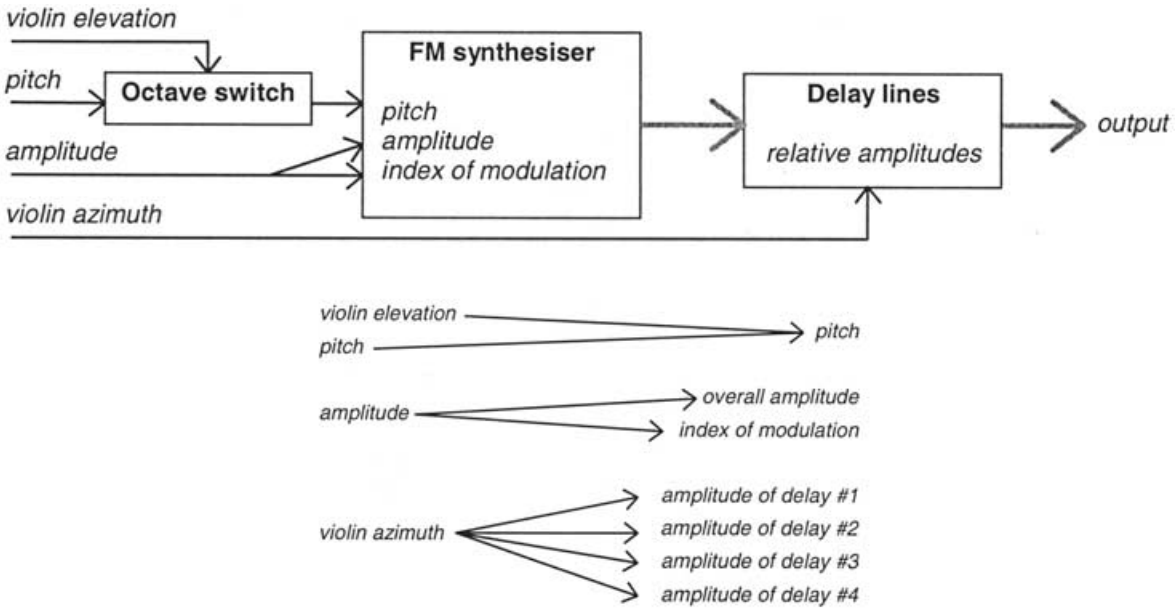


Figure 2. Driving graph of an instrument based on an electric violin tracked in pitch, amplitude and spatial orientation. Rotating the violin upwards raises the output pitch by an octave; rotating about a vertical axis changes the relative amplitude of several delay lines through which the sound passes.

or integrated, respectively.) Orders greater than 1 or less than –1 play a relatively minor role in both industrial processes and musical instruments, so the terms proportional, integral and derivative control often suffice. However, sophisticated models of transient phenomena such as the breakdown and re-establishment of Helmholtz motion when bow velocity changes or reverses (i.e. acceleration) may involve greater orders.

Three commonplace examples illustrate PID controls. Fingertip placement on a fingerboard, driving pitch, is a proportional control. Only where the finger is, not how fast it moves or where it has been, determines pitch. Amplitude often uses a derivative control: the speed of a bow, not its current or previous positions, determines the amplitude of the string's vibration. Laptop computers are now often seen on stage in computer music concerts; a sound synthesis parameter (perhaps volume, pitch, or a filter parameter) driven by the laptop's 'trackpoint' built-in mouse exemplifies an integral control, because the parameter's value depends purely on the history of nudges applied to the trackpoint.

Several results from PID control theory apply to musical instruments. Proportional controls are most common. Derivative controls are more agile than proportional controls but fare worse at holding a constant value, which suits them well to controlling sound amplitude. Integral controls are, not surprisingly, the opposite: they lack agility but once they reach a desired output value they easily maintain it (by zeroing their input: once $x(t)$ is zero, as t continues to increase then $\int_0^t x(t)\mathrm{d}t$) remains constant). Derivative and integral controls can also be characterised in terms of transient response, the ability to execute sudden changes in value. Derivative controls have good transient response while integral controls, especially ones with large gain, tend to overshoot the desired output value. Finally, integral controls work better when the controlled system has high hysteresis (Dean and Wellman 1991: 148–9).

3. INPUTS TO A MAPPING

If a controller has several controls, particular controls can be *selected*, *adjusted* and *deselected*. Selection is the allocation of performer resources to a particular control; adjustment is the actual change of state of the control; and deselection is the relinquishing of resources in preparation for subsequent selections. Less abstractly, when a performer selects a control, he directs attention and possibly executes some muscular motion. By resources we mean the performer's finite cognition, attention and physique: only this many limbs with this range of motion, this spatial resolution, and this speed.

Selecting a control may be as simple as moving a limb to it. But sometimes it may be unwieldy to have all the controls directly accessible by mere positioning of a hand. Then we need to replace 'space multiplexing' with 'time multiplexing' (Buxton 1986); in other words, we need *secondary controls* to change the behaviour of the *primary controls*. Mutes are a common example of secondary controls. Also, a single multi-way selector switch can associate a primary control with one of several sets of scalars, for example keyboard couplers on an organ. It is difficult to rigorously define when a control is secondary, but the label applies well when several of the

https://doi.org/10.1017/S1355771802002029 Published online by Cambridge University Press

following hold: the control modifies the behaviour of another control which is manipulated more often; the control can be left unattended for a while, physically or only attentionally; the control causes a sound or behaviour which is in some way non-standard.

Introducing secondary controls reduces the number of primary controls, thereby simplifying the controller. Of course this is a compromise: simultaneous adjustment of multiple controls is reduced. Also, the interface is deepened even while it is narrowed: the performer's mental model of the instrument is more elaborate and takes longer to learn. A range of compromise in fact exists. At one extreme there are k primary controls (strings on a stringed instrument, keys on a multiply touch-sensitive clavier) and no secondary controls, at the other extreme one primary control with a single k-way selector switch (e.g. the Ondes Martenot). Between these two extremes there may be m primary controls with an n-way selector switch, where $mn \geqslant k$ (bass guitar: $m = 4$ strings, $n = 3$-way pickup switch). If only a few secondary controls extend the interface of an orchestral instrument, they can often be operated by the feet, for instance as a bank of toggle switches or a three- or four-way 'gas pedal'.

3.1. Continuous controls: sliders and multisliders

We abbreviate 'continuous control' with the term *slider*, distinguishing one-dimensional *scalar sliders* and higher-dimensional *multisliders*. Scalar sliders include linear sliders and rotary knobs in a physical control apparatus, sliders or scrollbars on a computer display, and non-manually controlled sensors of pressure, proximity, and so on. Joysticks, mice, virtual reality 'wands', and motion-tracking systems for dancers are examples of multisliders. As a multislider concurrently drives several dimensions, it is suitable when performance gestures are desired more in the product space of these dimensions than in each individual dimension. (To draw freehand, one prefers a mouse; to draw straight vertical and horizontal lines, the humble Etch-a-Sketch is better (Buxton 1986).) Considering which dimensions are coupled in performance gesture and which are independent shows the instrument designer where multisliders are appropriate.

A slider can be *absolute* or *relative*. An absolute slider has a direct correspondence between the slider's position and the scalar's value. A relative slider, in combination with a secondary control, can change the origin of its coordinate system relative to that of the scalar. This secondary control is usually a momentary switch (one which remains engaged only as long as force is applied to it). When the switch is engaged, the slider is active: moving the slider adjusts the scalar's value. When the switch is not held, the slider is inactive and can be moved without adjusting the scalar's value. Another way to think of this is that the switch changes the slider's behaviour between changing the parameter value and changing the origin of the coordinate system. A more sophisticated kind of relative slider varies its gain continuously, with yet another slider.

A relative slider is useful if the effective range of the slider needs to extend beyond its physical range ('pawing' a computer mouse – here the secondary control is lifting the mouse from the desk). Three reasons explain the rarity of relative sliders in musical instruments: the time required to perform a change of coordinate system undesirably constrains real-time performance; the secondary controls introduce what human–computer interface specialists call *mode*, undesirable state in the interface itself; relative sliders do not offer direct proprioceptive feedback (a consequence of this extra state).

A bank of sliders with *multiple selection* typically requires two-handed or multi-fingered operation. If all sliders in a bank can be selected simultaneously, the bank differs from a single multislider only in that it provides a set of primary axes for the space in which gestures are performed. 'Rotated' gestures are harder to perform with an array of sliders than with a true multislider: try drawing straight *diagonal* lines with an Etch-a-Sketch. If all controls cannot be selected simultaneously, this may be due to either physical or attentional limitations. Physical limitations, as is the case with a pianist's ten fingers and finite hand span, impose hard rather than gradual constraints on which gestures are performable. Attentional limitations lead to gradually more constrained gestures as more controls are selected. This is because of rehearsal: the violinist can learn to play double and triple stops with more accurate intonation and bowing, trading off accuracy and rehearsal time. But no amount of rehearsal gives a pianist extra fingers.

If only one control at a time can be selected, as with sliders displayed on a computer screen and manipulated with a mouse, then no gestures involving correlated parameters can be performed, a severe restriction. A bank of sliders is better suited for 'set-and-forget' parameters than for direct real-time control. At the other extreme, the Continuum controller is a sophisticated multislider with multiple selection. It accurately tracks the pressure and x–y position of up to ten fingers on a smooth surface as large as a piano keyboard (Haken, Fitz, Tellman, Wolfe and Christensen 1997). A bank of multiple-selection controllers may have an additional implicit secondary control: each member of the bank is enabled by depressing a momentary switch (touching the Continuum's surface). This is particularly natural for polyphonic controllers, where selecting one more control produces one more sound.

3.2. Multisliders and cross-coupling

Vertegaal, Eaglestone and Clarke (1994) investigated the use of several controllers for a four-dimensional timbre

https://doi.org/10.1017/S1355771802002029 Published online by Cambridge University Press

space (overtone content, brightness, articulation of attack, and speed of envelope). One controller was a single multislider, a glove tracked in three-dimensional position and in one rotational dimension. The other controllers had a single two-dimensional primary control (mouse, joystick); secondary switches applied it to either the first or the last pair of dimensions of the timbre space. This pairwise division was reasonable: the first pair dealt with steady-state spectral content, the second with attack. Visual feedback of the four dimensions was presented as two square grids each containing a cursor, again emphasising the pairwise division.

The glove controller fared most poorly in this study. This was attributed to its high latency, but subjects may have also had difficulty cognising three separate presentations of information: glove position; the visual display; and the actual sound. In particular the pairwise structure of the display and the sound was not reflected in the measured parameters of the glove. The visual feedback may have distracted rather than assisted the performer: even a simple joystick suffers if its *x*–*y* position is rendered indirectly, for instance as the size and hue of a coloured disc instead of as the *x*–*y* position of a point in a square.

A more recent study found that a multislider sometimes fared better than a bank of sliders with multiple selection, even when subjects could not verbalise how the multislider worked. Hunt and Kirk (1999, 2000) conducted several thousand trials where subjects used three different controllers: a mouse controlling four sliders displayed on a screen; four physical sliders; and a mouse plus two physical sliders. With each controller the subject had to duplicate a short sound which might vary in volume, pitch, unidimensional timbre, or stereo position. In all cases the mouse performed most poorly, as predicted by our theory of selection overhead. In simple tasks where only one parameter of the sound changed, the bank of physical sliders showed the best performance though the multislider (mouse plus two sliders) showed improvement as trials progressed. For complex tasks the multislider was best. This is remarkable since the four parameters were not simply assigned to slider one, slider two, mouse *x*-position, and mouse *y*-position. Rather, each parameter depended on expressions like overall mouse speed plus average of the slider positions, or mouse *y*-position plus the speed of slider one. The experimenters concluded that the simplest mapping of controls to synthesis parameters is not necessarily optimal. (We hope for another study where the four parameters *are* simply assigned to the four obvious linear controls, for an even stronger conclusion.)

We call a set of controls *cross-coupled* if they lie in the same connected component of the driving graph (figure 2: violin elevation and pitch). In other words, the controls cannot be considered individually. Rovan, Wanderley, Dubnov and Depalle (1997) elaborately cross-couple the components of a MIDI wind controller to more realistically drive an additive-synthesis clarinet sound model. They suggest that simpler, less cross-coupled, mappings can help novices learn to play; technically, cross-coupled controls are harder to learn because part-task training on individual controls transfers poorly to the whole task. But both Rovan's instrument and Hunt and Kirk's multislider indicate that cross-coupling can produce a better controller, once learned. If an instrument seems to demand cross-coupled controls, grouping them together to drive an HDI (again following Rovan) encourages the performer to think of the controls as a single entity. The converse holds, too: a set of strongly non-coupled controls like a bank of sliders is inappropriate for driving an HDI, because it incorrectly suggests that each control has an inherent meaning.

3.3. Input devices

Rotary knobs and linear sliders provide inexpensive control of continuous parameters. Bounded sliders typically work absolutely while unbounded sliders are naturally relative. The simple joystick is like a pair of rotary knobs cross-coupled. *Force-feedback* joysticks offer variable resistance to motion or haptic display of impulses and vibrations. SensAble Technologies' *Phantom* is a desktop-sized articulated arm which senses position, and applies force in, three dimensions; it has been extended to measure orientation and apply torque, for an impressive total of six dimensions each of input and output (Chen 1999).

Pressure sensors such as force-sensing resistors offer continuous control in one dimension. They can be combined into an isometric joystick (two or three degrees of freedom) or a *spaceball* (six degrees of freedom, strongly cross-coupled).

The *ribbon controller* and *touchpad* track the position (and pressure) of one or more fingertips along a line and on a surface, respectively. A touchpad may be divided into smaller touchpads, sliders and switches on a single physical surface. This is the input-side analogue to a window manager for graphical output. Tactile feedback can be provided for such multiple devices by laying a cardboard template on the physical device (Buxton, Hill and Rowley 1985).

Light pens and *tablets* with styli are more accurate than touchpads but require the hand to hold a stylus, so they work poorly when the hand also has other tasks to perform. Buttons mounted on the stylus may offer a greater repertoire of gestures than a touchpad. *Motion tracking systems* measure the position and/or orientation of sensors freely moving in space, typically attached to a glove. Each sensor is like a tablet plus stylus, extended to three (or six) dimensions: the working area is now a volume instead of a surface.

https://doi.org/10.1017/S1355771802002029 Published online by Cambridge University Press

Selection overhead → ***Property sensed*** ↓	*low*	*medium*	*high*
Discrete state	momentary switch, toggle switch, hat switch	multi-way switch	numerical keypad
Fingertip location (and unidirectional force)	ribbon (1–2), touchpad (2–3)	multi-touch pad (15)	Continuum (30)
Unidirectional force applied by hand	force-sensing resistor (1)	torque sensor (1), isometric joystick (2–3), spaceball (6)	
Location (and orientation) of manipulandum	large knob (1), slider (1), trackball (2)	small knob (1), mouse (2), joystick (2–3)	tablet+stylus (2–4), wand (3–6), motion tracking glove (15–30)
Location (and orientation) of, and force applied by, manipulandum			pressure-sensitive tablet+stylus (3–5), force-sensing mouse (4), Phantom (6)

Figure 3. Elementary manual controls. Parenthesised numbers indicate how many degrees of freedom a control has.

Properties of these devices are summarised in figure 3. Beyond what is shown in this figure, a control may be:

- primary / secondary;
- separate / an element of a bank / an element of a multiply selectable bank;
- absolute / relative;
- bounded / unbounded in its motion;
- with / without visual feedback (at the control itself, or separately in a computer display).

Figure 3 shows that selection overhead generally correlates with number of degrees of freedom. Efficient controls, those with a high ratio of degrees of freedom to selection overhead, are: touchpads, trackball; spaceball; motion-tracking glove. Other considerations being equal, these are therefore especially recommended as manual controls for musical instruments. Those with low ratios may have other advantages like high resolution or small size.

This terminology concisely summarises the findings of Vertegaal, Ungvary and Kieslinger (1996), supported by experimental results (Wanderley, Viollet, Isart and Rodet 2000). An effective isometric (scalar) control is integral with respect to applied force, while an effective isotonic control is proportional with respect to position. Isometric controls rely particularly on tactile feedback, how hard the control is pushed, rather than visual feedback. A discrete control works best when its different states correspond to different spatial positions, not different speeds or forces; visual feedback can play a greater role here.

Finally, common orchestral instruments can be themselves used as controls. The instrument's sound can be analysed into parameter streams which then drive a sound synthesizer. Examples of such parameters are pitch, amplitude, and timbral information such as spectral centroid and amount of unpitched noise. The literature speaks of *amplitude following* (*envelope following*) and *pitch tracking*. Secondary parameters like depth of vibrato can also be derived from the tracking data. We can also add other devices to the instrument such as motion tracking and pedals.

4. INTERPOLATION

4.1. Classical interpolation

In the most general sense, interpolation is 'the performance of a numerical procedure that generates an estimate of functional dependence at a particular location, based upon knowledge of the functional dependence at some surrounding locations' (Watson 1992: 101). Interpolation is a convenient starting point for constructing a mapping between two Euclidean spaces, from a space of dimension equal to the number of degrees of freedom of the instrument's continuous controls, to the space of parameters fed to the synthesizer. (This can be taken even farther, to the space of perceptual parameters of the synthesizer's sound output; this perception can also be automated (Goudeseune 2001: 151–6). Here we simply identify 'sound' with 'synthesizer parameters'.)

It is easy to specify several pairs of points in the input

https://doi.org/10.1017/S1355771802002029 Published online by Cambridge University Press

and output spaces, in effect specifying 'when the controls have these values, make this sound'; interpolation then defines intermediate sounds for intermediate values of the controls. Having tried out the instrument resulting from this map, one can then refine the map by moving input points (make that sound over here instead), moving output points (no, that sound needs to be more like this one), or introducing new pairs of points (adjust the sounds which the interpolator happened to produce in this little region). With an appropriate choice of interpolator we need assume little else about the system, not even linearity of the synthesizer or of human perception. In short, constructing continuous maps by extending pointwise maps through interpolation has the advantages of low effort, generality, scalability and local adjustment. We do require that the interpolator accept arbitrary spatial arrangements of points, not only rectilinear grids; otherwise these advantages vanish.

The result of all this is that given such points, an interpolator can be used as a controller which, moment by moment, takes as input a point in $\mathbf{R}^d$ and outputs a point in $\mathbf{R}^e$. This output point moves through an unbounded nonlinear d-dimensional region containing the given fixed output points. In a real-time musical application the controller takes as input d real-valued data streams from physical controls and outputs e data streams to a sound synthesizer. Typically d is 2 or 3 while e lies between 5 and 50, but the mathematics holds for any $0 < d < e$.

In this context, we define the unidimensional interpolation problem thus: given a finite pointwise map $S = \{(x_i, y_i)\} \subset \mathbf{R}\times\mathbf{R}$, construct a function $f : \mathbf{R}\rightarrow\mathbf{R}$ such that $y_i = f(x_i)$ for all i, and such that f has nice properties. Among these properties, continuity is usually most desirable; others are differentiability, having bounded higher derivatives, being C^∞ and having adjustable smoothness and tension. More generally: given sets A and B and a function $g : \mathrm{A}\rightarrow\mathrm{B}$, construct $f : \mathrm{A}\rightarrow\mathrm{B}$ such that $f \supset g$ and f is nice. Our niceness is not treated directly in classical interpolation theory, because we have no ideal function approximated by the interpolation; there is no error measurement to speak of.

Certain implicit assumptions of interpolation theory may not hold for musical instruments. For interpolation to make sense, the input and output spaces must themselves be continuous. For the mapping to be repeatable, the input and output spaces should not change with time; in particular, large hysteresis in the sound synthesizer (when its output is influenced by past as well as current inputs) is incompatible with this method of constructing mappings.

The property that the extended map f agrees with the pointwise map S characterises the restricted class of interpolators called *exact*. We assume exactness from now on (and thereby disregard neural network interpolation; neural networks are better at classifying than interpolating). Classical exact interpolators include proximal interpolators, B-splines and kriging (Collins and Bolstad 1996). Though simple, proximal interpolation and its generalisations are inappropriate for most musical uses since they produce many discontinuities in f. B-splining does better, constructing f from patches of polynomials so it is not only continuous but C^2. It tends to produce artifacts such as overshoots beyond the input values when these input values are not monotonically distributed, because it prefers minimum curvature to avoiding such artifacts. It is not recommended for points not spaced on a grid (Eckstein 1989, Hutchinson and Gessler 1994). As musical applications need not in general be smooth (breaks in woodwind register, for example), B-splining seems ill-suited for general musical use. The more sophisticated regularised spline with tension avoids overshoots and constructs f to be C^∞. Mitasova, Mitas, Brown, Gerdes, Kosinovsky and Baker (1995) describe its implementation in a geographical information system; Mitas and Mitasova (1999) sketch a generalisation to more than three dimensions.

Kriging considers how quickly the variance between input points changes in different parts of the space (Krige 1981, Oliver and Webster 1990). It becomes cumbersome in higher dimensions. Sárközy (1998) warns that the data should satisfy several stationarity conditions; these may not hold in musical situations. Kriging also performs poorly on sparse data, which can well be the case with timbre spaces. Hardy (1990) demonstrates that kriging is relatively poor at handling smoothness, an important factor for musical instruments.

Finally, multiquadrics (Hardy 1990) and polynomial interpolation with basis functions are two examples of function-fitting methods. These are less commonly used, perhaps because they generalise poorly to non-gridded data points (Sárközy 1998).

4.2. High-dimensional interpolators

'How can I control e parameters with only d scalar controls ($d < e$)?' We can restate this question as 'how can I make a *desirable* collection of gestures in $\mathbf{R}^e$ using only d degrees of freedom?' Here $\mathbf{R}^e$ is a family of sounds, perhaps a family of steady-state timbres; we leave desirability undefined in this general context. At any rate, by ruling out undesirable and redundant gestures we hope to find an interesting d-dimensional space inside $\mathbf{R}^e$. It is in fact possible to reduce the number of dimensions from e to d, practically because a high-dimensional space is simply too large to be exhausted by one instrument. More technically, the size of the space grows exponentially with the number of dimensions (e is an exponent, after all). Exhaustive exploration of such a space risks producing an incomprehensible instrument. So we expect that fewer dimensions suffice to describe the subset of the whole space which we call interesting or desirable.

https://doi.org/10.1017/S1355771802002029 Published online by Cambridge University Press

One way to find an interesting d-dimensional space inside $\mathbf{R}^e$ is to first specify several desirable points p_i in $\mathbf{R}^e$ (chosen to correspond to *a priori* interesting sounds), then construct a d-space enveloping them, and finally define a mapping from $\mathbf{R}^d$ into this d-space. Since this d-space is in $\mathbf{R}^e$, this defines a mapping all the way from $\mathbf{R}^d$ to $\mathbf{R}^e$. (At least d+1 points are needed; otherwise a smaller value of d could have been chosen.) If we build up such a mapping from a pointwise map by choosing corresponding pre-image points q_i in $\mathbf{R}^d$, we call the new mapping a *high-dimensional interpolator* or HDI. Beyond the mere existence of this mapping, there are desiderata to balance: continuity, differentiability, linearity, including all important parts of the range space, exactness, extensibility to larger spaces, ease of editing the map locally and globally. For now we require only continuity.

4.2.1. Automatic generation of pre-image points

We divide an interpolator into an initialisation phase and a running phase. The first phase is self-explanatory; in the second phase the interpolator actually performs the mapping from a query point q to an image point p.

As part of the initialisation phase, an interpolator can automatically find good pre-images q_i in $\mathbf{R}^d$. Taking the unit d-cube as the domain from which to choose the q_i, we find a set of q_i such that (i) their normalised pairwise Euclidean distances approximate those of the p_i, and (ii) they adequately fill the d-cube. By filling we mean that their convex hull has non-zero volume (to take advantage of all d dimensions at our disposal), and that the projection of this hull onto some principal axis of $\mathbf{R}^d$ yields the unit interval (to take advantage of all the distance at our disposal).

I have implemented two methods of choosing the q_i, a genetic algorithm (GA) and Sammon's mapping; both are random incremental algorithms. The population members of the GA are sets of q_i, and the fitness function minimised is RMS error between corresponding distance pairs of the p_i and a given set of q_i. Sammon's mapping iteratively refines a set of q_i in the manner of simulated annealing. Kohonen (1997: 31–2) suggests that Sammon's mapping requires $10^4 \cdot k$ to $10^5 \cdot k$ steps to arrive at an adequate solution, where k is the number of points q_i, but in fact certain choices of temperature gradient yield a good solution in merely $10 \cdot k$ to $100 \cdot k$ steps – if a good result is forthcoming at all. Since most of the improvement happens in the first few iterations, these two incremental algorithms are run with only a few iterations but many times, keeping the best result found. Sammon's mapping uses 700 runs of $70 \cdot k$ iterations each; the GA uses 10 runs averaging around 1,000 iterations each. Both take about 100 ms on current desktop computers to find an adequate solution for moderate values of k. (Recall that $k \geqslant d$+1; k has no upper bound, though.) The GA better avoids sub-optimal local minima, but Sammon's mapping uses less memory and is slightly faster. One cannot be universally recommended over the other.

4.2.2. Simplicial interpolation

Recall a few definitions from algebraic topology and computational geometry. The triangle in $\mathbf{R}^2$ and the tetrahedron in $\mathbf{R}^3$ generalise to a *d-simplex* in $\mathbf{R}^d$. A triangular mesh similarly generalises to a *simplicial d-complex*. A *Delaunay triangulation* of a set of points is a simplicial complex based on those points, with certain nice properties. A *simplicial map* is a kind of mapping between Delaunay triangulations. A point q lies in the *convex hull* of a set of points $\{q_i\}$ if it lies 'between' the q_i: $q = \Sigma w_i q_i$ for some positive weights w_i which sum to 1. Given a d-simplex σ with vertices $\{V_i\}$, any point q in $\mathbf{R}^d$ is expressible in terms of *barycentric coordinates* with respect to σ as $q = \Sigma b_i V_i$ where $\Sigma b_i = 1$. This is a weighted sum of the V_i. The point q lies in σ if and only if all $0 \leq b_i \leq 1$. For more rigour and background, see textbooks such as Alexandroff (1961), de Berg, van Kreveld, Overmars and Schwarzkopf (1997) or look up individual terms in the online reference, Weisstein (1999).

Simplicial interpolation consists of an initialisation phase and a running phase. At initialisation, a Delaunay triangulation of the pre-image points q_i in $\mathbf{R}^d$ is constructed and used to extend the pointwise map $q_i \rightarrow p_i$ to a simplicial map defined on the convex hull of the points in $\mathbf{R}^d$. The simplicial map is then extended beyond the hull to all of $\mathbf{R}^d$, to allow arbitrarily input values. In the running phase, the algorithm determines which simplex contains a given query point q in $\mathbf{R}^d$, computes q's barycentric coordinates with respect to that simplex, and then finds the corresponding point p in the simplicial d-complex in $\mathbf{R}^e$ (figure 4). Two points in $\mathbf{R}^d$ and $\mathbf{R}^e$ correspond if they lie in corresponding simplices, and have the same barycentric coordinates with respect to those simplices.

The running phase first induces a d-triangulation of the p_i in $\mathbf{R}^e$ from the d-triangulation of the q_i. Given a query point q in $\mathbf{R}^d$, it computes which d-simplex σ contains q with a constant-time bucket-search point location algorithm derived from a FORTRAN code by Edahiro, Kokubo and Asano (1984). It then computes the barycentric coordinates of q with respect to σ. The image p of q is then the point in the simplex in the triangulation in $\mathbf{R}^e$ corresponding to σ, with the same barycentric coordinates as those of q (see figure 4 again). In other words, q is characterised as a weighted sum of the q_i; p is the same weighted sum of the corresponding p_i. (Recall that barycentric coordinates can be interpreted as weights in a weighted sum.) This approach is due to a refinement by H. Edelsbrunner of my earlier algorithm (Choi, Bargar and Goudeseune 1995).

The special case where q lies outside the convex hull

https://doi.org/10.1017/S1355771802002029 Published online by Cambridge University Press

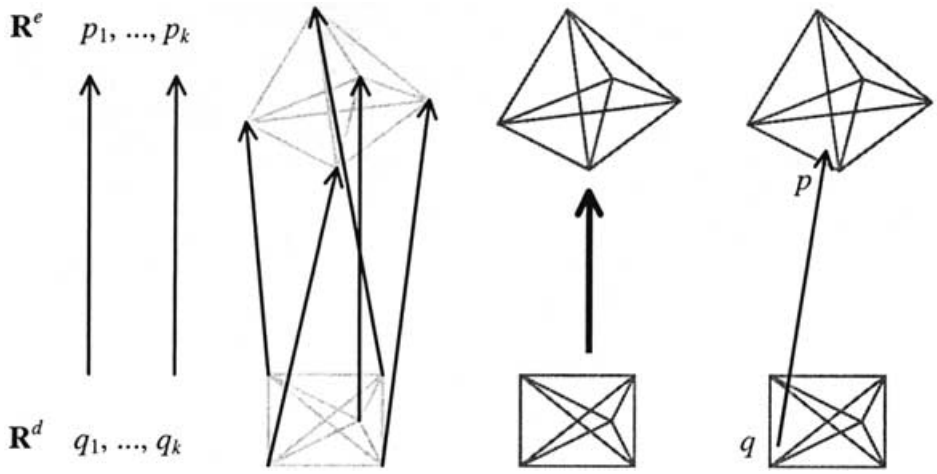


Figure 4. Pointwise map extended to a simplicial map, used to map an arbitrary query point q in $\mathbf{R}^d$ to an image point p in $\mathbf{R}^e$.

of the q_i is handled by partitioning $\mathbf{R}^d$ with a set of *ray-simplices*. Instead of one of the already described simplices we choose the unique ray-simplex which contains q, and then compute barycentric coordinates as usual with respect to that ray-simplex (figure 5). First we place a central point C in $\mathbf{R}^d$, at the centroid of the smallest axially aligned hyper-rectangle containing the hull (the hull's 'bounding box'). Since the hull is convex, C is in fact contained in the hull. Consider a face of the hull, i.e. a (d–1)-simplex on the boundary of the hull. Together with C, its vertices define a d-simplex. The set of all such simplices partitions the hull; but if we extend each simplex away from C (imagine rays extending outwards from C through all the vertices of the hull), the resultant ray-simplices in fact partition all of $\mathbf{R}^d$. (We ignore overlapping faces of adjacent ray-simplices, since their barycentric coordinates agree there). Formally, we define the ray-simplex (σ, V) of a simplex σ and a vertex V of σ as those points whose barycentric coordinates with respect to s are all non-negative, with the possible exception of the coordinate corresponding to V. There are many ways to choose the point in $\mathbf{R}^e$ corresponding to C and thereby induce corresponding ray-simplices which partition $\mathbf{R}^e$; the particular way does not affect the continuity and piecewise linearity of this extension to the mapping. The point in $\mathbf{R}^e$ corresponding to C we conveniently define one dimension at a time: its i^{th} coordinate is the median of the i^{th} coordinates of each point on the boundary of the hull in $\mathbf{R}^e$.

We now analyse the complexity of the algorithm. Initialisation consists of running the genetic algorithm or Sammon's mapping (both use $O(k)$ time and space); computing a Delaunay triangulation (also $O(k)$, based on an efficient implementation by Clarkson, Mehlhorn and Deidel (1993)); and initialising the bucket-search algorithm. The bucket search works only for the case $d = 2$, and initialises in $O(k)$ time and space; in short, initialisation is entirely $O(k)$. For larger d a generalised bucket search takes $O(k^3d^3)$ time and $O(k^2)$ space, not linear but still eminently practical (Goudeseune 2001: 145). During the running phase, the total time complexity for mapping a point from $\mathbf{R}^d$ to $\mathbf{R}^e$ is $O(d^3 + de)$ (Goudeseune 2001: 145). This is well behaved for large e, and does not increase at all as points are added (as k increases).

Since the domain of the simplicial interpolator is the unbounded totality of $\mathbf{R}^d$, its range may also be unbounded. This may not be compatible with the synthesizer's inputs, where bounds such as non-negative amplitudes and stable filter coefficients may apply. Three methods can ensure bounded output from a simplicial interpolator (or any generator of unbounded data, for that matter). (i) If a real-valued component of $\mathbf{R}^e$ must remain within the closed interval [a, b], it can simply be clamped: if out of range, instead use a or b as appropriate. (ii) If the endpoints must be excluded, a sigmoid-shaped function like an arctangent monotonically maps $\mathbf{R}$ to (a, b). (iii) In some cases valid input cannot be broken down into individual dimensions of $\mathbf{R}^e$, because of interdimensional constraints (stable filter coefficients, for example). If the valid subset V of $\mathbf{R}^e$ can still be computed *a priori*, dilate a maximal convex subset of V (ideally V itself) to cover an e-rectangle. This reduces the problem to one solvable by (i) or (ii): the output of the simplicial interpolator is some point p in $\mathbf{R}^e$. Applying a clamp or sigmoid to each coordinate of p moves p into this e-rectangle. Inverting the dilation then carries p into V, thereby making p a valid input for the synthesizer.

Source code for simplicial interpolation, including automatic generation of pre-image points, is available at <http://zx81.isl.uiuc.edu/interpolation/>.

4.2.3. Other interpolators

Bowler, Manning, Purvis and Bailey (1990) present a way to define continuous piecewise linear mappings from $\mathbf{R}^d$ to $\mathbf{R}^e$. The technique takes as input a d-lattice whose vertices are analogous to the q_i of simplicial interpolation. At each point of this lattice is stored the value

https://doi.org/10.1017/S1355771802002029 Published online by Cambridge University Press

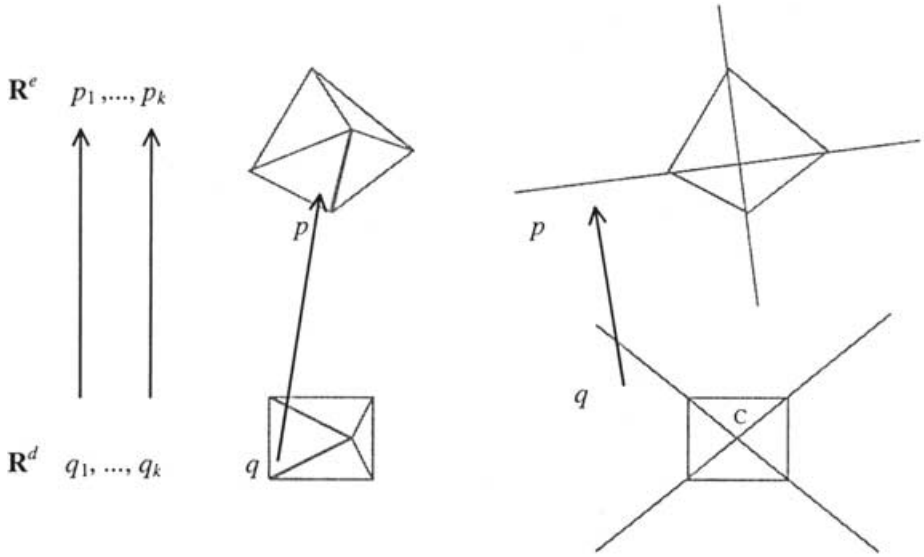


Figure 5. The centroid C and the four edges of the hull of the q_i induce four ray-simplices, one of which is shaded. If q lies outside the hull, its mapping to p is defined in terms of corresponding ray-simplices instead of corresponding simplices.

of the corresponding point p_i. This defines a pointwise map from $\mathbf{R}^d$ to $\mathbf{R}^e$. A naive extension from this pointwise map to a continuous map would first find which cell of the lattice contains a given point q in $\mathbf{R}^d$, and then construct the image p of q by interpolating among the p_i-values associated with each of the cell's 2^d vertices. By dividing each lattice cell canonically into d-simplices they reduce the number of points to interpolate among from 2^d to d+1. In fact this turns out to be a special case of simplicial interpolation, with the q_i arranged in a lattice and using a fixed (not necessarily Delaunay) triangulation.

A lower bound for computing the image of q has been shown to be $O(d^3 + de)$, the same as the exact cost using simplicial interpolation (Goudeseune 2001: 147–8). So Bowler's interpolator is no faster than simplicial interpolation. It differs in that it constrains the q_i to be a regular d-lattice. Simplicial interpolation needs far fewer points than this lattice. If a particular application already has this d-lattice constraint and can afford its high $O(2^d)$ memory usage, Bowler's simpler algorithm may be indicated. Otherwise simplicial interpolation is preferred for its greater generality and flexibility.

Bilinear (trilinear, multilinear) interpolation (Watson 1992: 139) differs from simplicial interpolation in that the number of control parameters is not constant but rather increases as $\log_2$ of the number of data points (which value must be a power of 2). Bilinear interpolation may therefore be preferred when the number of data points is fixed to be 4 or 8, or if like Bowler's interpolator the q_i are constrained to a lattice (in which case interpolation is done cell by cell, as in Rovan *et al.* (1997)). It is even simpler than Bowler's algorithm, but becomes intractably slow for large d since each cell has 2^d points among which to interpolate.

Neither of these interpolations deeply uses the structure of $\mathbf{R}^e$. Since the actual operation of interpolating is computing p as a weighted sum of p_i, the only operations required on $\mathbf{R}^e$ are scalar multiplication and (vector) addition. Since the real line itself has these operations, mapping to $\mathbf{R}$ and mapping to $\mathbf{R}^e$ are structurally equivalent here.

CONCLUSION

Stated colloquially, reducing how many dimensions of control an instrument has makes it less frightening to its performer. More formally, such a reduction concentrates the set of all possible inputs into a more interesting set by avoiding the redundancy inherent in the exponential growth of increasing dimensionality. Even more formally, it reduces the dimensionality of the set of synthesis parameters to the dimensionality of the set of perceptual parameters: it rejects all that the performer cannot actually understand and hear, while performing. Designing an instrument around the performer's cognition/perception instead of the engineer's convenience is echoed by Jacob, Sibert, McFarlane and Mullen (1994) in the context of visual tasks: they conclude that 'choosing an input device for a task requires looking at the deeper perceptual structure of the task, the device, and the interrelationship between task and device'.

The value of any dimension-reducing controller is found exactly in how well it loses information. A controller based on a custom pointwise mapping extended to a continuous mapping by simplicial interpolation precisely defines what information is lost; a controller which tries to preserve all information (such as a complete set of linear sliders) effectively still loses information because it is difficult to use. A synthesizer with many degrees of freedom can be played *ad hoc* by a finitely attentive human performer, exploring first this and then that region of its parameter space. But richer

https://doi.org/10.1017/S1355771802002029 Published online by Cambridge University Press

music is more likely if the instrument's rate of information consumption is systematically matched to its performer's rate of production. Put another way, controlled loss of information is about discovering what the performer can and cannot do, about matching that dividing line with the one between expressive and inexpressive.

REFERENCES

Alexandroff, P. 1961. *Elementary Concepts of Topology*, trans. A. Farley. New York: Dover.

de Berg, M., van Kreveld, M., Overmars, M., and Schwarzkopf, O. 1997. *Computational Geometry: Algorithms and Applications*. Berlin: Springer.

Bowler, I., Manning, P., Purvis, A., and Bailey, N. 1990. On mapping *n* articulation onto *m* synthesiser-control parameters. *Proc. of the Int. Computer Music Conf.*, pp. 181–4.

Buxton, W. 1986. There's more to interaction than meets the eye: some issues in manual input. In Norman and Draper (eds.) *User Centered System Design: New Perspectives on Human-Computer Interaction*, pp. 319–37. Hillsdale, NJ: Erlbaum.

Buxton, W., Hill, R., and Rowley, P. 1985. Issues and techniques in touch-sensitive tablet input. *Proc. of SIGGRAPH*, pp. 215–24.

Chen, E. 1999. Six degree-of-freedom haptic system for desktop virtual prototyping applications. *Proc. of the First Int. Workshop on Virtual Reality and Prototyping*, pp. 97–106. Laval, France.

Choi, I., Bargar, R., and Goudeseune, C. 1995. A manifold interface for a high dimensional control space. *Proc. of the Int. Computer Music Conf.*, pp. 385–92.

Clarkson, K., Mehlhorn, K., and Deidel, R. 1993. Four results on randomised incremental constructions. *Computational Geometry: Theory and Applications*, pp. 185–221. <http://cm.bell-labs.com/netlib/voronoi/hull.html>.

Collins Jr., F., and Bolstad, P. 1996. A comparison of spatial interpolation techniques in temperature estimation. *Proc. of the Third Int. Conf./Workshop on Integrating GIS and Environmental Modelling*. Santa Barbara: National Center for Geographic Information and Analysis. <http://www.ncgia.ucsb.edu/conf/SANTA_FE_CD-ROM/sf_papers/collins_fred/collins.html>.

Dean, T., and Wellman, M. 1991. *Planning and Control*. San Mateo, CA: Morgan Kaufmann.

Eckstein, B. 1989. Evaluation of spline and weighted average interpolation algorithms. *Computers and Geosciences* **15**(1): 79–94.

Edahiro, M., Kokubo, I., and Asano, T. 1984. A new point-location algorithm and its practical efficiency – comparison with existing algorithms. *ACM Transactions on Graphics* **3**(2): 86–109.

Garnett, G., and Goudeseune, C. 1999. Performance factors in control of high-dimensional spaces. *Proc. of the Int. Computer Music Conf.*, pp. 268–71.

Goudeseune, C. 2001. *Composing with Parameters for Synthetic Instruments*. DMA thesis, Urbana-Champaign, IL: University of Illinois. <http://zx81.isl.uiuc.edu/camilleg/dissertation>.

Haken, L., Fitz, K., Tellman, E., Wolfe, P., and Christensen, P. 1997. A continuous music keyboard controlling polyphonic morphing using bandwidth-enhanced oscillators. *Proc. of the Int. Computer Music Conf.*, pp. 375–8.

Hardy, R. 1990. Theory and applications of the multiquadric-biharmonic method. *Computers and Mathematics with Applications* **19**: 163–208.

Hunt, A., and Kirk, R. 1999. Radical user interfaces for real-time control. *Proc. Euromicro 1999* **2**: 6–12. Los Alamitos, CA: IEEE Computer Society.

Hunt, A., and Kirk, R. 2000. Mapping strategies for musical performance. In Wanderley and Battier (eds.) *Trends in Gestural Control of Music*. Paris: IRCAM, Centre Pompidou.

Hunt, A., Wanderley, M., and Kirk, R. 2000. Towards a model for instrumental mapping in expert musical interaction. *Proc. of the Int. Computer Music Conf.*, pp. 209–12.

Hutchinson, M., and Gessler, P. 1994. Splines – more than just a smooth interpolator. *Geoderma* **62**: 45–67.

Jacob, R., Sibert, L., McFarlane, D., and Mullen Jr., M. 1994. Integrality and separability of input devices. *ACM Transactions on Computer-Human Interaction* **1**(1): 3–26.

Kohonen, T. 1997. *Self-organizing maps*, second edition. Berlin: Springer.

Krige, D. 1981. Lognormal-de Wijsian geostatistics for ore evaluation. South *African Institute of Mining and Metallurgy Monograph Series: Geostatistics I*. Johannesburg: South Africa Institute of Mining and Metallurgy.

Mitas, L., and Mitasova, H. 1999. Spatial interpolation. In P. Longley, M. Goodchild, D. Maguire and D. Rhind (eds.) *Geographical Information Systems: Principles, Techniques, Management and Applications*, pp. 481–92. Cambridge: GeoInformation International.

Mitasova, H., Mitas, L., Brown, W., Gerdes, D., Kosinovsky, I., and Baker, T. 1995. Modelling spatially and temporally distributed phenomena: new methods and tools for GRASS GIS. *International Journal of Geographical Information Systems* **9**(4): 433–46.

Oliver, M., and Webster, R. 1990. Kriging: a method of interpolation for geographical information systems. *International Journal of Geographical Information Systems* **4**(3): 313–32.

Rovan, J., Wanderley, M., Dubnov, S., and Depalle, P. 1997. Instrumental gestural mapping strategies as expressivity determinants in computer music performance. *Kansei, the Technology of Emotion. Proc. of the Associazione di Informatica Musicale Italiana Int. Workshop*, pp. 68–73.

Ryan, J. 1992. Effort and expression. *Proc. of the Int. Computer Music Conf.*, pp. 414–16.

Sárközy, F. 1998. *GIS Functions–Interpolation*. <http://www.agt.bme.hu/public_e/funcint/funcint.html>.

Shepard, D. 1968. A two-dimensional interpolation function for irregularly spaced data. *Proc. of the National Conf. of the Association for Computing Machinery* **23:** 517–24.

Sheridan, T., and Ferrell, W. 1981. *Man–Machine Systems: Information Control and Decision Models of Human Performance*. Cambridge, MA: MIT Press.

Vertegaal, R., Eaglestone, B., and Clarke, M. 1994. An evaluation of input devices for use in the ISEE human-synthesiser interface. *Proc. of the Int. Computer Music Conf.*, pp. 159–62.

Vertegaal, R., Ungvary, T., and Kieslinger, M. 1996. Towards a musician's cockpit: transducers, feedback and musical function. *Proc. of the Int. Computer Music Conf.*, pp. 308–11.

https://doi.org/10.1017/S1355771802002029 Published online by Cambridge University Press

Wanderley, M., Viollet, J.-P., Isart, F., and Rodet, X. 2000. On the choice of transducer technologies for specific musical functions. *Proc. of the Int. Computer Music Conf.*, pp. 244–7.

Watson, D. 1992. *Contouring: A Guide to the Analysis and Display of Spatial Data*. New York: Pergamon Press.

Weisstein, E. 1999. *Eric Weisstein's World of Mathematics*. <http://mathworld.wolfram.com>.

https://doi.org/10.1017/S1355771802002029 Published online by Cambridge University Press