A FRAMEWORK FOR THE DEVELOPMENT AND EVALUATION OF GRAPHICAL INTERPOLATION FOR SYNTHESIZER PARAMETER MAPPINGS

Darrell Gibson
Faculty of Science & Technology,
Bournemouth University, UK
dgibson@bournemouth.ac.uk

Dr Richard Polfreman
Faculty of Arts and Humanities,
University of Southampton, UK
r.polfreman@soton.ac.uk

ABSTRACT

This paper presents a framework that supports the development and evaluation of graphical interpolated parameter mapping for the purpose of sound design. These systems present the user with a graphical pane, usually two-dimensional, where synthesizer presets can be located. Moving an interpolation point cursor within the pane will then create new sounds by calculating new parameter values, based on the cursor position and the interpolation model used. The exploratory nature of these systems lends itself to sound design applications, which also have a highly exploratory character. However, populating the interpolation space with "known" preset sounds allows the parameter space to be constrained, reducing the design complexity otherwise associated with synthesizer-based sound design. An analysis of previous graphical interpolators is presented and from this a framework is formalized and tested to show its suitability for the evaluation of such systems. The framework has then been used to compare the functionality of a number of systems that have been previously implemented. This has led to a better understanding of the different sonic outputs that each can produce and highlighted areas for further investigation.

1. INTRODUCTION

A fundamental problem of synthesizer programming is knowing how to set the parameters to create a certain sonic output. Many synthesizers have a large number of parameters and although having direct access to every parameter (*one-to-one* mapping) gives very fine control of the sounds, it complicates the process of designing new sounds. Alternatively, it is possible to map a smaller number of control parameters to a larger number of synthesizer parameters (*few-to-many* mapping) to reduce the control complexity. One way in which this can be done is to use *interpolation*, where sets of parameter values ("presets") for known sounds can be assigned in a point-wise manner to the control variables of a suitable controller. Then as the control variables are changed, via the controller, interpolation generates new values for the synthesizer parameters. In this way, it is possible to create sonic outputs that are constrained by the known sounds and the control changes. This provides a mechanism for exploring a defined interpolation space.

A number of such interpolation systems have been developed, but the systems of particular interest in this body of work are those that use a graphical interface for the interpolation control. These map presets of synthesis parameters to specific locations in a (normally) two-dimensional pane and the system calculates interpolated parameter values for the *interpolation point* (cursor) position as it moves between the preset locations. This facilitates the discovery of new "custom" parameter values that blend characteristics of two or more parameter presets. The resulting sounds are a function of the interpolation model used, the parameter presets, their locations within the interpolation space, the position of the interpolation point [1] and the synthesis engine itself. It is also possible to define trajectories for the interpolation point that result in new sonic gestures.

Of particular interest here, is the use of such interpolators for sound design, which in this work is taken to be the design of new sounds, often to accompany visual or other media. Sound design is a creative process and as a result there is a desire to remove or minimize any technical barriers between the creative artist and sonic results. A large part of the creative process involves generation and exploration [2], so it is desirable to provide a platform that supports this paradigm effectively.

2. PREVIOUS WORK

Over a period of many years a number of graphical interpolation systems have been developed for use with synthesizer/sound processing technology. A summary of these is given in the following sections and an evaluation is undertaken that results in the formulation of a framework. Figure 1 shows the visual representation for each interpolator reviewed.

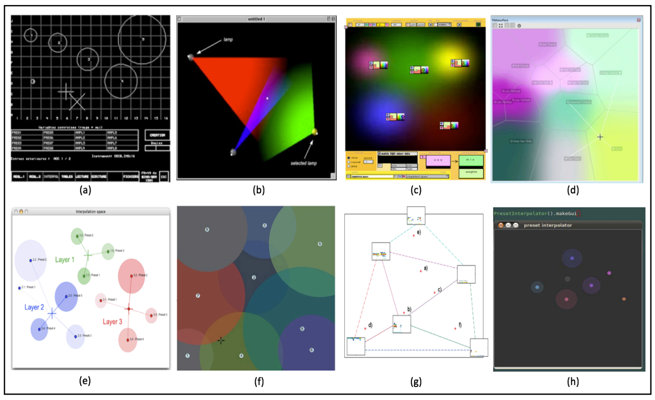

Figure 1 Graphical Interpolator Models

2.1 SYTER

Work in this area was first completed at GRM (Group de Recherches Musicale) in the early 1980's on the SYTER system, a hardware workstation that was designed to allow real-time audio processing and synthesis. SYTER had a two-dimensional graphical interface, which offered a user-friendly real-time control window, called INTERPOL [3] to control the relationship between different parameter presets in a real-time sound-processing engine (Figure 1a). The positions of points on the visual interface are mapped to presets of up to 16 parameters. Each preset has a circular representation (a *planet*) in the interpolation space and clicking on a planet recalls the sound of the corresponding preset. However, the system also allows interpolation between presets using a gravitational model, where influence varied with the distance between the planets and their size. In this way, larger planets have a higher gravitational force and so a stronger field-of-influence compared to smaller planets. The work has been expanded over the years for three-dimensional graphical control and a generalized Inverse Weighted Distance (IWD) model [4], where the exponent value can be user controlled.

2.2 Interpolator

The SYTER style gravitational model for interpolation was further expanded with a system called Interpolator, which was developed in collaboration between GRM and University of Hertfordshire in the early 2000's [5]. This prototype system was designed as a graphical control interface for the GRM Tools software plug-ins. The system used a light model for the interpolation, where presets were represented as *lamps*, with each having an angle, aperture and extent of the light source (Figure 1b). The light beams gave a visual representation of the corresponding preset's field-of-influence. In addition, if the angle of a lamp's aperture is opened up to 360 degrees then it becomes similar to the planetary system, except the lamp shows the field-of-influence and the planet's area is not lost from the interpolation space. Users could then explore interpolated sounds where the lamp's light beams intersected. Different colours (up to 4) were used to signify different mapping layers for the interpolation. Hence, a colour represented a set of parameters mapped to either single or multiple GRM Tools plug-ins (up to 4) and a lamp is a specific set of values for the parameters. This design allowed layers to be created in the interpolation on the same or different plug-ins.

2.3 Gaussian Kernels

In 2003, Momeni defined a system that allowed the spatial layout of objects that relate to musical material – either recorded samples or synthesis parameters. Each of these can be placed at locations within a two-dimensional graphical pane and represent a Gaussian kernel, whose value at any given point in the pane indicate the weight of the associated preset point in the interpolation. This allows weighted interpolation among the preset points based on the values of the Gaussian kernels at each point in the interpolation space. For each kernel the user could modify the location, amplitude and standard deviation [6]. The interpolation space then shows a two-dimensional visual representation of all the kernels in the space and the kernel's amplitudes are mapped to the brightness scale of a selected colour (Figure 1c). For more accurate visualization the Gaussian kernels can also be viewed as a three-dimensional image. The space created can be explored and interpolation between the presets is calculated based on the cursor position and weight of the kernels. The Gaussian kernels provided not only a mechanism for interpolation, but also for extrapolation beyond the perimeter of the points specified in the space. As this system was implemented in the visual programming environment Max, it is possible to control any sound engine that can be created in it.

2.4 Metasurface

In 2005 the Metasurface was developed as a control interface for the AudioMulch Interactive Music Studio, a software application for live performance, audio processing, sound design and music composition [7]. Metasurface can be used to control synthesis and processing parameters and allows any number of parameter presets to be defined and placed in the interpolation space. When the presets are placed in the interpolation space a Voronoi tessellation is constructed where each preset is at the centre of a convex polygon (Figure 1d). Any position contained in each polygon is closer to the centre point of that polygon, than the centre point of any another polygon. Moving the cursor within the tessellated pane performs natural neighbour interpolation. This is calculated by adding a new polygon for the current cursor position and the weight of each neighbour is then calculated as the area "stolen" from the neighbours by the polygon centred at the cursor position. Moving the cursor results in smooth interpolation between the cursors natural neighbour as it moves through the space.

2.5 INT.LIB

Work in the mid 2000s saw the SYTER style gravitation model revived, updated and expanded by Oliver Larkin. INT.LIB is a library for Max that allows the control of multiple layered presets using a gravitational model (Figure 1e). Each layer is color-coded and has its own cursor that indicates the location of the interpolation point for that layer [8]. Optionally the interpolation points can be linked so that all layers are controlled simultaneously. Each of the layers has its own instances of a synthesizer or signal processing plug-in and allows interpolation between a number of patches on that sound engine. As INT.LIB is implemented in the Max environment it again means it has open ended possibilities for the synthesis engine.

2.6 Nodes

Andrew Benson, a visual artist, created the *nodes* object for Max in 2009 and it proved so popular that it has been included in subsequent Max distribution (Figure 1f). When combined with the *pattrstorage* object it can provide a graphical interpolation system. Although the *nodes* system uses a distance-based interpolation function, it uses a different model, where each preset is represented as circular node within the interpolation space. The interpolation is only performed in regions where the nodes intersect. When the cursor is inside a node the distance to the node's centre is used as the weighting for the corresponding preset [9].

2.7 Spike-Guided Delaunay Triangulation

In 2009 Drioli et al., developed another graphical interpolation scheme as a sound design interface for physically-based synthesis models. A visual spike representation for the sonic output of each synthesized preset can be positioned in an interpolation pane (Figure 1g). The presets form a scatter of points on the graphical pane and interpolation is performed based on a Delaunay triangulation of the points. The user can select points in the space and the synthesizer parameters are calculated through linear interpolation of the three presets of the containing triangle [10].

2.8 Intersecting N-Spheres Interpolation

Developed by Martin Marier in 2012, Intersecting N-Spheres Interpolation is a mapping strategy for interfaces including multiple continuous sensors [11]. This system uses a two-dimensional space where the presets and interpolation point are positioned. The visual representation shows a circle around the interpolation point, with a radius equal to the distance of the nearest preset point. Circles are also drawn around each preset point, with the radii of these circles being equal to the distance to the nearest preset location or the interpolation point, whichever is nearest (Figure 1h). Any preset point circles that intersect the interpolation circle are considered neighbours and influence the interpolation. The value of the interpolation point is calculated as a weighted average of the value equal to the ratio of intersecting circles area. This system is realized in SuperCollider where it can control the audio processing and synthesis parameters running on this platform.

3. EVALUATION OF GRAPHICAL INTERPOLATORS

Although the systems examined in Section 2, have all been created to allow interpolated control of parameters they represent different realizations, are implemented with different technologies and have a number of different application areas. Nonetheless, there is a common thread, in that interpolation allows the adjustment of sound parameters between defined presets, via some form of visual model. From the systems examined it is apparent that they can be decomposed into five different, but dependent areas that should be considered when developing such systems. These are:

1. Control – input controls of the interpolation model
2. Visual Metaphor – the visual interpolation model and how it is represented graphically
3. Interpolation – the interpolation weighting calculations
4. Mappings – the synthesis parameters that are interpolated
5. Synthesis – type/architecture/implementation of the sound engine

Each of these areas will be considered separately, however, for a number of the systems examined in Section 2 there was not a clear partitioning between them. In addition, in this work they will be considered in a sound design context.

3.1 Interpolation Control

There have been many different ways of controlling interpolation systems. Here these have been constrained to those offering a graphic interface that corresponds to the visual interpolation model. Many of the older systems had two modes of operation: one for the creating and editing the interpolation space and another for actually performing the interpolated sonic output [2, 5, 7, 12]. This meant that the interpolation space could not be changed in the middle of a sonic exploration, without changing mode. However, if the interpolation calculations and graphics updates can be performed real-time it opens up the possibility of being able to control the interpolated sonic output, either by changing the location of the interpolation point within the space or by modifying the interpolation space itself: moving the preset locations or adding and deleting presets. Moreover, with the layered interpolation system presented in INT.LIB, it is possible to have multiple interpolation points and these can either be moved individually or linked so they can all be moved simultaneously [8].

With the possibility of altering the interpolation space in real-time, it is also worth considering the input mechanism for controlling a graphical representation. Using traditional computer-based spatial control devices (mouse, drawing tablet, joystick, trackball, etc.) only one point can be controlled at a time. This means that only one preset position or the interpolation point can be moved at a time. Whereas with multi-touch screen technology it opens the possibility that sound design can be undertaken by simultaneously controlling multiple points in the interpolation space. Being able to change the interpolation space real-time and multi-touch technology, opens the following potential modes of operation for changing an interpolated sound, creating new possibilities for the control of an interpolated sound design process:

1. Move the interpolation point(s)
2. Change the field of influence for one or more preset
3. Simultaneously move one or more preset locations, while the interpolation point remains static
4. Simultaneously move the interpolation point and one or more preset locations

Discontinuities in the interpolation space are not normally desirable for this application domain, as noted by other authors [7, 12]. However, if the user does want to produce audible jumps either a new instantaneous position for the interpolation point can be selected (a jump) or if real-time mode is available the position of the presets could be instantaneously changed.

For sound design applications, the addition of performance expressions is often desirable to “bring the sounds to life”, e.g. to match the sound to on-screen actions. It has already been shown that interpolation methods provide an opportunity to apply expressive control to synthesized sounds [12, 13]. However, it does not necessarily follow that expressive control of the interpolation should be performed at the same time as the design of the base sound. It may also be the case that more traditional avenues for applying expressions will still be preferred, for example, through physical actions [14].

3.2 Visual Model & Graphical Representation

As can be seen by the range of systems examined, there have been many proposed visual metaphors for graphical interpolators. The visual model provides the user with feedback on the state of the interpolation method, location of the presets and their relative influence. This is delivered in addition to the auditory feedback generated by the synthesizer output. However, for a sound design task it is not clear if the visual representation is needed or actually aids the process.

In the systems examined there are different visual representations for the presets within the interpolation space, often using geometric shapes: circles, triangles, polygons, etc. However, there tends to be some form of visual linkage between these representations and the actual interpolation model. For example, circles have been used to represent presets in a number of different interpolation systems [3, 8, 9, 11], but the way they are interpreted is directly linked to the interpolation paradigm being used in each case. In some, the shapes used in the visualization are linked to which presets are included in the interpolation calculations. For example, where triangulations are generated between the preset locations it provides the implication that the interpolation is being performed between the three presets of an enclosing triangle [10], a rectilinear grid implies interpolation between four local presets and polygons implies interpolation between the closest presets that form a convex hull around the interpolation point [7]. Even a straight line (slider) can be used to imply interpolation between two presets (a 1-D interpolation space). For intersecting interpolation paradigms, where the interpolation is performed when preset objects overlap in the space, the intersection itself implies which presets are included in the interpolation [5, 9]. In other cases, the sounds included in the interpolation are shown by links between the interpolation point and the presets [8].

As well as different geometric representations for presets in the interpolation space, colour is also used in the majority of the systems examined. In most cases the colour is used to differentiate between the presets within the interpolation space. However, in some cases the colours or shadings are visually interpolated to give a visual cue for the interpolated values between the presets [1, 5, 7]. On the multi-level interpolation systems, Interpolator and INT.LIB, colour is used to distinguish between different layers in the interpolation space [5, 8], but the influence of each preset is provided by linking the visual transparency of the preset’s display colour. In this way, a solid colour shows a preset has a high degree of influence and it becomes more transparent as the influence decreases. This is also the case for the linkage lines that show which presets are included in the interpolation, although the base colour already provides this information.

It is also worth noting that with most of the systems examined the visual representation relates to the interpolation model (*parameter space*) and not the systems sonic output (*sound space*). As seen through the work on *timbre space,* it is possible to use a sound-based representation for the control the synthesis parameters [15]. Although work has continued in this area, for sound design applications, the use of a predefined timbre space may be restrictive. The visual interpolator systems examined do not use automatic positioning of presets within the interpolation space. Instead the user can define the presets that will be used in the interpolation space, the positional relationships between them in the space and in some cases the influence of individual presets. These aspects allow sound designers to constrain the sonic output, while also supporting the exploratory nature of a design process [7, 12].

Finally, with most graphical interpolation systems, individual presets can be recalled by positioning the interpolation point cursor directly on the preset’s position. However, with the SYTER gravitational model, the gravity remains the same while on the planet’s surface so any position on the planet will recall that preset [3]. As a result, the area of each planet effectively reduces the potential size of the interpolation space [5]. Conversely, using the Max *nodes* object, the associated preset can only be recalled by clicking on an area of the node that does not intersect with another node. If a non-intersecting area does not exist then it is not possible to hear the defining sound.

From this analysis, the systems already created have used the following visual cues in the interpolation space:

1. Preset handle (location in the space)
2. Preset field-of-influence
3. Interpolation point(s)
4. Number of presets included in the interpolation
5. Interpolation strength at the interpolation point
6. Navigable space

3.3 Interpolation Methods

A variety of methods have been used to calculate the interpolation values between the presets. For example, linear, power, regularized spline with tension, etc. The method chosen will affect the sensitivity and “feel” of the interpolation system, as was demonstrated with LoM [16]. However, it is not clear how the system’s control, visual model, parameter mapping and synthesis engine combine to affect the feel.

As already noted in Section 3.1, it is desirable that an interpolation system should produce a “smooth” sonic output that does not possess discontinuities or overshoots. Therefore, the interpolation function should be smooth to provide even changes and variation to the synthesis parameters and so the sonic output. Although, in some situations jumps maybe required, this should be under user control and not occur unexpectedly as a result of the interpolation method.

3.4 Parameter Mappings

Although the use of interpolation gives the user a mechanism to adjust multiple parameters simultaneously between preset values, the sonic output is defined by which parameters are mapped to the interpolation points. Interpolating all the parameters within a set of presets can create large sonic changes, whereas a mapping that contains a subset of parameters offers more focused control. Moreover, with some forms of synthesis there is not a simple link between the synthesis parameters and the sonic output. As a result, selecting mapped parameters to create a specific sonic result can be difficult. Previous research into the mapping of synthesis parameters has tended to focused on musical outputs and instrument gestural control [12, 13]. This is a different application area and the outcomes have been fairly broad, not considering specific relationships. Multi-layered mappings have been proposed, where intermediate abstract parameters can be used [12], but for interpolation systems being used in musical instrument design. Nonetheless a number of desirable characteristics have been identified for the mappings, such as, differentiability, linearity, range space, exactness, extensibility and editability [13].

With the graphical interpolation systems examined, the mapping between the synthesis engine and the interpolation points is often controlled by the user. This is done by presenting the user with a list of parameters and allowing them to select the desired parameters to map between the visual interface and the synthesis engine. Although this process gives control to the sound designer, completely different sonic outputs will be generated depending on which parameters are selected and those that are not. With the majority of the systems examined one set of mappings is controlled by the graphical interface, however, both Interpolator and INT.LIB considered a multiple mapping approach offering simultaneous control [5, 8]. With INT.LIB, each mapping was sent to a different sound module so different sounds could be layered and controlled separately. Whereas with Interpolator it also allowed multiple mappings to be associated to the same sound module, which allows different aspects of a sound to be controlled independently.

3.5 Synthesis

From the range of systems examined, it can be seen that interpolation has been used with many kinds of audio processing and synthesis engines. A number of the earlier interpolation systems are directly integrated into the same platform as the synthesis engine. This means that although the sound can be changed within the remit of the given synthesis engine, it is not possible to use the same interpolation platform with a different synthesis engine. The later exceptions to this have been developed through programming environments [3, 8, 9]. The flexibility of using the programming environment means that it is possible to build new synthesis engines to be used with the interpolation system. Moreover, as the Max environment also supports use of common audio plug-in formats, it is possible to use many commercially available software synthesizers with interpolators built in Max [4, 8].

An interpolator user interface can mask the details of the synthesis and the associated parameter manipulation from the user, allowing the sound designer to concentrate on the design process, without having to worry about the underlying details of the synthesis engine.

4. GRAPHICAL INTERPOLATION FRAMEWORK

Although a number of graphical interpolation systems have been created and documented, they were developed over a thirty-five-year period, using different implementation platforms, different synthesis architectures and were designed for different application purposes. Consequently, many of the realizations used technologies that are now obsolete and no longer available making it impossible to do back-to-back evaluation between the original systems. In order to be able to evaluate the suitability of these graphical interpolation systems for the purpose of sound design, they require re-implementation on contemporary hardware and software platforms. This will allow direct comparisons to be undertaken between the different interpolation systems.

It is important to also consider the characteristics that a sound design graphical interpolator should ideally possess. The following summarizes the most important factors from the evaluation section:

1. Synthesis independent interpolation – the same interface can be used with different synthesis engines
2. Clear relationship between interpolation control and the sonic output – sound space defined by the populated parameter preset
3. Constrain the navigation and exploration of the parameter space – user selecting and positioning presets in the interpolation space
4. Control a number of parameters simultaneously – reduce the control complexity of many parameters
5. Changeable parameter mappings – provide user with control over the parameter mappings
6. Exploration of the sound space with both course and fine levels of detail – change resolution and precision
7. Smooth interpolation – no discontinuities unless user selects
8. Real-time interpolation (not different edit/interpolate modes) – allow either preset points or cursor to be moved to change sounds
9. Support the design of base sounds and the application of performance expressions
10. Usability, repeatability, predictability and playability – user can design a sound based on the supplied preset sounds.

In order to be able to evaluate these aspects of different graphical interpolators a hierarchical framework is proposed that compartmentalizes each of the system elements. This works from the control input at the top-level to the sonic output at the bottom, as shown in Figure 2. Although the final output, sound, is at the bottom level it is worth noting that the visual representation also gives the user visual feedback on the current configuration of the interpolation system and therefore, the sound. Equally the user maybe given inputs that allow the mappings to be modified. However, what the framework shows is the interdependencies of the different elements of an interpolation system and the relationships between them. For example, the sonic output from the synthesizer is dependent on the control inputs, the visual model, the interpolation function, the parameter mapping and the synthesis engine used. In addition, it is envisioned that in the future different realisations may be created that will still be encapsulated by the framework.

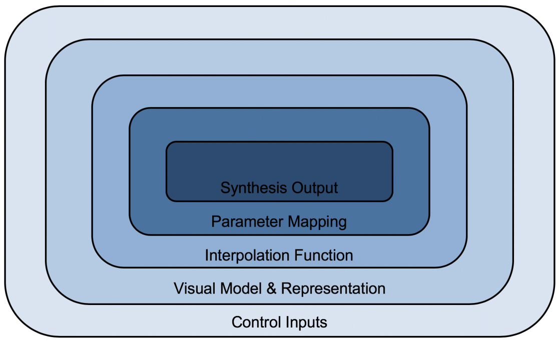

Figure 2 Graphical Interpolation Framework

4.1 Framework Structure

Having formalized the framework, the next stage was to consider an implementation. Using the framework defined in the previous section, it is possible to structure the different levels (control, visual model, interpolation, mappings and synthesis) into separate modules and test them separately. In this way, it becomes possible to directly compare these aspects of each system and evaluate their impact on the usability. This can be done through comparative user tests where only one element is changed at a time. The results can then be measured, compared and evaluated to determine the suitability of each for sound design applications. To facilitate this the framework has been implemented in the Max environment using the architecture shown in Figure 3.

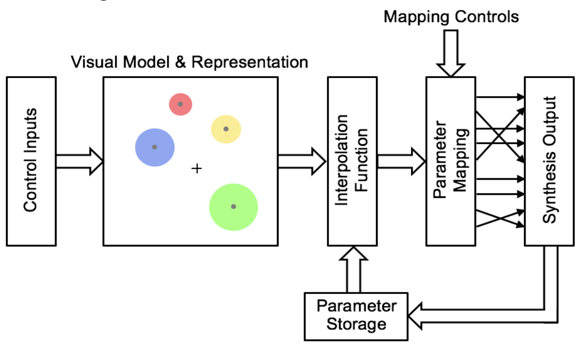

Figure 3 Framework Architecture in Max

4.2 Framework Implementation

As an initial investigation, a graphical interpolation system was built in Max using the *nodes* object detailed in Section 2.6. In this way, the nodes graphical interpolation system acted as proof-of-concept for the framework defined. When this interpolator system was implemented, care was taken to develop each of the five elements of the interpolation framework into separate entities. This was done through a modular design approach where each part is created as a separate module so that each can be modified independently of the others.

The first implemented was the **interpolation function** module, which is storage that holds the parameter values and performs the interpolation. The parameter values for each synthesis preset are stored as a new data set and it then interpolates between the parameter data sets, generating interpolated values for all the individual parameters. The interpolation is performed based on the modules input which is the relative weightings for each preset. By default, the calculation performed is linear interpolation, but it is possible to change the mode so that any interpolation function can be realized.

As the *nodes* object has been specifically designed as a graphical interpolator, the object has been created with specific functionality for the **visual model** and the **control inputs**. The **control inputs** realized in the *nodes* object are standard computer-based spatial controls. However, it is also possible to send the object positional input data from other sources. This provides the possibility of using other input devices to control the interpolation space. The interpolation point on the *nodes* object can be moved within the space and an output weighting for each node is generated. The **visual model** generated node weights are normalized (0.0 – 1.0) and are proportional to the interpolation point's distance from the circumference of a containing node to its centre. Therefore, when the nodes in the interpolation space overlap and the interpolation cursor is placed in an overlapped region, a weighting is generated for each node. (In Figure 4 - 1 = 24%, 2 = 0% & 3 = 76%).

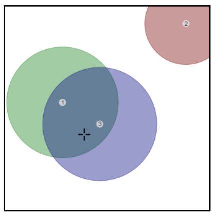

Figure 4 Nodes Outputs Normalized Distances

These weightings are used as the input to the **interpolation function**. As the visual interpolation model is encapsulated by a single object (*nodes*) it is possible to replace it with different implementations.

The **synthesis engine** has been constructed to be separate from the interpolation platform by using software plug-ins, allowing different (commercially available) synthesis engines to be loaded and tested. However, the framework would also allow bespoke synthesis patches to be used. When a new synthesizer is loaded, it is interrogated to determine all the parameter values for the number of presets loaded. Each preset is associated to a node in the interpolation space and all of the preset's parameter values are sent the **interpolation function** storage.

By default, all of the parameters for the presets are associated to the corresponding node and so every aspect of the sounds synthesis is controllable. However, the **parameter mappings** between the **interpolation function** and **synthesis engine** can be changed by user selection.

4.2.1 Framework Testing

The prototype nodes-based interpolator was initially tested to ascertain if each module built in the framework could be changed independently of the others and to establish the impact on sound design tasks. Through exploratory testing, where the nodes-based interpolator and its parameter space were left the same (shown in Figure 5), it became apparent that changes to each module in the framework leads to the system generating different sonic outputs and results in a different user experience with each realisation.

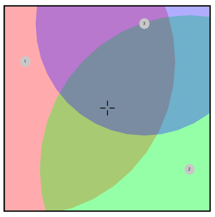

Figure 5 Nodes Prototype Test Layout

From testing with different **synthesis engines**, it was found that changes to the engine (preset changes, synthesis realisation or changes of synthesis type), were the main determinants of the sonic output. Moreover, with some forms of synthesis, changes to a single parameter can produce large sonic variations, but for others, more subtle alterations resulted. Changes to the **control inputs** allowed different mechanisms for interacting with the sonic manipulation, and potentially changing the usability of the interpolator. Modifications to the **parameter mappings** permitted the refinement of the sonic changes that it is possible to generate with the interpolator. Mapping lots of the synthesis parameters to the nodes resulted in big sonic changes, whereas mapping a few parameters permitted more subtle variations to be generated. Changing the **interpolation function** resulted the subtlest differences. The chosen function affects how the sound transitions as the interpolation point is moved between preset locations.

4.3 Graphical Interpolator Implementation

The prototype nodes-based interpolator was used as the basis for the subsequent development of different graphical interpolation systems. For each **visual model** and its control, the *nodes* object was replaced with an interactive user-interface built using OpenGL for the interpolation model's visual representation and JavaScript to create the control mechanism and calculate the preset weightings. Each model was constructed and integrated with the other elements of the framework for testing. To-date six interpolators have been built, integrated with the framework and functionally tested. These are:

1. Nodes (Overlapping Circles)
2. Gravitational (Planets & Space)
3. Radius-based IWD (Scatter Points & Interpolation Point Circle)
4. Light (Lamps)
5. Delaunay (Triangulation)
6. Voronoi Tessellation (Polygons)

The nodes interpolator was reimplemented so that it could act as a benchmark for the other interpolators, but also so the visual representation can be changed to assess the influence of different visualisations using the same interpolation model. The other interpolators where chosen to represent the key traits of the interpolation systems that have been previously created.

4.3.1 Graphical Interpolator Testing

Following functional testing the different interpolators were back-to-back tested by placing the same ten presets at identical locations in each. The nodes interpolator was populated first and although the size of each node was randomly selected, they were chosen to ensure the whole space was covered. For the gravitational interpolator, while the same locations were used, this model requires space between the planets, where the interpolation is performed. However, so that each preset has the same relative influence as they do in the node interpolator, the sizes were scaled by one tenth of those in the nodes interpolator. For the radius-based IWD the interpolation point's radius was chosen to cover approximately 50% of the interpolation space so for all interpolation positions, multiple presets are enclosed by the radius. For the light interpolator although the same locations were used, as each lamp has an angle and aperture, it results in each lamp having a specific directionality. To try and give coverage over the whole interpolation space the extent of each lamp was scaled to four times the nodes size. Despite this the lamps directionality also needed to be selectively chosen to ensure the whole interpolation space was covered, whilst still giving a good spread of intersecting light beams. For the two remaining interpolators the presets do not have different influences or directionalities so the locations were kept the same as the nodes layout. The test layouts for the six interpolators are shown in Figure 6.

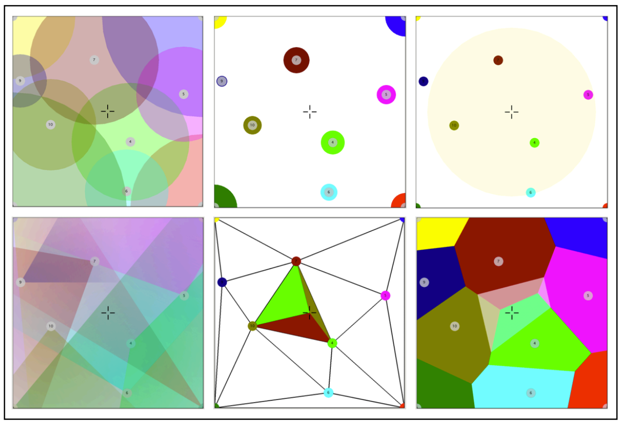

Figure 6 Test Layout for Graphical Interpolators

These layouts were used to perform back-to-back tests where output from the different graphical interpolators were compared. For the tests the **control inputs**, **interpolation function**, **parameter mappings** and **synthesis output**, all remaining the same, as detailed:

1. Control Inputs – Fixed 2-D movement of interpolation point only
2. Interpolation Function – Linear interpolation
3. Parameter Mappings – All synthesis parameters mapped to the corresponding preset location
4. Synthesis Output – Native Instruments *Massive* with ten presets loaded

The tests compared the sonic output from the different interpolation models for the same interpolation positions. This was first done by instantaneously moving the interpolation cursor to ten different locations and comparing the sound generated with each system. From this test, it was evident that each visual interpolator generated significantly different sonic results, despite being populated with the same preset sounds. To try and get a better understanding of each system's sonic nature, another comparative test was created, where the interpolation point was moved through a fixed trajectory path around the defined interpolation spaces. The path began at the centre of the space, moved diagonally towards the left-top corner until the mid-point and then moved around parallel to the outside edge of the space. It was found that each interpolator gives a very different range of sonic outputs across all interpolation positions. The fact they were different was not necessarily surprising, but the diversity of the sonic differences was not anticipated. Moreover, each interpolator results a completely different sonic palette that it can generate, meaning it is very difficult to create the same sound with each interpolator. This is because each interpolation model results in different preset weightings for the interpolation function. As an example, Figure 7 shows the preset weightings for just the centre position of each interpolation spaces, as shown in Figure 6.

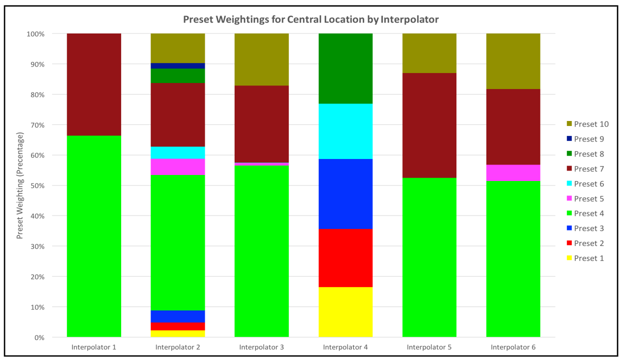

Figure 7 Comparison of Interpolator Preset Weighting's

In all cases, the relative positioning (layout) of the presets determines the interpolated outputs. Different layouts of the same presets results in different outputs being obtained. It was also noted that for interpolators 1, 2 & 4 the extent (size) of each preset, further changes the interpolations space. Also, the directionality of the lamps in interpolator 4 gives an added element for further modifying the interpolation space. For interpolators 3, 5 & 6 the influence of each preset is potentially the same, but the layout determines the relative strengths. However, for interpolator 3 this is constrained by the interpolation point's radius that determines which presets will be included. If the radius size is changed, corresponding presets will be added or removed from the interpolated output. Whereas interpolator 5 uses only the three closest presets and interpolator 6 uses the natural neighbours.

5. CONCLUSIONS

The framework presented has been shown to provide a suitable platform for the testing and evaluation of different graphical interpolation systems. The modularity of the framework components means that each can be modified independently of the others, offering a suitable mechanism for performing formal comparative user testing. In this way, the use of the framework has led to a more detailed understanding of different interpolation models and the identification of where and how sonic differences are obtained. From the testing that has been undertaken so far three areas have been identified for immediate further investigation. The first of these will be to undertake formal user testing to assess the level of feedback provided to the users by the visual representations of the interpolation model. The second will be to undertake formal user testing to evaluate the suitability of the presented interpolators for sound design applications. Finally, as different synthesis engines reactions to interpolation can be drastically different this will also be examined further.

6. REFERENCES

[1] J.J. van Wijk and C.W. van Overveld, "Preset based interaction with high dimensional parameter spaces,". In Data Visualization, 2003 (pp. 391-406).

[2] T. I. Lubart, "Models of the Creative Process: Past, Present and Future," Creativity Research Journal, 2001 Oct 1;13(3-4):295-308.

[3] T. Todoroff, "Control of digital audio effects," DAFX: Digital Audio Effects, 2002:465-97.

[4] T. Todoroff and L. Reboursière, "1-d, 2-d and 3-d interpolation tools for max/msp/jitter," In Proc. ICMC'09, 2009.

[5] M. Spain and R. Polfreman, "Interpolator: a two-dimensional graphical interpolation system for the simultaneous control of digital signal processing parameters," Organised Sound, 2001 Aug 1;6(02):147-51.

[6] A. Momeni and D. Wessel, "Characterizing and controlling musical material intuitively with geometric models," In Proc. of Conf. on NIME, 2003.

[7] R. Bencina, "The metasurface: applying natural neighbour interpolation to two-to-many mapping," In Proc. of Conf. on NIME, 2005, (pp. 101-104).

[8] O. Larkin, "INT. LIB–A Graphical Preset Interpolator For Max MSP," ICMC'07: Proc of ICMC, 2007.

[9] "nodes," Max Reference, Cycling 74, 2018.

[10] C. Drioli, P. Polotti, D. Rocchesso, S. Delle Monache, K. Adiloglu, R. Annies and K. Obermayer, "Auditory representations as landmarks in the sound design space," In Proc. of SMC Conf., 2009.

[11] M. Marier, "Designing Mappings for Musical Interfaces Using Preset Interpolation," In Conf. on NIME, 2012.

[12] C. Goudeseune, "Interpolated Mappings for Musical Instruments," Organised Sound, 7(2):85–96, 2002.

[13] A. Hunt, M. Wanderley, and M. Paradis, "The Importance Of Parameter Mapping In Electronic Instrument Design," Journal of New Music Research, Volume 32, Issue 4, page 429–440, 2003.

[14] K. Gohlke, D. Black and J. Loviscach, "Leveraging behavioral models of sounding objects for gesture-controlled sound design," In Proc. of 5th International Conf. on Tangible, embedded, and embodied interaction, 2011 Jan 22 (pp. 245-248). ACM.

[15] D. L. Wessel, "Timbre space as a musical control structure," Computer Music Journal, 1979 Jun 1:45-52.

[16] D. Van Nort and M, Wanderley, "The LoM Mapping Toolbox for Max/MSP/Jitter," In Proc of ICMC, 2006 (pp. 397-400).