SplineSynth: An Interface to Low Level Digital Audio

Nick Collins *n.collins@mdx.ac.uk* http://www.axp.mdx.ac.uk/~nicholas15/

Research Fellow, Centre for Electronic Arts, Middlesex University

Keywords- spline, softsynth, real-time, time-domain morph, interpolation synthesis

Abstract

This paper sets out a design for a real-time software synthesiser. This design springs from a central idea: that it is possible to give a direct graphical interface to low level digital audio using a mathematical object, the interpolating spline. The spline is a representation for one period of a waveform, a single wave-cycle of digital audio in the time domain. By manipulating the control points of this spline, the user can continuously vary the shape of the waveform. On that basis of continuous change, well known synthesis techniques can be given a front-end geared for the exploration of the shape of underlying audio sources. Furthermore, the notion of continuous change of waveform implies a morphing synthesiser, transforming smoothly from sound to sound.

To automate timbral change over time, the user is allowed to establish an ordering of splines. The splines, and hence the audio, will morph continuously looping through this order. Techniques are investigated to interpolate the splines themselves, without invoking the pervasive audio crossfade.

Background

This is not the place for an exhaustive history of synthesis; the interested reader is referred to Miranda (1998), Roads (1995), Moore (1990) for reviews of synthesis methods. The field of synthesis that concerns us here is waveform synthesis, or more specifically, breakpoint interpolation synthesis. An original theoretical source for the area of waveform synthesis is Mitsuhashi (1982a, 1982b). Mitsuhashi covers the spectral analysis of waveforms based on his breakpoint interpolating functions. Waveform synthesis has also been extended to two or more dimensions (Mitsuhashi 1982c , Borgonono and Haus 1986, Roads 1995 under Wave Terrain Synthesis). Whilst one can envisage real-time interfaces for these models few if any are currently implemented as real-time interactive methods. The softsynth that accompanies this paper is an attempt to fill this breach. The “SplineSynth” is available from the web address given in the references.

The growth of computer power now allows the production of real-time software synthesisers on a home computer. Amongst the wide variety of softsynths available on the internet, two for PC deserve a specific mention. A good web source for downloadable softsynths is given in the bibliography, and both those discussed below are available there at the time of writing.

SMoRPhi 3.0 by Andre Karwath (1997) has similar aims to the model of this paper. It is a non-real-time synth, which allows the user to draw waveforms, and then morph between an arbitrary succession of them. The output is calculated, and can then be saved to a WAV file. The waveforms are not specified by splines or another mathematical model; they are fixed curves. The drawback of SMoRPhi is that the morphing is simply equivalent to a crossfade, the curves being interpolated pointwise.

Yet this is not to disparage the potential of the synthesiser for creating some wonderful sounds!

Orangator, by Sharonov Oleg Yurievich, is an example of a software synth that can be played in real-time, to the limits of the processing capabilities of the machine. It is relevent to this paper because it uses a control point method to draw waveforms. Control points are connected by straight line segments. Using the real-time mode, the waveform can be altered and the sonic result heard after some latency.

There are sophisticated (non real-time) frequency domain approaches to sound morphing, for example, software realisations like Ircam's Diphone or Raffaele de Tintis's Morph (1998). Yet time domain morphing inevitably brings in the spectre of the crossfade. Pointwise interpolation of mathematical functions representing time domain audio is equivalent to an interpolated sum of the two functions; that is, a crossfade. This paper has the further theoretical aim to fill the perceived gap of a time domain morphing method.

Model

Preliminaries

Real-time synthesis methods deal with some form of manageable control data as a handle to the job of producing a continuous output stream of digital audio in the time domain. Changes in the control data will have an immediate effect on the synthesis, though the relation between audio result and the parameter changed can be obscure. Interfacing to low level digital audio requires a manageable representation of control data to specify the shape of a waveform. The model solves this by the use of interpolating splines defined by an ordered list of control points.

Understanding the model requires an understanding of interpolating splines. A basic introduction, alongside computational implementations, can be found in Sedgewick (1992), Flannery (1993) et al. To summarise, an interpolating spline is a piecewise polynomial curve that passes through every control point (Figure 1).

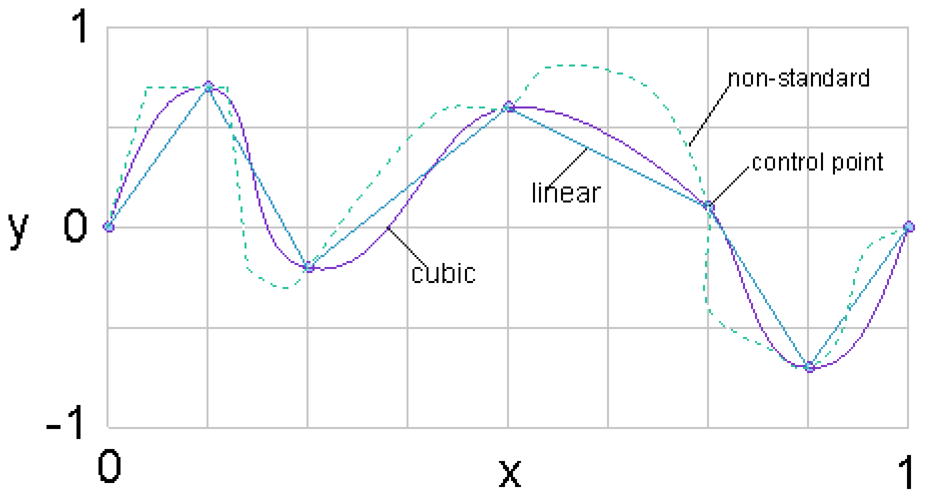

Figure 1 Interpolating splines

Usually, the pieces of the curve are defined between successive control points. The polynomials used could be linear, quadratic, quartic etc. We might require certain mathematical conditions upon the curve at the joins between polynomial segments. These conditions are of the form of continuity assertions. The conditions on the spline allow the formulation of a system of linear equations. This system could have multiple solutions, but we shall be dealing practically with cases where the solution is unique.

The spline that is used in practical matters throughout this paper is a standard cubic interpolating spline. Each of the piecewise polynomials is cubic; there is a cubic curve between any two successive control points. The continuity conditions are of C2 continuity (being twice differentiable) at the joins, and of zero second derivative at the endpoints. Under such restrictions, given a set of control points, there is a unique natural cubic interpolating spline through those points.

It should be noted that linear splines, a sequence of line segments between control points, are also suitable for the model of this paper. Linear splines are already used in software to model waveforms, because of the simplicity of connecting a sequence of points by line segments.

A basic knowledge of splines is henceforth assumed.

Splines to represent waveform shapes

Take a continuous function curve y=f(x) in the two dimensional space [0,1] x [-1,1] (Figure 2).

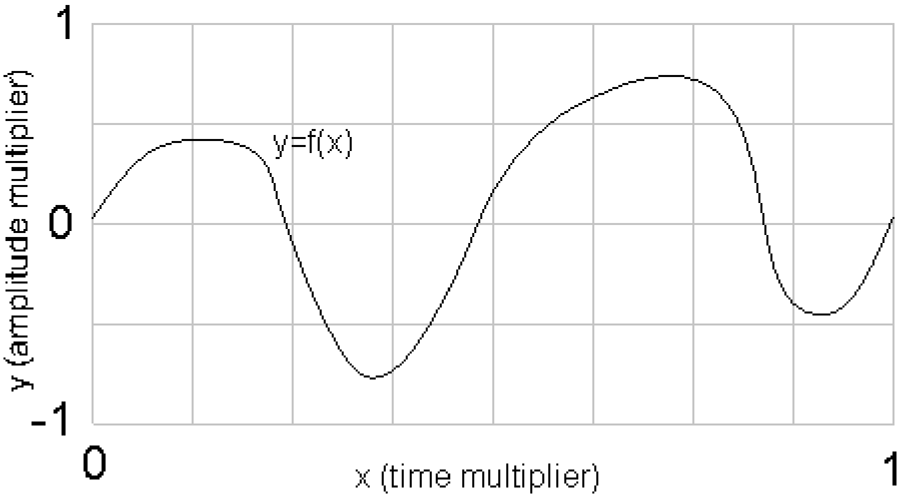

Figure 2 y=f(x)

This curve cannot double back on itself (else it would contradict the fact it is a function). It is safe therefore to sample the curve, with a view to using its shape as an audio waveform packet of limited duration. The x axis corresponds to time; the y to amplitude. The use of the particular co-ordinate space here allows ease of calculation, given any time interval over which to sample a packet of audio, or any maximal amplitude. Any number of evenly spaced samples N can be taken over the x range, at intervals of 1.0/(N-1). We can take a loop of the audio packet produced from the curve. If the curve does not satisfy f(0)=0 and f(1)=0, there will be a discontinuity at every loop.

Now we model a curve y=f(x) using an interpolating spline through a number of user specified control points (Figure 3). The control points are a list of points in the space [0,1] x [-1,1] ordered by increasing x co-ordinate. It is further required that all x co-ordinates are distinct. There are at least two control points, always preserving a

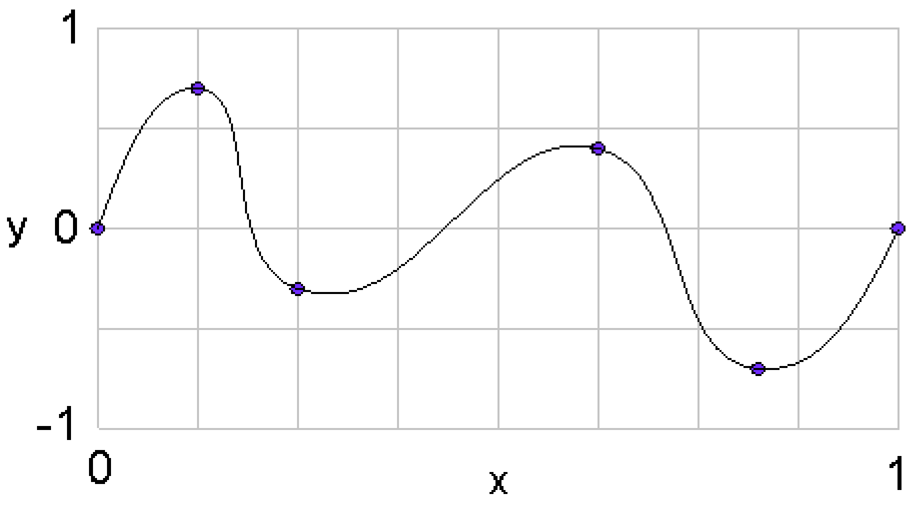

Figure 3 Spline through control points

minimum of two at the co-ordinates (0,0) and (1,0) (Figure 4). These two permanent control points facilitate looping of waveforms without discontinuity. Users have full control over all other control points, repositioning, adding or deleting them as they see fit.

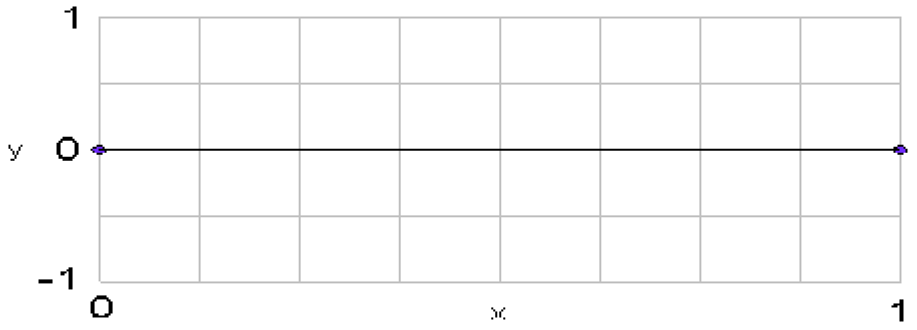

Figure 4 Minimal spline

Even though the y co-ordinates of the control points are within the range [-1,1] it is possible for the spline curve solution for those points to move outside that range. Y values can be clamped to the range, or allowed to spill over. In terms of the audio that will be read from the spline waveform representation, spill over will cause distortion. Under these conditions it is guaranteed by the properties of the interpolating spline that the curve provided will be suitable for continuous audio output (Figure 5) .

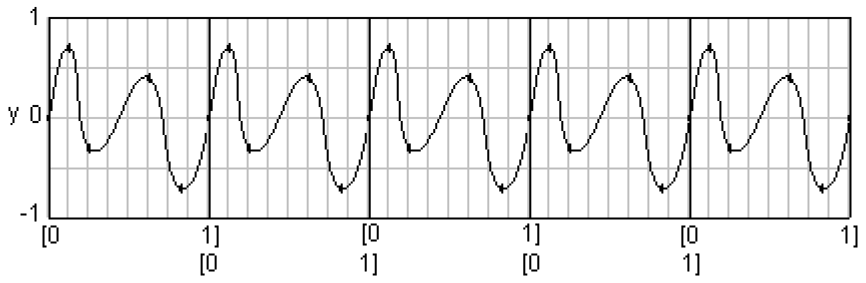

Figure 5 Looping for continuous audio output

The model allows direct manipulation of audio via manipulation of a spline curve’s control points. Any change in control points requires a recalculation of the spline’s shape, which in turn, alters the shape of the output waveform.

The output waveform packet must be taken from a fixed reading of the spline. If any control point was changed during the course of the audio data being read, a discontinuity could creep in into the output sound. This condition gives a minimum latency to the audio output of the number of samples taken across a reading of a spline. Since the number of samples per waveform is likely to correspond to a very short duration, the real-time response can give the illusion of immediacy.

Morphing

Suppose that we have two splines, representing two audio waveforms. Is it possible to turn one spline into the other in a controlled and continuous manner? The morphing of splines is achieved by some form of spline interpolation (Figure 6).

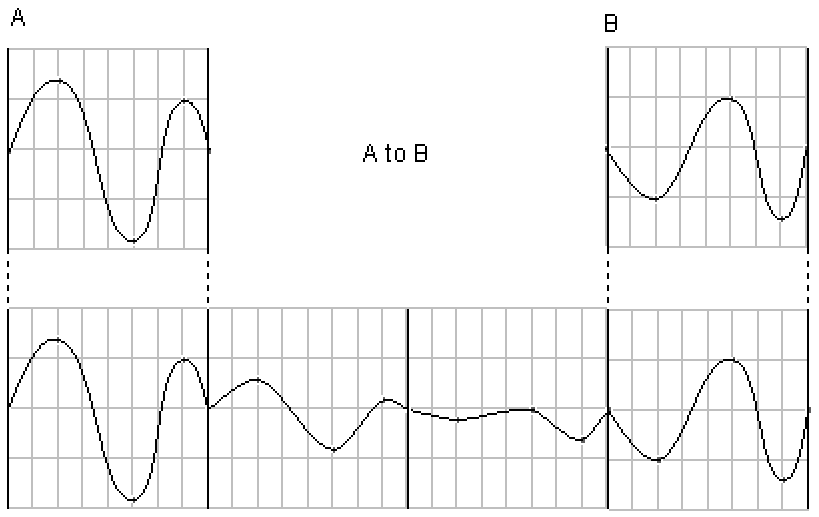

Figure 6 Spline interpolation

The appendix deals with the technicalities at depth, and is the main theoretical part of this paper. In terms of providing this morphing ability to the user of a program, the user can specify a sequence of spline keyframes. The audio output is a constant loop through the keyframes, appropriately creating intermediate splines to effect a continuous morphing transformation. At the end of keyframe sequence, the last keyframe loops back to the beginning. The user can establish interpolation properties (like interpolation length) between any keyframes.

Audio from splines

Audio data taken by these methods might be used as carrier or modulator in AM/FM synthesis, or heard directly as a pure sound. Keyframe splines can have associated synthesis parameters. Interpolation between keyframes then changes the spline, and concurrently interpolates audio parameters.

The user can specify how many samples to take across a spline. This will of course have a bearing on the apparent pitch of the audio output for a carrier, or the rate of effect for a modulator. It is likely that there will be more samples taken over a spline than control points; else the detail of the spline is obfuscated and lost.

Realisation

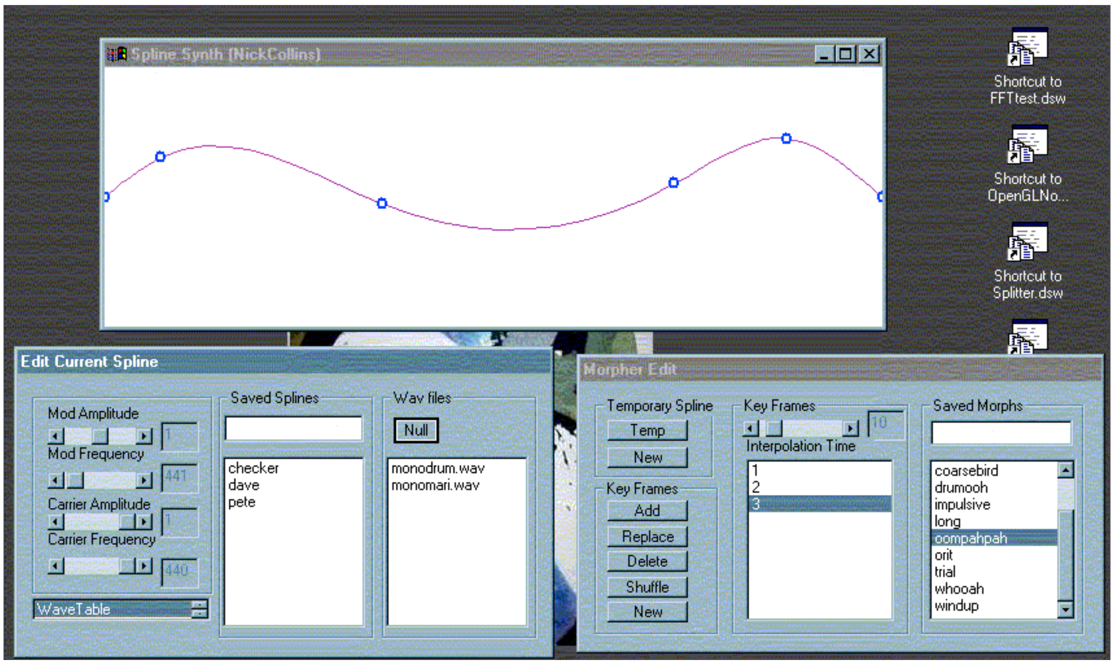

Figure 7 SplineSynth

A software synthesiser dubbed 'SplineSynth' was constructed to test the model. It is a Win32 application, built using Visual C++. This section describes some implementation details that are platform specific, without delving very deeply into the code.
A Win32 application was constructed from scratch, without MFC, utilising Win32 class building hints and example code from Reliable Software (1996/7). The application also makes use of some WAV reading code from Don Cross (1997). Otherwise, the code is the author's own work, intended as freeware. The software is primarily a feasibility study for the conceptual design of the interface, and makes no claims to be of commercial release quality. This would be unrealistic considering the programming team of one!

The application is multithreaded, based around an audio thread, and a main user interface thread. The audio thread is charged with maintaining an unbroken output of audio data to the sound card, using multiple buffers. The user interface is concerned with the user being able to alter the shapes of splines, choose the sonic ramifications of a spline (representing a packet of audio data for AM, FM, or as pure source), and to set up lists of splines for contiguous sound transformations ('morphs').

Figure 7 is a screenshot of the complete application at the time of writing. The main window is for directly editing a spline. The bottom left window concerns audio interpretations of that spline, and saving and loading individual splines. The bottom right window concerns establishing the 'morphs' or lists of splines to be transformed through. The morphs can also be saved or opened, like synthesiser presets. From a morph, the user can edit any spline of the sequence, or choose to edit an entirely new spline, not yet a part of any morph.

It is the morph that is heard, for the audio thread takes data from the current morph, looping through the spline keyframes, interpolating between them. A system of caching avoids the recalculation of interpolation when no changes have been made. Alteration of any spline in the morph will immediately invalidate the local caching, allowing a real-time effect on the audio output. Changes in the audio parameters of a spline also invalidate caching, for these variables are interpolated alongside spline shapes.

The model works very well in practice. Users can feel that they are having a critical effect on the underlying waveforms. It is fascinating to hear subtle shifts in the harmonics of a waveform realised by the slow dragging of a control point. This realisation does have a certain latency, due to the use of multiple buffers. If many samples are taken across a spline, then the size of data sent in each buffer can become very big, exacerbating the problem. In the main, changes propagate fast enough to make an aural impression. If a long morph is being looped through, changing the spline will only affect the audio output immediately if the morph is currently near the keyframe spline in question.

Further work on the model

Alternative interpolation methods for the splines could critically alter the audio output of a morph. A comparative study should be undertaken. The point of the interpolation schemas is to escape from the audio cross-fade to other aesthetically pleasing transformations, to provide alternatives.

An obvious further application of splines is in real-time filter manipulation. The graph shifts from time domain to frequency domain, but it is a non-trivial task to determine a digital filter from a given frequency response.

Further work on the application

Visually depicting the change of splines (and hence of audio packets) over is a priority. This could be done by drawing in real-time a miniature depiction of the current spline generating audio, or a window could exhibit the whole shape of the transformation, with key frame and interpolating splines marked accordingly.
It would even be possible to show in three dimensions the transformation of successive splines as a surface. Every keyframe spline would be on a parallel edge. the use of a surface can become a new interface itself. The surface could be deformed in real-time to change the morph. Such deformation outgrows the need for keyframes and spline interpolation altogether. We have reached two dimensional function synthesis (see the background).

The application contains no support for saving audio output as a soundfile, or determining base frequency from MIDI input. These features would be straightforward to implement compared to interpolating surfaces.

It would be gratifying to allow the user tools for manipulating multiple control points simultaneously. For example, the user could pick up and drag multiple control points at once, rather than one at a time. This is in fact a vital facility for more complex splines, where there are many control points and the manipulation of a single one is so constrained as to have little effect on the sound. Groups of control points might also be copied from place to place, or repositioned by affine transformations.

It would be healthy to allow multiple oscillators, that is, multiple concurrent morphs. Lastly, splines could determine the waveforms for both carriers and oscillators in synthesis methods. The splines should be a general resource for waveform shapes.

Appendix- Interpolation Methods for Splines

Note: it is assumed that the splines have the property of uniqueness; given a set of control points there is a unique spline solution. Otherwise, the material below might possibly be adapted, given the ability to make a sensible choice of spline for a given set of control points.

Given interpolating splines A and B with control points in the region [0,1] x [-1,1] we need to find a finite sequence of splines that will interpolate between them, such that A is the first element of this sequence, and B is the last. The interpolation length is the number of splines in this sequence, and will be at least 2 if A is not equal to B. It is not enough to just find any sequence; we shall usually require that each successive member of the sequence is within some distance of its predecessor, according to some metric.

The metric might be on the space of continuous maps from [0,1] to the real numbers, that is, a measure on the spline curve itself. However, this route can be impractical where control points are very close together in x, due to blow up in the shape of the spline curve. A solution to such blow up is to consider clamping to the y range [-1,1].

A metric could also be on the control points, as in the examples given here. This method requires that A and B (and all interpolating splines) have the same number of control points, and further, that control points are paired off between A and B. These corresponding control points will be interpolated along paths in the two dimensional space [0,1] x [-1,1] (Figure 8).

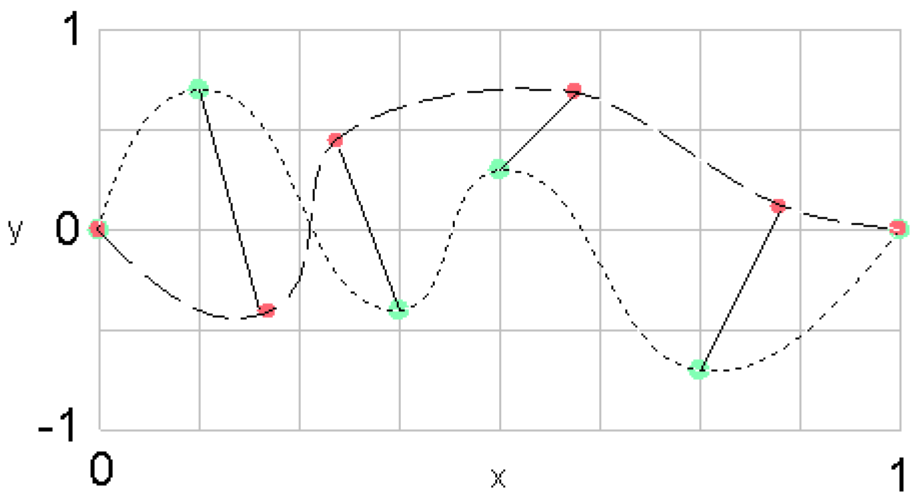

Figure 8 Interpolating control points

To reiterate, at any point of the interpolation from A to B, a set of control points is determined. These are sufficient to generate a unique spline as the intermediate spline at that stage of interpolation. The problem of interpolating the splines has become that of interpolating the spline control points, a reduction of the problem.

In order to make sure that intermediate splines are suitable for audio, in pairing control points, it is required that the first and last control points of any two splines be matched. Since A and B both have their first control point at (0,0), their last at (1,0), this will force a control point at (0,0) and one at (1,0) for every intermediate independent of whichever interpolation scheme is used. The schemes below will deal with preserving this state automatically.

Why could one interpolate control points and not worry so much about the shape of the spline curve? The answer to this is that our purpose is to find a way of interpolating sound that does not cause a normal crossfade. The first scheme, of a metric on a function space, is closer to a crossfade in that it is interpolating at every value of x along [0,1]. The second scheme interpolates only at some finite list of distinct x .In the audio output, the successive interpolating splines will follow on from one another in their sequential order. We will hear some sound transformation beginning and ending where we expect, but going via some intermediate sound that must bear some resemblance to A and B, but does not necessarily have to be an audio mix of A and B. Transformation of control points will preserve notions of interpolating shape and form in the intermediate spline curve, but will allow enough freedom (on the curves between control points) to give non-linear results.
The interpolation methods on splines have led us to interpolation of control points along some path in [0,1] x [-1,1] between positions c and d. The interpolation along a path can follow any continuous mapping m from [0,1] onto that path that satisfies m[0]= c, m[1]= d. Most typically, the mapping will be standard linear interpolation, and unless otherwise stated, this is assumed below. It must be noted that given a set of paired control points, the interpolation mapping along each path could be independent. Such variations in mappings will certainly lead to audio results quite distinct from crossfades.

The whole area of spline interpolation is fecund for further research, including both mathematical and psycho-acoustical study. Two schemas for control point interpolation of splines are presented. The first is flawed, and equivalent to a crossfade. It is presented as a first stage warm up, and because it becomes non-trivial if a variety of interpolations along control point paths is used. The second is more interesting, and is that implemented in the software. Two variants of that schema show the possibilities therein.

First we shall require a few preliminaries to assist the explanation of the schemas:

Lemma. Let S be an interpolating spline determined uniquely by an arbitrary set of control points in [0,1] x [-1,1]. Let x' be an x co-ordinate at which S has no control point. Add (x', S(x')) to the set of control points of S. Then S is determined by this enhanced set of control points.

Proof. This result is immediate from the uniqueness of S. If a spline T solves for the enhanced control point set, then it also forms a solution for the original control point set; it must interpolate that set as a lesser requirement of its existence. But by uniqueness of S, T=S. QED

The lemma shall be used to add control points to A and B such that they attain the same number of control points but retain their original shape. If enhanced by the addition of such control points, a spline S shall still be referred to as S. This is technically inaccurate, but avoids unnecessary complications in the notation.

It is also helpful to spell out the following simple result: If a spline S is evaluated at any x belonging to a control point, S(x) is equal to the y value of that control point. This is immediate from the interpolating property of the spline.

Now we proceed with the schemas-

Flawed Schema – Interpolation of y at control points fixed in x

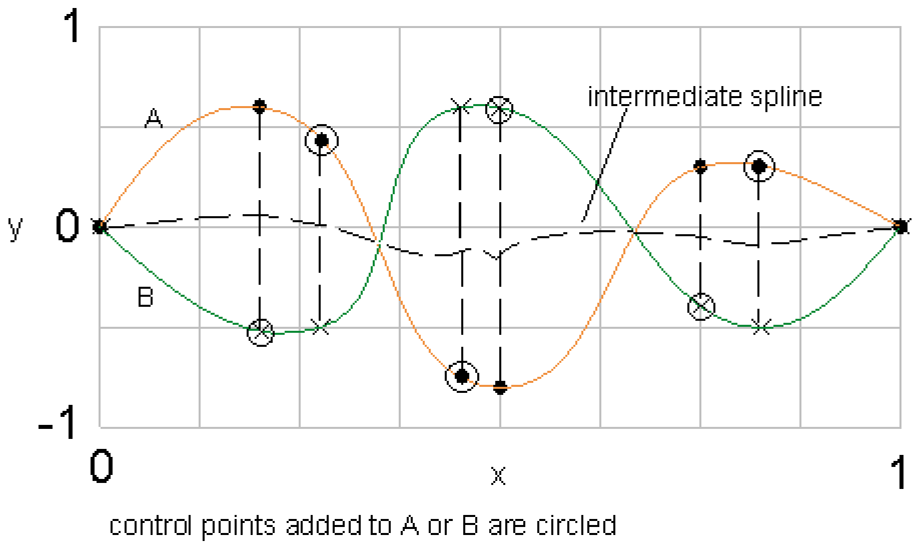

Figure 9 Schema 1

Let the list of the x positions of A's control points be $L_A = \{x_i: i = 0,\dots, M-1\}$. It is assumed that these the x values are increasing, as the list of control points of A is so ordered. Similarly, for B, take $L_B = \{x_i: i = 0,\dots, N-1\}$. Let $L = L_A \cup L_B = \{x_i: i = 0,\dots, L-1\}$, ordered of course with respect to increasing x. Then the cardinality of L is at most M+N, but may be less, depending on repetitions of x values of control points in L_A or L_B.

By the lemma, the list of A's control points can be taken as $\{(x,A(x)): x\ e\ L\}$ without contradicting the shape of A. Similarly, the list of B's control points can be taken as $\{(x,B(x)): x \varepsilon L\}$. Control points of A and B at the same x value are paired. Then the interpolation paths are vertical. (Figure 9).

This schema is flawed. It is equivalent to a crossfade because of the uniqueness of splines. Take a pointwise summation of two splines A, B with a common set of x values of their control points. We can write $C= (1-t)*A + t*B$ where t is a real number. Interpolate at the common x control point pairs between A and B. The resultant control points for interpolation coefficient t are the points (x,C(x)). Since C interpolates those points, and C is a spline, C is the intermediate spline according to this schema, but C has been shown to be the result of an audio crossfade. Therefore, the schema can only supply an audio crossfade.

Schema two- Interpolation of control points in x and y

The diagrams should give a good understanding of the nature of the algorithms.

Suppose spline A has M control points, and B has N.

Without loss of generality, suppose M<=N. Otherwise, the following methods are calculated from B to A, the resultant sequence then reversed to give an interpolation from A to B.

Let the list of A's control points be CPA= {CP(A,i); i = 0 to M-1} where CP(A,i) is the ith control point. Let those of B be CPB= {CP(B,i); i = 0 to N-1}. These lists are ordered by increasing x, as per usual.

The intuitive idea here is to add control points to A until pairings are established between A and B. In some cases, it may be necessary to add control points to B, due to the matching algorithm. Algorithms to add control points iteratively extend A and B and hence CPA and CPB until a 1-1 onto mapping from CPA to CPB is found. The establishing of a mapping m: CPA-> CPB which is 1-1 and onto is the general aim of all control point spline interpolation schemas, and the first flawed schema could be subsumed into this one on that understanding.

Two algorithms are presented here. Neither allows the crossing of control point interpolation paths. More general algorithms might establish such pairings, particularly to cause stranger waveform distorting effects.

Algorithm 1

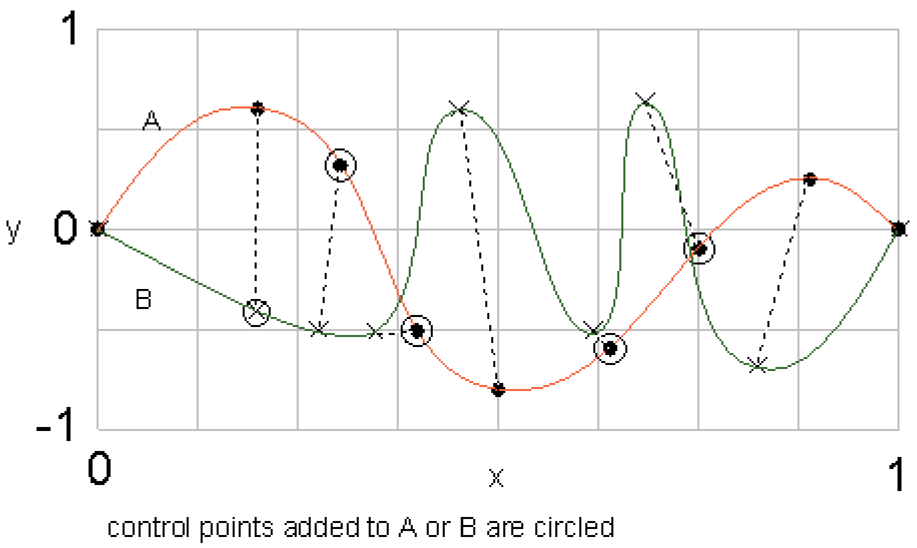

Figure 10 Schema 2 Algorithm 1

Take the first unmatched control point CP(A,n) of A. The x value at this control point is x_n. Search L_B for the greatest x less than or equal to x_n such that the control point CP(B,m) corresponding to x has not been paired so far, and no control point after CP(B,m) has been paired off. If there is no such x, add a control point to B at (x_n, B(x_n)). Pair CP(A,n) and Cp(B,m), or CP(A,n) and the new control point of B.
If this process if continued for all control points of A, there may still be control points left over in B unmatched. A map m from A's control points to B's has been established which is 1-1 but not necessarily onto. However, the pairings so far have laid out convex hulls within which additional control points in A can quickly be set up against those remaining in B. Given any two consecutive control points (CP(A,n), CP(A,n+1) of A, add as many control points to A evenly spaced in x between those two as are required to match left over B control points between m(CP(A,n)) and m(CP(A,n+1)). Pair them by increasing x.
Figure 10 demonstrates this algorithm in a specific case. The diagram should aid comprehension of the method of the algorithm.
The algo rithm guarantees the pairing of (0,0) and (0,0), and (1,0) and (1,0).

Algorithm 2

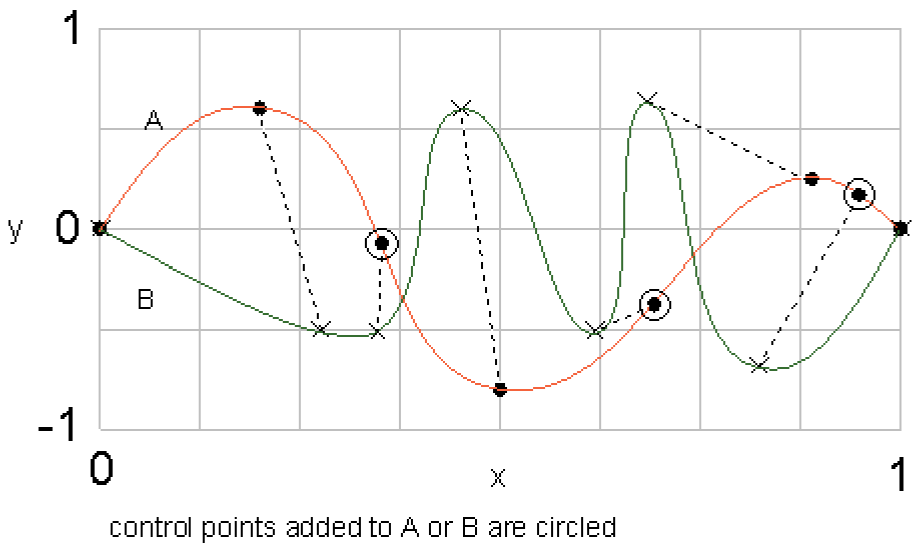

Figure 11 Schema 2 Algorithm 2

This method requires no extension to B.
Match the ith control point of CPA to the jth of CPB where

j= i * (N-1)/(M-1) rounded down (remember indexing is from 0).

this indexing guarantees the pairing of 0 and 0 and M-1 and N-1, that is, control points (0,0) and (0,0), and (1,0) and (1,0).

Convex hulls have been established, just like algorithm 1 above. Follow the second stage of that algorithm to add extra control points to A as necessary to pair 1-1 onto against those of B.

Figure 11 gives a visual aid.

References

Borgonovo, A and Haus, G "Sound Synthesis by Means of Two-Variable Functions: Experimental Criteria and Results" Computer Music Journal 10 (4): 57-71 1986

Collins, Nick http://www.axp.mdx.ac.uk/~nicholas15/

Cross, Don http://www.intersrv.com/~dcross/audio.html

Flannery, Brian P ; Press, William H; Teukolsky, Saul A; Vetterling, William T *Numerical Recipes; The Art of Scientific Computing* Cambridge University Press 1993

Karwath, Andre http://www.tu-chemnitz.de/~aka 1997

Miranda, Eduardo Reck *Computer Sound Synthesis for the Electronic Musician* Focal Press 1998

Mitsuhashi, Yasuhiro “Musical Sound Synthesis by Forward Differences” J Audio Eng Soc Vol 30 No 1/2 1982a

Mitsuhashi, Yasuhiro “Piecewise Interpolation Techniques for Audio Signal Synthesis” J Audio Eng Soc Vol 30 No 4 1982b

Mitsuhashi, Yasuhiro “Audio Signal Synthesis by Functions of Two Variables” J Audio Eng Soc Vol 30 No 10 1982c

Moore, F. Richard *Elements of Computer Music* PTR Prentice Hall 1990

Reliable Software, http://www.relisoft.com/win32/, example code 1996/7

Roads, Curtis *The Computer Music Tutorial* MIT Press 1995

Sedgewick, Robert *Algorithms in C++* Addison Welsey 1992

Softsynths, http://www.maz-sound.de/

Tintis, Raffaele de, “Morph: Timbre Hybridization Tools Based on Frequency Domain Processing” http://www.iua.upf.es/dafx98/papers/ 1998