Interpolating Break-Point Sets To Obtain Sound Transformations Distinct From a Cross-Fade

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

Introduction

The genesis of this article can be traced to the following question: given two short time-domain segments of audio, are there time-domain methods for transforming one into the other distinct from a cross-fade? Methods of morphing in the frequency domain have been explored (for instance, see Depalle et al 1993, Slaney et al 1996). Yet, the author has found a rewarding set of transformations by considering the subset of time amplitude curves modeled via break-point sets. This is the domain of break-point interpolation synthesis (Bernstein and Cooper 1976, Mitsuhashi 1982a). Algorithms for the interpolation of break-point sets are investigated herein as pertaining to sound transformation.

Since break-point sets may be useful wherever two or more parameters are mapped against each other, the transformations can have applicability to situations other than time amplitude curves. For instance, a function constructed from a break-point set could determine a center frequency versus amplitude map for a filterbank at one instant. As break-point sets are interpolated over time, the filter will change dynamically. Alternatively, break-point sets could determine surfaces, using NURBS or otherwise. These surfaces are transformed over time, with a one dimensional scan determining digital audio output, that is, dynamic wave terrain synthesis (Mitsuhashi 1982b, Borgonovo and Haus 1986). In the work here, we restrict ourselves to the case of the time amplitude curves of traditional break-point interpolation synthesis which will be sampled for digital audio.

The key-frame approach is a widely used paradigm originating in hand animation, especially familiar in computer graphics animation. Break-point set interpolation algorithms allow the interpolation of user defined key-frames. The

user becomes the chief animator, and the computer automates the drudgery of inbetweening. We gain an automatic synthesis technique that allows immediate control of low-level wave shapes, whilst maintaining continuous transformation within sound, a quality well known as necessary for dynamic audio. As we shall see, most interpolation algorithms one could construct are distinct from the standard all-pervasive cross-fade.

Break-point Set Interpolation Algorithms

The time-amplitude break-point sets are defined within the space [0, 1] by [-1, 1], so are immediately scalable as floating point audio. We allow free values of abscissa and ordinate, though there must be only one break-point for a given value of the abscissa. Each break-point set must include at least two break-points at (0, 0) and (1, 0). This maintains continuity between successive concatenated break-point sets, and in audio terms, avoids clicks every period. Classic break-point interpolation synthesis considers an interpolation function between successive break-points. We differ here in further utilizing linear and cubic splines. The linear spline is equivalent to the linear interpolation function, but it shall be convenient to talk of splines including this case. The technical reason for the emphasis on splines is that we may add extra break-points to a spline without changing the interpolation curve as long as those break-points are placed exactly on the existing curve. This follows from the uniqueness of the spline demonstrated in (Collins 99). This property is vital in that we will equalize the number of break-points between two successive break-point set key-frames in order to interpolate intermediate break-point sets, whilst preserving the key-frames themselves as part of the transformation.

One side effect of using splines is that we must occasionally clip ordinates on the interpolating curve to the range [-1,1], whereas interpolating functions always reside within the rectangle laid by the break-points interpolated.

Given the above conditions, transformations equalize the number of break-points and pair up break-points between two key-frames, then linearly transform one break-point into another. In general, an arbitrary interpolation function could determine the speed of transformation independently for each pair, and the least distance path, that is, the linear path, could be replaced by any circuitous route. There are an infinity of plausible transformations, differing in the 'wildness' of the intermediate interpolants with respect to their begetting key-frames.

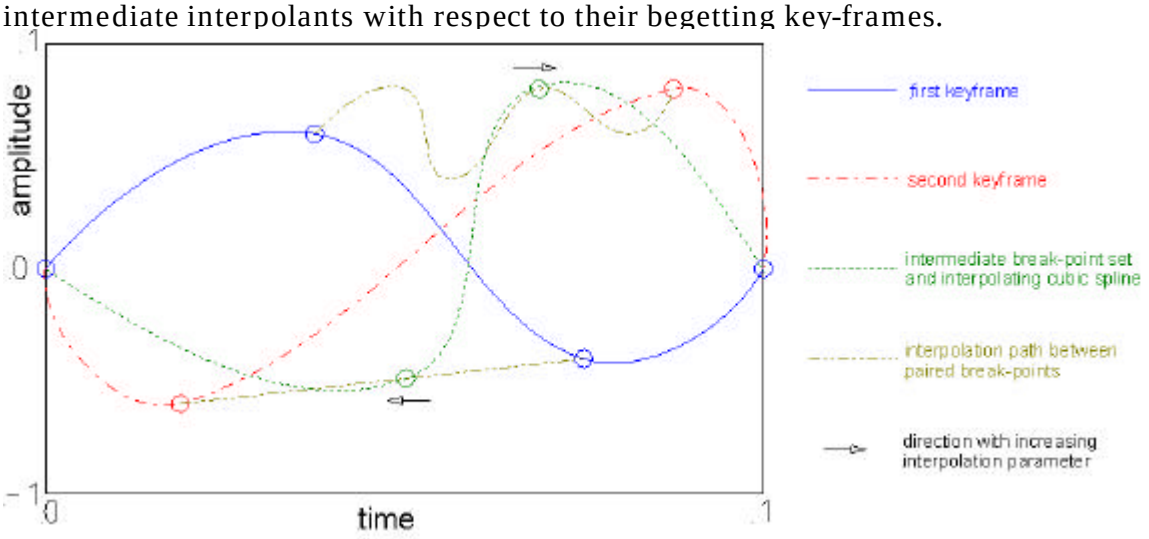

Figure 1 An Example Transformation

Figure 1 shows an example transformation. Note that even with the classic interpolation functions, a non linear interpolation path between break-points may require clipping of the ordinate values of intermediate break-points within [-1,1], or if break-points outside the basic square are accepted, clipping of the interpolating curve.

Six Algorithms

Six algorithms are studied herein. The first three are discussed in more detail with an alternative method in (Collins 99). These are general algorithms which for any given break-point set key-frames will supply intermediate break-point sets bearing

a healthy relation to the begetters. The key-frames themselves will be generated at t=0 and t=1, where t is the interpolation parameter analogous to time taking values from [0, 1]. The reader should be careful not to confuse interpolation time t used for transformation between break-point sets with the x axis in a single time-amplitude break-point set.

We must set up a mathematical description sufficient to portray the six algorithms.

Given two key-frame break-point sets A and B, we will transform A into B. Without loss of generality, A has a less than or equal number of break-points than B, for otherwise transform from B to A then reverse the time order of the transform. We establish new break-point sets A' and B' that will end up with an equal number of break-points, and some match function, a finite 1-1 onto map pairing indices of break-points from A' to B'. A' and B' begin as A and B, and are gradually expanded within a given algorithm by the addition of break-points. The deletion of break-points is never allowed, as the shape of the original curve would be lost. For splines, A' and B' must admit the same interpolating curve as A and B respectively. So any break-point added to A' or B', whatever the abscissa, has ordinate equal to the evaluation of the interpolating spline of A or B at that given abscissa. This guarantees the property desired (Collins 99). Because the ordinates are dependent on the abscissae in this sense, we need consider only the list of abscissae of A and B, XA and XB, ordered by increasing x. We denote the expanded lists for A' and B', XA' and XB' , which begin as equal to XA and XB . It is hoped that the reader will forgive the ambiguities in the non-rigorous presentation of this material. We will not consider named intermediate sets for every stage of the algorithm, but update the dashed sets. We must separate A from A' so that the original can always be referenced. We will speak interchangeably of adding abscissae to XA' and break-

points to A'. We also write card(object) for the size of a finite object, hence card(A)= card(XA) is the initial number of break-points in A' or x values in XA'.

All algorithms will always match (0, 0) in A to (0,0) in B, and similarly (1,0) to (1,0). All break-point sets utilized for time amplitude curves have these break-points, so there is no difficulty here. It will be seen that the algorithms usually deal with these particular matches implicitly, or match them immediately and then consider all remaining break-points separately in the main part of the algorithm.

For the special case of monotonic match functions between general continuous curves, the reader might compare (Slaney et al 1996). In this paper we deal with a finite domain and range only, since our parameters are the finite break-point sets.

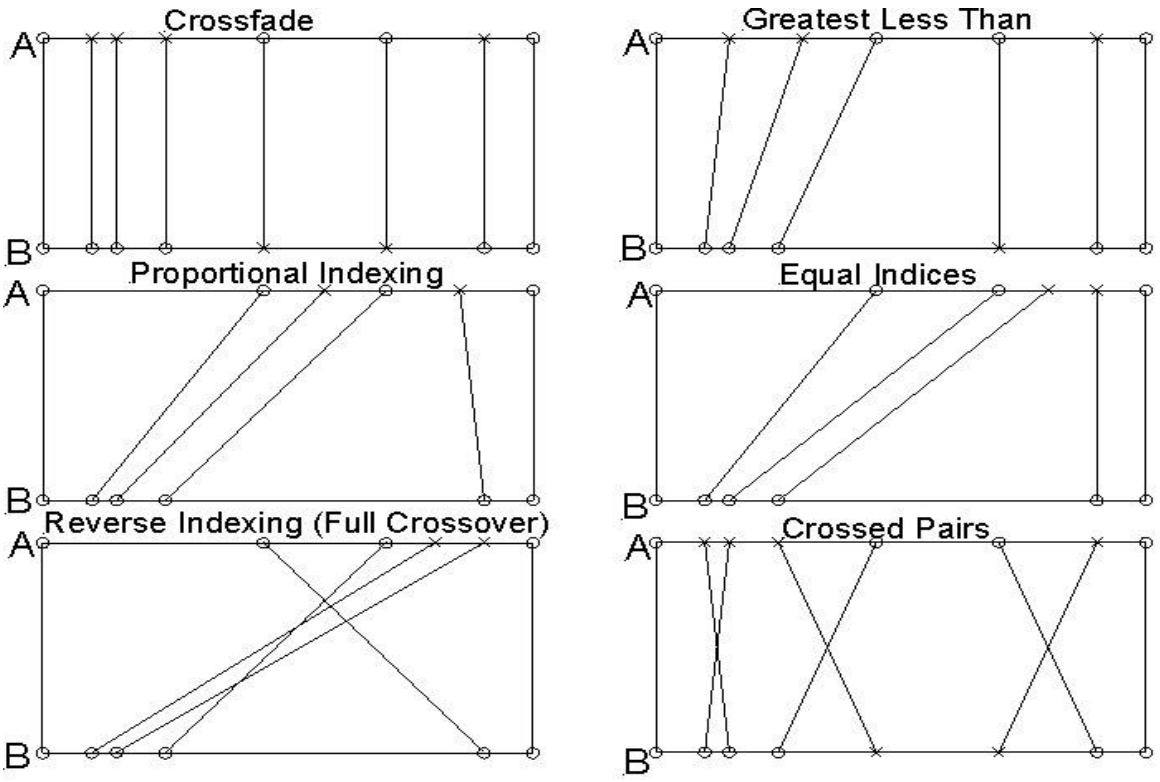

Figure 2 Six Transforms

Now examine Figure 2, which shows the six transforms under discussion, for a specific A and B (more properly, XA' and XB'). The circles denote the original break-

point abscissae, which are the same for all six diagrams. The crosses denote the positions of added break-points. The lines denote the match functions and the intermediate linear interpolation of the associated x values if one imagines interpolation time going linearly down the page from A to B. It should be noted that all six transforms are distinct in their pairings and hence the intermediate waveshapes for this relatively simple case.

The transforms have been given suggestive names to help in understanding them.

Cross-fade

Let the final AX' and BX' be the union of AX and BX. Pair matching indices (the identity permutation).

A little thought will show this transform is equivalent to a simple pointwise interpolation, and hence a cross-fade.

Greatest Less Than

Iterate through AX starting at 0. For given x

Match x to the greatest value x_B in BX less than or equal to x. If no such x_B exists, add a break-point to B' at x, and use this to pair.

For any abscissae missed out in BX less than or equal to x, pair it by add break-points to A' at equal spacings of the abscissae between x and the most recently matched abscissa in AX.

Proportional Indexing

Let PROP= card(B)/card(A). Match (1,0) in A to (1,0) in B. Consider the remaining points indexed from 0. Then match the i^{th} x value in AX to the j^{th}= (int) (PROP*i) in BX, where we are taking the integer part of the calculated real (floating point) value.

Convex hulls have been set up (examine figure 2) within which it is easy to add

break-points to A' matching those x values in B missed out by the proportional indexing scheme.

Equal Indices (with differing interpolation speeds)

Match (1,0) in A to (1,0) in B. Of the remaining break-points, match the i^{th} x value in AX to the i^{th} in BX. There are UNACC= card(B)-card(A) values in BX unaccounted for, positioned consecutively just before x=1 . Add UNACC break-points to A' spaced linearly in x between the second highest x of AX and 1, matching them by increasing x to those remaining unpaired in B. Further, let the interpolation time (not shown on diagram) between break-points depend on the index as follows: at interpolation time t for index i, t in [0, 1], use t'= pow(t, i) as the interpolation position along the i^{th} pair's line.

Reverse Indexing (Full Crossover)

Pair the x values at zero and one. For all remaining x in AX, pair them to the reverse indexed x in BX. For any remaining x in BX, pair them to additional x in A' placed linearly between the second highest x value and 1, matching x increasing in A' to x increasing in B. Of course, x increasing in A' to x decreasing in B would be an even greater twist.

Crossed Pairs

Use the cross-fade scheme to set up an equal number of break-points in A' and B'. Now cross in pairs as many internal (0<x<1) abscissae as possible.

TECHNICAL NOTE: It should be made explicit that the processes of equalizing the number of break-points, and of matching break-points between two key-frame break-point sets could be separated. If in the algorithms the addition of break-points

may seem to depend on the pairing of the initial break-points, it would still be possible to 'change the wiring' once the equalization of break-points has taken place. The algorithms are just some interesting demonstrations of what is possible, and could be quickly cross-bred with each other to produce countless variants. Use the cross-fade scheme then reverse index, etc.

Making your own Algorithms.

In general, the number of break-points in the two key-frames will not be equalized before transformation. In this case, break-points must be added or deleted at specific points in the transformation. This is analogous to the termination or introduction of partial tracks in sinusoidal modeling (for example, births and deaths of partial tracks in Depalle et al 1993). In the methods presented above, splines, and equality of the number of break-points were used so as to make sure that the key-frames themselves were part of the interpolation. Transformations can easily be designed which flaunt those conditions, whilst keeping some rough character of the defining curves.

Whilst interpolation functions between successive break-points like Mitsuhashi's half-cosine will not preserve the key-frames in the interpolation, in practice you would not notice, never hearing the original but just the interpolants. However, the changes you made to a break-point set key-frame may have a less honest effect on the interpolates.

Discretisation, Aliasing and Uncontrollable Harmonics

Because of the free abscissae, the Fourier analysis conducted by Mitsuhashi for linearly spaced abscissae does not hold. In general, the calculation to predict the spectral components from given break-points interpolated by cubic spline is too difficult to carry out analytically. This does not stop us using the techniques herein

to make interesting and novel sounds, it just limits the applicability of the technique for analysis / resynthesis.

Continuous curves will be sampled at a scan rate corresponding to a desired fundamental frequency. Uncontrollable prominent higher harmonics may cause noticeable aliasing at high frequencies, though low pass filtering of the curve prior to sampling could be implemented to avoid such cases if desired.

One practical result of great import is that crossing break-points will lead to concentrations of energy into the higher harmonics (and hence aliasing) as the proximity of break-points at different ordinates closes. This may be a desired musical effect. We can always choose to control this by removing break-points from intermediate break-point sets if they are judged too close together. We already specified only one break-point per abscissa, and piecewise constant functions can still be sampled.

An Implementation in SplineSynth2

The algorithms have been implemented in SplineSynth2, the next generation of earlier experiments discussed in (Collins 1999). This software synthesizer allows a user to manipulate break-point sets and establish key-frame sequences, all in real-time, hearing results of changes immediately. They can severely deform key-frames by moving multiple break-points simultaneously, rather than the usual one break-point at a time restriction. The highlighted break-points in the stage 2 window in figure 3 can be moved simultaneously relative to the mouse cursor. MIDI control allows 8 note polyphony, and the latency can be set as small as the soundcard allows (below 10mS, for perceptually immediate response). The novel part of the program is in the algorithms that interpolate through a sequence of any number of break-point set key-frames, those algorithms introduced in this text and the earlier

paper. A demo of SplineSynth2 (Windows PC only) should be available at the time of writing from the author's academic web site.

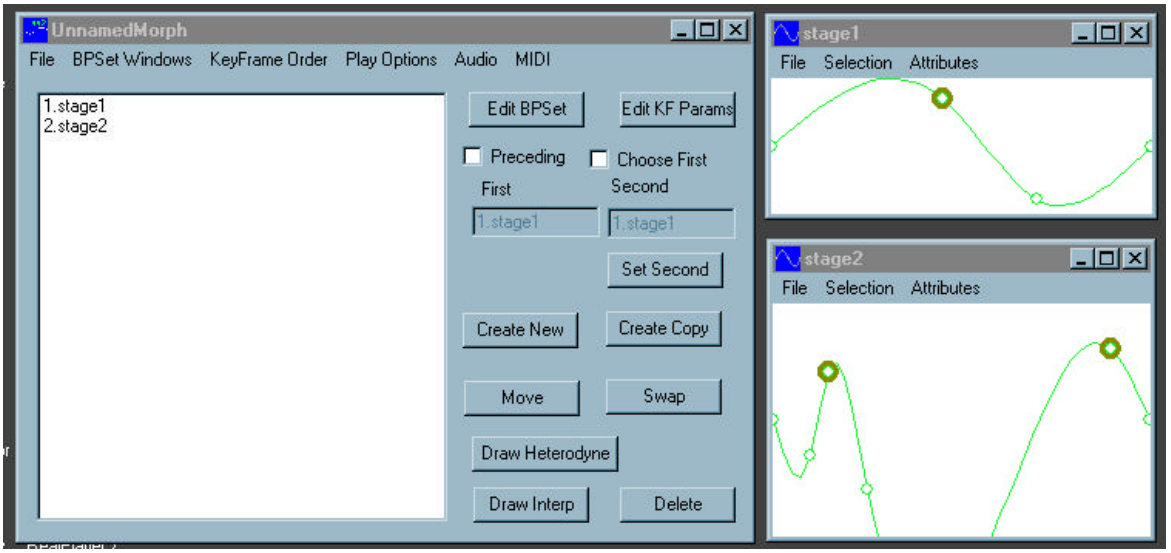

Figure 3 Screenshot of SplineSynth2

Comparison of the Six Transforms

As a demonstration of the six transforms, figure four shows some sample break-point transformations between two key-frames. The seven interpolated sets correspond to equally spaced interpolation times 0.0, 0.167, …, 0.833 and 1.0. The key-frames correspond to the far left and far right interpolated break-point sets and are shown in their original state in figure three above. The order of the six algorithms is as presented in this paper, from top to bottom, cross-fade, greatest less than, proportional indexing, equal indices (with differing interpolation rates), reverse indexing and crossed pairs.

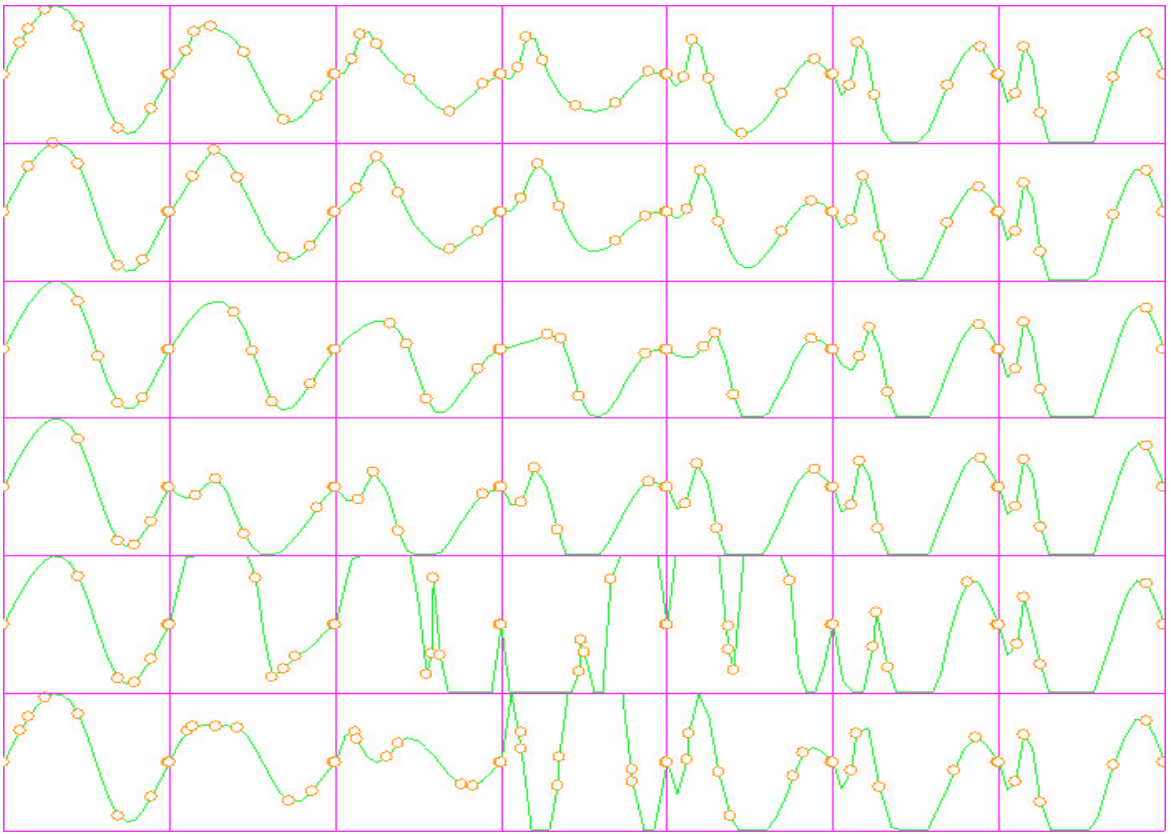

Figure 4 Comparison of intermediate break-point sets with cubic splines over six algorithms

Figure five shows the corresponding heterodyne analysis of the first ten harmonics of the output sound of a scanned a3 (220 Hz) over the six algorithms. The keyframes are as in figures 3 and 4, and the cubic spline provides the interpolating curve. The amplitude is mapped under log(log(amplitude)) to make more subtle differences noticeable. They are all distinct, proving that the transforms are all distinct from one another and the cross-fade. Note that algorithms that cross break-point interpolation paths (i.e with a non-monotonic match function) can cause a single wavetable of silence, if break-points pile up directly on top of one another and are not allowed through in that cycle. As might be expected the reverse indexing causes the most aural clutter.

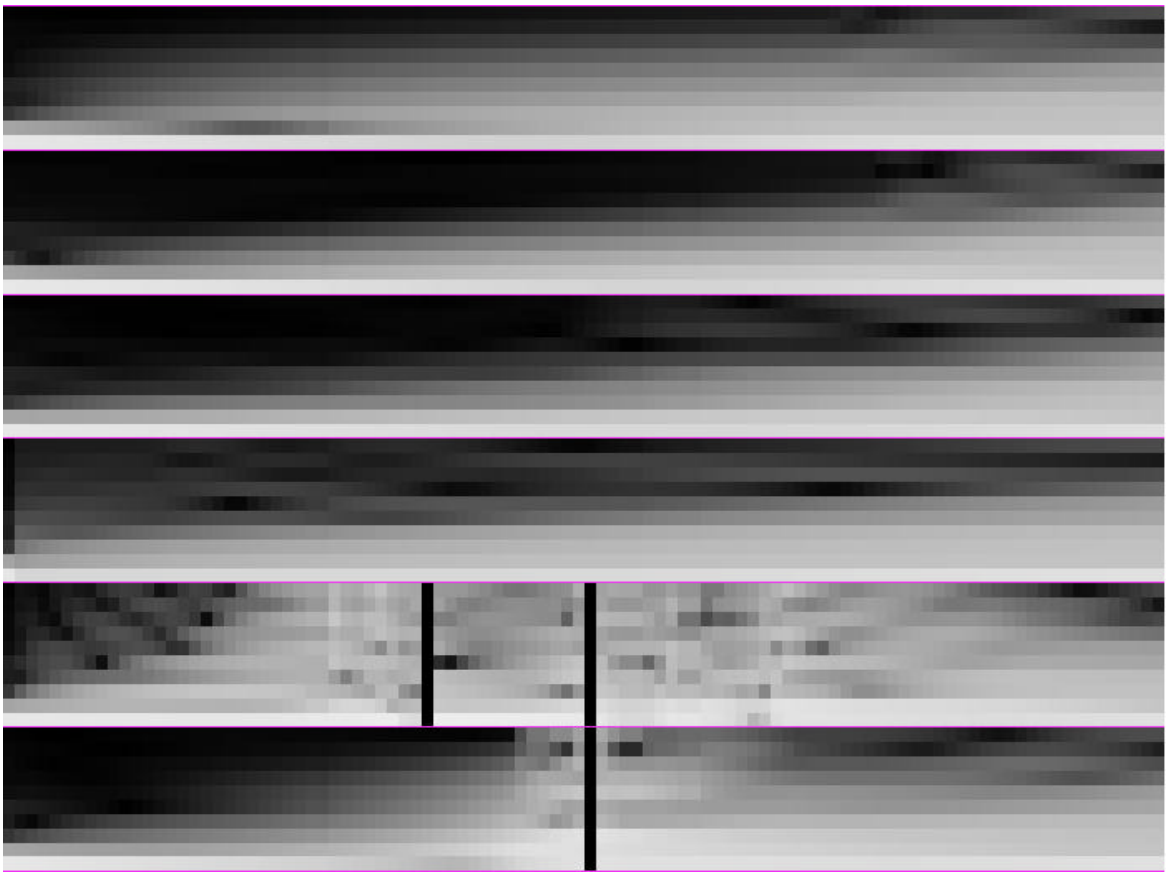

Figure 5 Comparison of the heterodyne analysis of one hundred interpolating break-point sets over the six algorithms, with log(log(amplitude)) as intensity and time versus frequency on the axes.

Further work and conclusions

SplineSynth version 1 already included a mode for the filterbank control, though further development would be desirable. The 'NURBSSynth' would be a much greater task. A 'Generalized Transformation' software package for defining one's own break-point set transformations would be extremely useful. A graphical interface would allow one to establish match functions, birth and death of control points during a transformation, interpolation rates and other facets of animation.

Such packages already exist to control animation sequences in the computer graphics world, but not to my knowledge, for audio purposes.

It would be gratifying to control grain shapes for granular synthesis over time from key-framed break-point interpolation synthesis. We may also control overlapped waveshapes in this context. To avoid the exact repetition of a sound with each trigger, a slight random perturbation of break-points for each new note might give a more dynamic system with a controlled but subtly varying sound output.

This article dealt with the introduction of transformation algorithms for break-point sets over time, with a view to making dynamic sounds, and timbre transformations distinct from the crossfading of waveshapes. I hope the reader is inspired to try out their own break-point set transformations to see what new sounds and transformations they can produce.

References

Bernstein, A., and Cooper, E. 1976. "The Piecewise Linear Technique of Electronic Music Synthesis" *Journal of the Audio Engineering Society*, Vol 24, July / Aug: 446-454.

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

Collins, Nicholas. "SplineSynth: An Interface to Low-Level Digital Audio." *Proceedings of the 1999 Diderot Forum on Mathematics and Music (Vienna)* ISBN 3-85403-133-5. pp 49-61.

Depalle, Ph., Garcia, G., and Rodet, X. "Analysis of Sound for Additive Synthesis: Tracking of Partials Using Hidden Markov Models." *Proceedings of the 1993*

International Computer Music Conference. San Francisco: International Computer Music Association.

Mitsuhashi, Yasuhiro. 1982a. “Piecewise Interpolation Techniques for Audio Signal Synthesis.” *Journal of the Audio Engineering Society*, 30(4).

Mitsuhashi, Yasuhiro. 1982b.“Audio Signal Synthesis by Functions of Two Variables”. *Journal of the Audio Engineering Society*, 30(10).

Slaney, Malcolm., Covell, Michele., and Lassiter, Bud. "Automatic Audio Morphing." *Proceedings of the 1996 International Conference on Acoustics, Speech, and Signal Processing*. Atlanta: IEEE.