OPEN-SOURCE MATLAB TOOLS FOR INTERPOLATION OF SDIF SINUSOIDAL SYNTHESIS PARAMETERS

Matthew Wright
U.C. Berkeley and Stanford
CNMAT and CCRMA
matt@{cnmat.berkeley,ccrma.stanford}.edu
http://ccrma.stanford.edu/~matt/additive

Julius O. Smith III
Stanford University
Center for Computer Research in Music and Acoustics (CCRMA)
jos@ccrma.stanford.edu

ABSTRACT

We have implemented an open-source additive synthesizer in the matlab language that reads sinusoidal models from SDIF files and synthesizes them with a variety of methods for interpolating frequency, amplitude, and phase between frames. Interpolation techniques currently implemented include linear frequency and amplitude ignoring all but initial phase, dB scale amplitude interpolation, stair-step, and cubic polynomial phase interpolation. A plug-in architecture separates the common handling of SDIF, births and deaths of partials, etc., from the specifics of each interpolation technique, making it easy to add more interpolation techniques as well as increasing code clarity and pedagogical value. We ran all synthesis interpolation techniques on a collection of 107 SDIF files and briefly discuss the perceptual differences among the techniques for various cases.

1. INTRODUCTION

Sinusoidal models are a favorite of the analysis/synthesis community [1-7], as Risset and Wessel describe in their review [8]. Sinusoidal analysis almost always happens on a per-frame basis, so that the continuous frequency, amplitude, and phase of each resulting sinusoidal track are synchronously sampled, usually at a regular interval such as every 5ms. An additive synthesizer must interpret these sampled parameter values to produce continuous amplitude and phase trajectories. Although the basic idea of synthesizing a sum of sinusoids each with given time-varying frequencies and amplitudes is clear enough, there are many subtle variants having to do with the handling of phase and with the interpolation of frequency, amplitude and phase between frame centers.

An additive synthesizer that simply keeps frequency and amplitude constant from one analysis frame until the next analysis frame generally introduces amplitude and frequency discontinuities at every analysis frame, resulting in undesirable synthesis artifacts. Therefore, all additive synthesizers use some form of interpolation.

Likewise, as phase is the integral of instantaneous frequency, all additive synthesizers have some kind of special joint handling of frequency and phase (unless, of course, they ignore the phases produced by the analysis.) Andersen and Jensen [9] review some phase representations for pseudo-harmonic sounds that allow modification of the fundamental's phase (e.g., by time-stretching) while preserving relative phase; they also propose an improved model along the same lines.

For the ICMC 2000 Analysis/Synthesis Comparison panel session [10], six participants each analyzed the same collection of 27 input sounds with their own analysis/synthesis systems. One point emphasized in the resulting discussion was that there is not a single universal "sinusoidal sound model," but rather that each analysis technique makes assumptions about how the synthesizer will interpolate parameters. Here are some examples:

- Linear interpolation of frequency and amplitude between frames, ignoring all but initial phase of each partial (SNDAN's pvan [11])
- Quadratic polynomial phase ([12])
- Cubic polynomial phase interpolation allowing phase and frequency continuity between frames ([13] , used by IRCAM [14, 15], SNDAN's mqan)
- 5^{th}-order polynomial phase interpolation allowing phase, frequency, and frequency slope continuity between frames ([16])
- Spline interpolation (PartialBench [17])
- Linear interpolation of amplitude on dB scale (SMS [18])
- Adjust each frequency trajectory to meet the partial's stated instantaneous phase by the time of the next frame, e.g., by inserting a new breakpoint halfway between two frames and using linear interpolation (IRCAM)
- Match phase only at transient times (Levine [19], Loris [20])

So questions arise: Theoretically, should all of these "sinusoidal" models be considered different sound models or variants of the same model? Practically, how much quality is lost when a sound model is synthesized with a different interpolation method from the one assumed by the analysis? How meaningful is it to interchange sound descriptions between differing assumptions about how synthesizers will interpolate? How do we switch gracefully between a sinusoidal model that preserves phase and one that doesn't, e.g., for preserving phase only at signal transients? Girin et al. [16] compare three frequency/phase interpolation models on a small collection of their own analysis results and synthetic examples, concluding that higher-order models can offer a performance gain[1].

[1] Their metric is signal-to-noise ratio (SNR); in this case, the "signal" is the original input sound and the "noise" is the residual upon subtracting the resynthesis from the original. They comment that this is "an imperfect perceptual metric" and remark that all synthetic examples are perceptually close to the originals in spite of large SNR differences.

To address these practical questions in an open-ended way, we have implemented an open-source framework for sinusoidal synthesis of SDIF ("Sound Description Interchange Format") [21] files. This allows anyone to synthesize according to any sinusoidal model in an SDIF file (e.g., any of the analysis results of the ICMC 2000 Analysis/Synthesis Comparison) with any of a variety of inter-frame parameter interpolation techniques. We call our solution "open-ended" because instead of comparing interpolation techniques on a particular set of our own analysis results we have provided general-purpose software that can compare interpolation techniques on any sinusoidal model represented with the now-common SDIF standard.

2. SDIF SINUSOIDAL TRACK MODELS

An SDIF file can contain any number of SDIF streams; each SDIF stream represents one sound description. (Most SDIF files contain a single stream, but SDIF can also be used as an archive format.) An SDIF stream consists of a time-ordered series of SDIF frames, each with a 64-bit floating-point time tag. SDIF frames need not be uniformly distributed in time, although in additive synthesis they typically are. Each SDIF Stream contains SDIF Matrices; the interpretation of the rows and columns depends on the Matrix Type ID.

SDIF has two nearly-identical sound description types for sinusoidal models: "1TRC" for arbitrary sinusoidal tracks, and "1HRM" for pseudo-harmonic sinusoidal tracks. 1TRC frame matrices have a column for "partial index", an arbitrary nonnegative integer used to maintain the identity of sinusoidal tracks across frames at different times (as they potentially are born and die). For 1HRM matrices the equivalent column specifies "harmonic number" but otherwise has the same semantics. For both types, the other three columns are frequency (Hz), amplitude (linear), and phase (radians).

We use the terminology "analysis frame" as a synonym for "SDIF frame", because for sinusoidal track models, each SDIF frame generally results from the analysis (FFT, peak picking, etc.) of a single windowed frame of the original input sound. We say "synthesis frame" to indicate the time spanned by two adjacent SDIF frames, because the synthesizer must interpolate all sinusoid parameters from one analysis frame to the next. In other words, an "analysis" frame is a single SDIF frame, while a "synthesis" frame is the time spanned by two adjacent SDIF frames.

3. SOFTWARE ARCHITECTURE

Our synthesizer is implemented in the matlab language and is available for download.[1] It uses IRCAM's SDIF Extension for Matlab [22], also available for download,[2] to read SDIF files from disk into Matlab data structures in memory. We designed the software architecture to make it as easy as possible to plug in different methods for interpolating sinusoidal track parameters between SDIF frames. Therefore there are two parts of the program.

3.1. The General-Purpose Part

The general-purpose part of the program (currently 360 lines of matlab code) handles everything except the actual synthesis, namely:

- reading from an SDIF file
- selecting a particular SDIF stream
- selecting an arbitrary subset of the SDIF frames
- error checking (for bad SDIF data)
- warning when a partial's frequency exceeds half the synthesis sampling rate
- making sure every partial track begins and ends with zero amplitude, possibly extending a partial's duration by one frame to create an amplitude ramp.
- matching partial tracks from frame to frame (handling births and deaths of tracks, etc.)
- allocating a persistent array to remember current phase values between synthesis frames
- dispatching to the appropriate synthesis subroutine and calling it for each synthesis frame
- returning the resulting time-domain signal as a matlab array

The arguments to the general-purpose synthesizer are

1. The name of the SDIF file
2. The sampling rate for the synthesis
3. The synthesis type (i.e., form of interpolation)
4. (Optional) SDIF stream number indicating which stream to synthesize. (If this argument is omitted, the function uses the first stream of type 1TRC or 1HRM that appears in the file.)
5. (Optional) list of which frames (by number) to synthesize. (If this argument is omitted, the function synthesizes all the frames of the stream.)
6. (Optional) maximum partial index number.

3.2. The Interpolation-Method-Specific Part

The remainder of the program is an extensible collection of synthesis "inner loop" subroutines (currently ranging from 22 to 85 lines of matlab code). Each implements a different interpolation method by synthesizing one synthesis frame. The arguments to each subroutine are:

1. Starting sample number to synthesize
2. Ending sample number to synthesize
3. An "additive synthesis to-do list" matrix whose rows are all the partials that appear in a given synthesis frame and whose columns are the starting and ending frequency, amplitude, and phase, plus the partial/harmonic index number (so that a synthesizer can keep track of each partial's running phase between frames)
4. Synthesis sampling interval (reciprocal of the synthesis sampling rate)

The result is accumulated into the array `output`, which is a global variable so that it can be passed by reference.

[1] http://ccrma.stanford.edu/~matt/additive
[2] http://www.ircam.fr/sdif/download.html

4. INTERPOLATION TECHNIQUES

4.1. Stair Step (no interpolation)

Here is the "stair-step" synthesizer, performing no interpolation whatsoever, given as the simplest example of a synthesis function that works in this architecture. Note that it does not use the new values of frequency, amplitude, or phase; all three parameters will generally be discontinuous at synthesis frame boundaries.

```
function stairstep(fromSamp, toSamp, todo, T)
 for j = 1:size(todo,1)
  f =  todo(j, OLDFREQ) * 2 * pi;
  a =  todo(j, OLDAMP);
  p =  todo(j, OLDPHASE);
  dp = f * T;
  % Inner loop:
  for i = fromSamp+1:toSamp
    output(i) = output(i) + a*sin(p);
    p = p + dp;
  end
 end
return
```

4.2. Stair-Step with Running Phase

This synthesis function also does no interpolation, but it avoids phase discontinuities by using the "running phase" technique: initial phase for each partial is respected when it first appears; afterwards, phase is just the integral of the instantaneous frequency (and subsequent phase values from the analysis are therefore ignored). Each partial's running phase is stored in the persistent global array `phase`, which is indexed by each partial's SDIF partial index.

4.3. Linear Amplitude & Frequency, Running Phase

This function linearly interpolates amplitude and frequency across each synthesis frame. The inner loop is based on two variables: `dp`, the per-sample change in phase (proportional to frequency), and `ddp`, the per-sample change in `dp` (to provide linear interpolation of instantaneous frequency):

```
% Additive synthesis inner loop:
  dp = oldf * T;
  ddp = (newf-oldf) * T / R;
  a = olda;
  R = toSample - fromSample;
  da = (newa-olda) / R;
  for i = fromSample+1:toSample
    output(i)=output(i)+a*sin(phases(index));
    phases(index) = phases(index) + dp;
    dp = dp + ddp;
    a = a + da;
  end
```

4.4. Linear Frequency, dB amplitude, running phase

This function is the same as the previous one, except that amplitude is linearly interpolated on a dB scale (with 0 dB equal to a linear amplitude of 1).

4.5. Cubic Polynomial Phase Interpolation

This function uses a cubic polynomial to interpolate the unwrapped phase values [13]. Since frequency is the derivative of phase, the given starting and ending frequencies and phases provide a system of four equations that can be solved to determine the four coefficients of the cubic polynomial.

The unwrapping of phase works by first estimating the number of complete cycles each sinusoid would go through if the frequency were linearly interpolated. This is then minimally adjusted so that the ending phase, when wrapped modulo 2*pi, gives the correct ending phase as specified in the SDIF file. It can be shown that this rule is equivalent to the closed-form expression derived in [13] for obtaining "maximally smooth" phase trajectories.

5. SOUND RESULTS

The original reason for writing this program was to test the audibility of the difference between linear frequency interpolation and cubic phase interpolation. We conducted informal listening tests to compare the different interpolation techniques. We offer the following opinions only to give some sense of these perceptual differences; we hope that others will use our contributed software to perform their own listening tests.

Comparing cubic phase to linear frequency, the first result was to uncover some bugs with inconsistent interpretations of the SDIF phase parameter. In particular, there were cases for which reaching the specified phases entailed an audible amount of frequency skew within a frame. As a safeguard against this, synthesis systems could have a maximum frequency-deviation limit which, when reached, provides the phase closest to the desired phase while obeying the frequency deviation limit. For SDIF files with reasonable phase values, a typical problem with cubic phase interpolation is a "raspy" frame-rate artifact. In some cases the cubic phase interpolation sounds better, for example, for the SDIF files produced by the IRCAM analysis, the bongo roll is more "crisp" and the angry cat has more "depth" with cubic phase interpolation. In many cases the difference between running phase and cubic phase was not audible to us, including the tampura, giggle, speech (male voice saying "research" at 44.1k), and the acoustic guitar from the IRCAM analysis.

The stair-step technique, as one would expect, had "buzzing" artifacts caused by frequency and amplitude discontinuities at the synthesis-frame-rate. However, a surprisingly large minority of SDIF files from IRCAM sounded good when synthesized by this technique. The yodel sounded perfect with stair-step interpolation, but only with discontinuous phase. Conversely, trumpet and khyal vocals had only a few artifacts with running phase, while discontinuous phase was much worse. Acoustic guitar was nearly perfect in both cases.

In almost all cases there was no perceptual difference between linear interpolation and dB amplitude interpolation. The one exception was the berimbao, which had slightly more "depth" with dB interpolation.

6. FUTURE WORK

Our convenient plug-in architecture invites the implementation of additional interpolation techniques. In particular, we would like to include 5^{th}-order polynomial phase (if an SDIF representation for frequency slope were to be defined), some methods based on overlap-add, as well as Laroche's method based on non-overlapping inverse transforms [23].

We invite members of the community download our software, synthesize their own sinusoidal models with multiple interpolation methods, and compare the results. In addition, we believe that our framework could be the basis for a new method for formally expressing the semantics of interpolation techniques: providing an implementation of the synthesis.

We believe that this implementation has pedagogical value in teaching additive synthesis; we would like to develop further explanatory material, student projects, etc.

Obviously our informal listening tests are no substitute for proper psychoacoustic listening tests that could quantify the perceptual differences among interpolation methods (e.g., along the lines of [9, 24, 25]).

We would like to extend SDIF to allow an explicit representation of interpolation type for each parameter. For a small amount of extra storage, one could know what form of synthesis interpolation was assumed when the file was created; using an alternate form of interpolation would then be an explicit transformation of the sound model on par with frequency scaling, rather than a guess on the part of the person running the synthesizer.

7. ACKNOWLEDGEMENTS

Chris Chafe, Adrian Freed, Diemo Schwarz, David Wessel.

8. REFERENCES

[1] Risset, J.C. and M.V. Mathews, Analysis of musical-instrument tones. Physics Today, 1969. 22(2): p. 23-32.

[2] Schafer, R.W. and L.R. Rabiner, Design and simulation of a speech analysis-synthesis system based on short-time Fourier analysis. IEEE Transactions on Audio and Electroacoustics, 1973. AU-21(3): p. 165-74.

[3] Crochiere, R.E., A weighted overlap-add method of short-time Fourier analysis/synthesis. IEEE Transactions on Acoustics, Speech and Signal Processing, 1980. ASSP-28(1): p. 99-102.

[4] McAulay, R.J. and T.F. Quatieri. Mid-rate coding based on a sinusoidal representation of speech. IEEE International Conference on Acoustics, Speech, and Signal Processing. 1985. Tampa, FL, USA: IEEE.

[5] Julius O. Smith, I., PARSHL Bibliography. 2003. (http://ccrma.stanford.edu/~jos/parshl/Bibliography.html)

[6] Amatriain, X., et al., Spectral Processing, in DAFX - Digital Audio Effects, U. Zolzer, Editor. 2002, John Wiley and Sons, LTD: West Sussex, England. p. 373-438.

[7] Quatieri, T.F. and R.J. McAulay, Audio Signal Processing Based on Sinusoidal Analysis/Synthesis, in Applications of DSP to Audio & Acoustics, M.K.a.K. Brandenburg, Editor. 1998, Kluwer Academic Publishers: Boston/Dordrecht/London. p. 343-416.

[8] Risset, J.-C. and D. Wessel, Exploration of Timbre by Analysis Synthesis, in The Psychology of Music, D. Deutsch, Editor. 1999, Academic Press: San Diego.

[9] Andersen, T.H. and K. Jensen, Importance and Representation of Phase in the Sinusoidal Model. J. Aud. Eng. Soc, 2004. 52(11): p. 1157-1169.

[10] Wright, M., et al., Analysis/Synthesis Comparison. Organised Sound, 2000. 5(3): p. 173-189.

[11] Beauchamp, J.W. Unix Workstation Software for Analysis, Graphics, Modification, and Synthesis of Musical Sounds. in Audio Engineering Society. 1993.

[12] Ding, Y. and X. Qian, Processing of Musical Tones Using a Combined Quadratic Polynomial-Phase Sinusoid and Residual (QUASAR) Signal Model. Journal of the Audio Engineering Society, 1997. 45(7/8): p. 571–585.

[13] McAulay, R.J. and T.F. Quatieri, Speech Analysis/Synthesis Based on a Sinusoidal Representation. IEEE Trans. Acoustics, Speech, and Signal Processing, 1986. ASSP-34(4): p. 744-754.

[14] Rodet, X. The Diphone program: New features, new synthesis methods, and experience of musical use. in ICMC. 1997. Thessaloniki, Hellas: ICMA.

[15] Rodet, X. Sound analysis, processing and synthesis tools for music research and production. in XIII CIM00. 2000. l'Aquila, Italy.

[16] Girin, L., et al. Comparing the Order of a Polynomial Phase Model for the Synthesis of Quasi-Harmonic Audio Signals. in IEEE Workshop on Applications of Signal Processing to Audio and Acoustics. 2003.

[17] Röbel, A. Adaptive Additive Synthesis using spline based parameter trajectory models. in ICMC. 2001. Havana, Cuba: ICMA.

[18] Serra, X. and J. Smith, III, Spectral modeling synthesis: a sound analysis/synthesis system based on a deterministic plus stochastic decomposition. Computer Music Journal, 1990. 14(4): p. 12-24.

[19] Levine, S.N., Audio Representations for Data Compression and Compressed Domain Processing, in Electrical Engineering. 1998, Stanford: Palo Alto, CA. (http://ccrma.stanford.edu/~scottl/thesis.html)

[20] Fitz, K., L. Haken, and P. Christensen. Transient Preservation Under Transformation in an Additive Sound Model. in ICMC. 2000. Berlin, Germany: ICMA.

[21] Wright, M., et al., Audio Applications of the Sound Description Interchange Format Standard. 1999, Audio Engineering Society: Audio Engineering Society 107th Convention, preprint #5032. (http://cnmat.CNMAT.Berkeley.EDU/AES99/docs/AES99-SDIF.pdf)

[22] Schwarz, D. and M. Wright. Extensions and Applications of the SDIF Sound Description Interchange Format. in International Computer Music Conference. 2000. Berlin, Germany: ICMA.

[23] Laroche, J., Synthesis of Sinusoids via Non-Overlapping Inverse Fourier Transform. IEEE Transactions on Speech and Audio Processing, 2000. 8(4): p. 471-477.

[24] Chaudhary, A., D. Wessel, and L.A. Rowe. Listener Evaluation of Reduction Strategies for Sinusoidal and Resonance Models. in International Computer Music Conference. 2001. Habana, Cuba: ICMA.

[25] Grey, J.M. and J.A. Moorer, Perceptual evaluations of synthesized musical instrument tones. Journal of the Acoustical Society of America, 1977. 62(2): p. 454-62.