Spline Modeling of Audio Signals with Cycle Interpolation ⋆

Matt Klassen[1]

DigiPen Institute of Technology, Redmond, WA 98052, USA
mklassen@digipen.edu
http://www.digipen.edu

Abstract. In this paper we introduce methods to model brief audio signals with cubic splines, presupposing a fundamental frequency f_0. The signal is broken up into cycles using zero-crossings, and each cycle is modeled with a C^2 cubic interpolating spline based on some target number of interpolation points. We reduce the data in the model by reducing the number of key cycles which are used to generate intermediate cycles by the method of interpolation of B-spline coefficients, or cycle interpolation.

Keywords: Spline · Audio · Signal · Interpolation · Cycle

1 Introduction

1.1 Motivation and Background

The modeling project which we summarize in this paper has grown out of collaboration on the research and software project UPISketch [2], which provides a framework for graphical manipulation of sound. The UPISketch software makes use of splines to form models of curves drawn by the user. These graphical gestures are translated through the spline model into musical gestures which modulate elements such as pitch, or fundamental frequency. The spline representation can be thought of as a discrete set of B-spline coefficients, which is an intermediary between the graphical and musical gestures. In this paper we focus on the modeling of timbre with splines, through the evolution of cycles. The notion of cycles, instead of periods, is meant to imply that we expect that realistic waveforms will be at best *almost periodic*, but not exactly. So, the B-spline coefficients will form a discrete representation of timbre.

The use of piecewise polynomials, or splines, to model audio signals has its origins in the familiar piecewise linear audio generators such as square, triangle and sawtooth waveforms. Higher degree piecewise polynomial models, in particular cubic splines, were used as the basis for waveforms in Nick Collins' work [3], in which a single cycle, or period, of the waveform can be manipulated by the movement of control points. Collins developed software which provides the user with the ability to modulate the timbre of the resulting waveform by moving control points. Additionally, Collins uses the analogy of key-framing from animation to "morph" the shape of successive cycles between two key cycles. Our approach is similar but begins with the approximation of a brief audio sample with interpolating cubic splines. In this way the goal is to start with something which models a recorded sound quite closely, but might also be used for synthesis.

⋆ Supported by DigiPen Institute of Technology.

In addition to timbral shaping, B-splines have been used in the modeling of envelopes for f_0 synthesis, or fundamental frequency, as in [4]. The authors say in their introduction: “While B-splines have been widely used in computer graphics, very few applications can be found in the field of sound processing.” We propose that it is computationally feasible to model audio signals at the cycle level, especially for short segments with a perceived f_0 in the lower end of the audible frequency range, say below 1000 Hz. Although it would also be reasonable to model the envelope with B-splines, in this paper we extract the maximum amplitude value per cycle, from the original audio sample, and apply this as a normalization factor to the B-spline model of each cycle.

As in Collins' work, we also take some motivation from animation with key framing. In this approach, the cycle plays the role of one frame in animation, so we have key cycles which are used to generate the intermediate cycles. Both the choice of interpolation points to represent a key cycle, and the choice of the key cycles, affect the amount of data used to define, and the quality of, the final model. Another analogy with animation is useful: the key frames might come from an artist's imaginative drawing, or they might come from motion capture data. In the same way, one key cycle might come from some idea or model of timbre, or it might come from one cycle which is captured from a particular audio recording. In this paper we explore mostly the latter case.

The spline models of audio data that we propose can also be considered from a few other perspectives:

1. They are *low resolution* models, since they can be used to reconstruct an audio signal from a smaller set of data.

2. They are *time domain* models, differing from typical compression models which work with frequency bands.

3. They are *locally computable* models, which means that the reconstruction of an approximation of the full resolution signal can be computed from a smaller amount of data, the B-spline coefficients, which give a local representation of the audio signal.

This reduction of data, from the original samples, happens in two basic directions: first in the use of some fraction of the number of samples per cycle as interpolation points, and second in the use of some fraction of the cycles as key cycles. Low resolution audio models may also be of interest in realtime rendering of audio where prioritization and level of detail can be important.

A word about regularity: Both for the placement of interpolation points in one cycle, and the selection of key cycles, we begin by choosing regularly spaced points and cycles. This is partly in order to make these choices work nicely together. For example, if the interpolation points are not of the same number and regularly spaced, from cycle to cycle, then interpolation between cycles becomes much more difficult. There are many schemes, to be considered later, which attempt to take into account the amount of variation in one cycle, changes in local curvature, and other properties which suggest that a non-regular placement of interpolation points would be more optimal.

First, we construct the basic model of an audio sample, cycle by cycle. This means that there is no cycle interpolation. Then we summarize various techniques for doing the cycle interpolation, and follow this by some observations on problems that arise in this process. Finally, we point to further development and questions.

2 The basic model

The basic model consists of a cubic spline on each cycle which is used to recompute the same number of samples in the original audio file on each cycle. This preserves any irregularity in lengths of cycles, which is expected. We now describe more details in the context of a basic reference example. An important point is that we consider the original audio data as forming a piecewise linear function of time, so that we can compute the value at any time, not just at samples or integer points. This also allows us to work with zero crossings between samples using linear interpolation.

For a reference sample we have used a guitar pluck with f_0 approximately 440Hz recorded at standard CD quality (sample rate 44100 Hz and bit depth 16). The file has length 60184 samples, or about 1.37 seconds. Our first step is to determine all zero-crossings in the audio sample, which we treat as a piecewise linear function of time measured in samples. The zero-crossings are typically not integer points. We will refer to the audio signal values $x(t)$ for time values t at or between samples. The user chooses a fundamental frequency f_0 which then determines an approximate period or cycle length. The choice of such f_0 might be based on spectral analysis or prior knowledge of the audio sample, for instance a recorded instrument sound. This f_0 "guess" is then used iteratively to compute the end of the next cycle by choosing the closest zero crossing. This way cycles are defined using zero crossings closest to the period length guess. In the reference example, we choose to use $f_0 = 220$, one octave below the supposed 440 (guitar pluck on first string fifth fret). This can be seen as a useful guess from the shape of the audio signal graph. With $f_0 = 220$ there are 1531 zero-crossings, of which 302 are used as cycle endpoints. The average number of samples per cycle is quite steady with mean value very close to 200. However, there are some problems that inevitably show up with this method. First, it may be that the evolution of cycles based on zero crossings leads to discontinuities or abrupt changes in cycle shape. We will illustrate this with cycle number 181 in the reference example. Second, it may be that the model improves by keeping the cycle lengths fixed, say at the average value, rather than mimicking the original cycle lengths. We will return to these questions in the context of cycle interpolation.

Continuing with the basic model, once the cycles are determined by the sequence of zeros z_i, $i = 0, \dots, p$, we do a cubic B-spline fit to the audio sample data. To specify this spline function, say $f(t)$, we need to make various choices. Many of these choices are for reasons of simplicity, but may be revisited or subject to change later. The degree $d = 3$ is the natural and most common default for spline modeling in almost any context. It allows us to achieve second order derivative continuity for the smallest cost. (For details on splines, bases, and knot sequences, we refer to [1].)

When working with one particular cycle, it is useful to normalize the interval to $[0, 1]$. A cubic spline on $[0, 1]$ is a sequence of cubic polynomials $p_i(t)$ on some connected sequence of k subintervals $[u_i, u_{i+1}]$ which can be given by a sequence of *breakpoints* $0 = u_0 < u_1 < u_2 < \cdots < u_{k-1} < u_k = 1$. Rather than work with an explicit polynomial sequence, we use a B-spline basis for the vector space of all such splines $f(t)$ which have continuous second derivative on the interval $[0, 1]$. The sequence of k subintervals can be chosen to have uniform length as a reasonable starting point. The dimension of the vector space V of C^2 cubic polynomial splines on the sequence of k subintervals is then $n = k+3$. This means that each cubic spline function $f(t)$ is a linear combination of n basis functions:

$$f(t) = c_0\mathcal{B}_0(t) + \cdots + c_{n-1}\mathcal{B}_{n-1}(t).$$

To solve for such an f as an interpolating spline, we need n points (t, s) with inputs $t \in [0, 1]$ and outputs $x(t) \in [-1, 1]$ obtained from the audio data function. There are many possible B-spline bases for this vector space V, which can be specified by a choice of *knot sequence*, which is a non-decreasing sequence of numbers which includes the subinterval breakpoints $u_1, \ldots, u_{k-1}$ and another $d+1 = 4$ values on either end. With these conditions we write the knot sequence as $\mathbf{t} = \{t_0, \ldots, t_N\}$ where $t_0 \leq t_1 \leq t_2 \leq t_3 \leq 0$, and $1 \leq t_{N-3} \leq t_{N-2} \leq t_{N-1} \leq t_N$.

We prefer to use the following knot sequence $\mathbf{t}$ which encodes some information at the endpoints of the cycle:

$$\mathbf{t} = \{t_0, \ldots, t_N\} = \{0, 0, 0, 0, \frac{1}{k}, \frac{2}{k}, \ldots, \frac{k-1}{k}, 1, 1, 1, 1\}.$$

Writing the B-spline basis functions associated to $\mathbf{t}$ as

$$\mathcal{B}_0(t), \mathcal{B}_1(t), \ldots, \mathcal{B}_{n-1}(t)$$

we note that $\mathcal{B}_0(0) = 1$ and $\mathcal{B}_{n-1}(1) = 1$, and all the other basis splines vanish at both 0 and 1. Since each cycle is defined based on endpoints which are zeros, we set $c_0 = c_{n-1} = 0$ and solve for the other $n-2$ coefficients for each cycle. In order to approximate the audio data in one cycle, we find an interpolating cubic spline which matches the (piecewise linear) audio data function $x(t)$ at $n-2 = k+1$ specified points. Using the interval $[0,1]$ we choose the $k-1$ subinterval breakpoints (u_i, $i = 1, \ldots, k-1$) and then add two more points at the middle of the first and last subintervals. Since we do not have derivative information for the audio data, this placement will help to approximate the derivatives at the ends. These choices are summarized below as the list of data for each spline which represents the model on an interval between two zero crossings:

- degree d = 3
- number of subintervals k
- subinterval sequence: $[u_i, u_{i+1}] = [\frac{i}{k}, \frac{i+1}{k}]$, $i = 0, \ldots, k-1$
- knot sequence: $\mathbf{t} = \{t_0, \ldots, t_N\} = \{0, 0, 0, 0, \frac{1}{k}, \frac{2}{k}, \ldots, \frac{k-1}{k}, 1, 1, 1, 1\}$
- dimension of vector space V of B-spline functions: $n = k + 3 = N - 3$
- B-spline basis functions: $\mathcal{B}_0(t), \mathcal{B}_1(t), \ldots, \mathcal{B}_{n-1}(t)$
- interpolating spline: $f(t) = c_1\mathcal{B}_1(t) + \cdots + c_{n-2}\mathcal{B}_{n-2}(t)$ ($c_0 = c_{n-1} = 0$)
- input values: $s_0 = 0, s_1 = \frac{1}{2k}$, $s_i = u_{i-1}$, $i = 2, \ldots, n-2$, $s_{n-2} = 1 - \frac{1}{2k}, s_{n-1} = 1$
- target (signal) values: $x(s_0), x(s_1), \ldots, x(s_{n-1})$

Now we can define the basic model to be the sequence of zero crossings z_j, $j = 0, \ldots, p$ and the sequence of spline functions $f_j(t)$ on the interval $[z_j, z_{j+1}]$. We also refer to each of these splines and their associated data as one "cycle" C_j, so that the basic model is the sequence of cycles C_j for $j = 0, \ldots, p-1$. Further, let the B-spline coefficients for cycle C_j be c_i^j, $i = 0, \ldots, n-1$.

The B-spline functions can be defined with divided differences as:

$$\mathcal{B}_i^d(t) = (-1)^{d+1}(t_{i+d+1} - t_i)[t_i, t_{i+1}, \ldots, t_{i+d+1}](t-x)_+^d$$

or using the (de Boor-Cox) recursion formula as

$$\mathcal{B}_i^d(t) = \frac{t - t_i}{t_{i+d} - t_i}\mathcal{B}_i^{d-1}(t) + \frac{t_{i+d+1} - t}{t_{i+d+1} - t_{i+1}}\mathcal{B}_{i+1}^{d-1}(t)$$

with base case:

$$\mathcal{B}_i^0(t) = \begin{cases} 1, & t_i \leq t < t_{i+1} \\ 0, & \text{elsewhere} \end{cases}.$$

We compute values of the B-spline functions, or of the interpolating spline f, with nested linear interpolation, known as the de Boor algorithm:

– (Stage zero:) Set $c_i^0 = c_i$ for $i = 0, \ldots, N - d - 1$.

– For $t \in [t_d, t_{N-d})$ set J to be the index so that $t \in [t_J, t_{J+1})$.

– (Stage p:) For $p = 1, \ldots, d$, for $i = J - d + p, \ldots, J$: set

$$c_i^p = \frac{t - t_i}{t_{i+d-(p-1)} - t_i} c_i^{p-1} + \frac{t_{i+d-(p-1)} - t}{t_{i+d-(p-1)} - t_i} c_{i-1}^{p-1}$$

– $f(t) = c_J^d$.

Note: to solve for the coefficients c_i, $i = 0, \ldots, n-1$, we set up the interpolation problem as a linear system by requiring that f agrees with the audio output data at each of the n input values. Since the input values are evenly distributed, we are guaranteed a unique solution, according to the Schoenberg-Whitney Theorem on B-spline interpolation (see [5]).

If we compute the basic model for the reference example with $f_0 = 220$ and $k = 47$ ($d = 3$, and dimension $n = 50$) the model takes about 83 seconds to compute on a mac laptop in our implementation with JUCE. This model uses about 1/4 of the original samples in the audio file and matches the signal very closely. In terms of audio quality, this model is close enough to the original to say that there are no obviously noticeable artifacts, or distortion.

It is not our intention to present this as a model of data compression, but rather as a starting point to understand how cycles evolve in time during the course of a brief audio sample, and to investigate whether it is possible to simulate this evolution through artificial means which use a smaller set of data. The fact that the model is an adequate representation of an original audio sample can be thought of in the way that the position of a human figure can be approximately captured by sensors which measure the positions of a number of points on the surface of the figure in 3D space.

3 Cycle Interpolation

First, we should specify what it means to interpolate between two cycles, each represented by an interpolating cubic spline, as in the basic model. Keeping the assumption that all cycles still have a common basis of n B-splines, the first obvious approach is to linearly interpolate between pairs of B-spline coefficients with the same index. But it is also reasonable to consider using quadratic or cubic splines. In all of these cases we will refer to this set of splines as *meta-splines*. In order to distinguish these from the splines used to generate each cycle in the basic model, we refer to the latter as *cycle-splines*. The cycle-splines then are modeling the timbre of the audio signal, whereas the meta-splines are controlling the evolution of the timbre, or the way in which one cycle morphs between key cycles.

Second, it is natural to use an approximation of the envelope of the audio signal which is being modeled. Since we are already partitioning into cycles, it is straightforward to extract the maximum absolute value in each cycle, then when computing a cycle-spline to normalize the B-spline coefficients so that they produce the same maximum.

There should be one meta-spline per B-spline coefficient of the cycle-splines. For example, if we are interpolating between key cycles C_{j_1} and C_{j_2}, then to compute the first B-spline coefficient c_0^j of cycle C_j, with $j_1 < j < j_2$, we use meta-spline $g_0(t)$. This meta-spline will have inputs and targets set proportionally to match the sequence of key cycles.

This means that if $g_0(t_{j_1}) = c_0^{j_1}$ and $g_0(t_{j_2}) = c_0^{j_2}$ then we compute for some t_j between t_{j_1} and t_{j_2}:

$$c_0^j = g_0(t_j).$$

Note that in forming the meta-splines it may be that we do not place the inputs for interpolation in a uniform sequence. This arises naturally when we choose to place key cycles more densely near the beginning than at the end, for instance when modeling an audio sample which has a more varied attack phase and a much less varied sustain and release phase, or tail.

We can now define the Cycle Interpolation Model to be given by the data:

– key cycles $C_{j_0}, \ldots, C_{j_{q-1}}$

– B-spline coefficients of key cycle C_{j_r}: $c_i^{j_r}$, $i = 0, \ldots, n-1$

– meta-splines $g_i(t)$ with target values c_i^j, $j = j_0, \ldots, j_{q-1}$

– max values for envelope α_i, $i = 0, \ldots, p-1$

– cycle zeros z_i, $i = 0, \ldots, p$

Two important things are still to be determined: 1) how to determine the form of the meta-splines, and 2) how to choose the key cycles.

Regarding the choice of meta-splines: If we use degree 1, or simply linear interpolation between corresponding B-spline coefficients, then the effect is similar to a cross-fade between two key cycles. As Collins points out, this is not just a cross-fade between B-spline coefficients (or control points) but with no other special conditions it is also a cross-fade between audio sample values. But since we are allowing for varying cycle lengths $z_{i+1} - z_i$ and we are also normalizing by the envelope values, the result is not strictly a cross-fade. For simplicity and computational speed we have mostly used the linear case, but it is interesting to also note some properties of the cubic case.

It is tempting to use both cubic meta-splines and also cubic cycle-splines, if for no other reason than elegance and consistency. But there are a few pitfalls. There are limits to the practical computation of linear systems used in solving for each interpolating spline. In the case of cycle-splines this dimension is $n = k+3$, the dimension of the vector space of splines. In the case of the meta-splines the dimension is equal to q, the number of key cycles. If we have say 300 cycles, as with the reference example, and we choose to use half of these as key cycles, then we have dimension $q = 150$ which can be prohibitively large for running experiments with modeling. Fortunately, we are interested in models with fewer key cycles. Another pitfall is that there can be too much variation in the cubic spline model which fits the key cycle data, especially if there are long gaps between key cycles, as may be the case in the tail of a signal. This suggests that there may be reason to explore other types of spline models for the meta-splines.

We also note a type of duality that arises when using cubic splines as both meta-splines and cycle-splines. In particular, the number of meta-splines is n, which is also the dimension for cycle-splines, and the number of cycle-splines is q which is also the dimension of the meta-splines. When generating the output audio data from a model, the spline function $f_j(t)$ is evaluated N_j times according to the number of samples per cycle N_j. Similarly, when producing intermediate B-spline coefficients for non-key cycles, the spline function $g_i(t)$ is evaluated Q times according to the total number of cycles Q to be generated by the model. Since we are interested in models which depend on a small amount of data, this can lead to some special cases. Suppose, although N_j typically varies, that it is constant: $N_j = N$ for all j. Additionally, suppose that $q = n$, and that $Q = N$, and further that $f_i = g_i$ for all i. This means our model is generated by one set of cubic splines which play the dual roles of cycle-splines and meta-splines. Since $Q = N$, the number cycles coincides with the number of samples per cycle, and we only need to evaluate each spline at the appropriate points once. This assumes that the intermediate cycles are represented by uniform subdivision of the interval $[0, 1]$, just as the samples in one cycle are. It also implies that each meta-spline has the value zero at the ends, since this is required for the cycle-splines. If a cycle interpolation model satisfies all of the above requirements, we call it a *reflexive* model.

Next, we consider different choices of key cycles.

Regular Cycle Interpolation

The first approach is to use regularly spaced key cycles, which we call *regular cycle interpolation*. If the entire set of cycles is C_i, $i = 0, \ldots, p-1$, then we specify some positive integer m and choose cycles C_{jm} as key cycles, $j = 0, \ldots, r$, with $rm \leq M-1$ and r maximal. There are two variations on this regular cycle interpolation, the first of which is to insist on the last cycle also being included. This allows for the obvious cycle interpolation between C_{rm} and C_{p-1}. The second variation is to not include the last cycle as key, and to simply use C_{rm} repeatedly for the final cycles. In this case, it useful to still follow the envelope of the original audio signal, as indicated above.

Exponential Cycle Interpolation

In order to focus more closely on the attack phase of a short audio sample, such as the reference example, it makes sense to choose more key cycles near the beginning and fewer towards the end, or tail. One useful sequence is to have key cycle indices 0 and then 2^i: $0, 1, 2, 4, 8, \ldots$ which we refer to as *exponential cycle interpolation*.

Fibonacci Cycle Interpolation

Similar to exponential, but a slightly denser pattern is $0, 1, 2, 3, 5, 8, \ldots$, which we call *Fibonacci cycle interpolation* using the recursion $k_{i+1} = k_i + k_{i-1}$.

These are some of the first types of sequences of key cycles that we found natural or useful. In the next section we discuss some of the problems that arise with cycle interpolation.

4 Problems with Cycle Interpolation

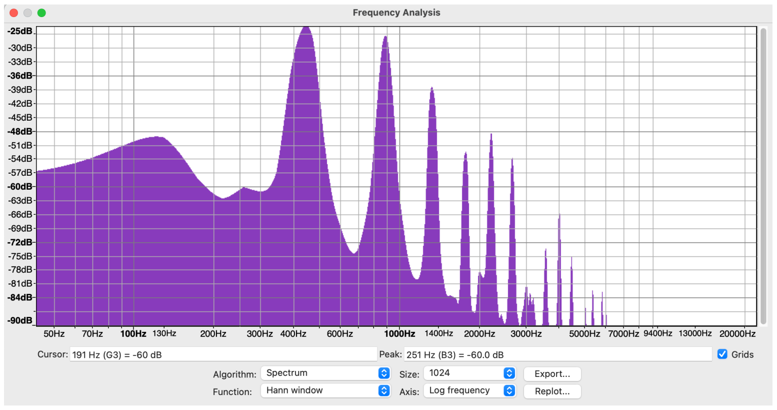
There are various problems which can cause spectral deficiencies or audio artifacts with cycle interpolation. We discuss two of these here: missing sub-harmonics, and cycle shape discontinuities.

Missing sub-harmonics:

This is a natural defect of cycle interpolation, since it is not a priori designed to follow the sub-harmonic oscillatory pattern of an original audio waveform, but rather to fill in the non-key cycles artificially with B-spline interpolation. This can be observed with the reference example. In figure 1 is a spectral analysis of a portion of the reference example using Audacity software.

Next, in figure 2, is a similar plot for the model using the Fibonacci cycle interpolation with 15 key cycles. Although the frequency guess 220 Hz and higher harmonics are well represented, the

Fig. 1. Spectrum for Reference Example



Cursor: 191 Hz (G3) = -60 dB
Peak: 251 Hz (B3) = -60.0 dB
✓ Grids
Algorithm: Spectrum
Size: 1024
Function: Hann window
Axis: Log frequency

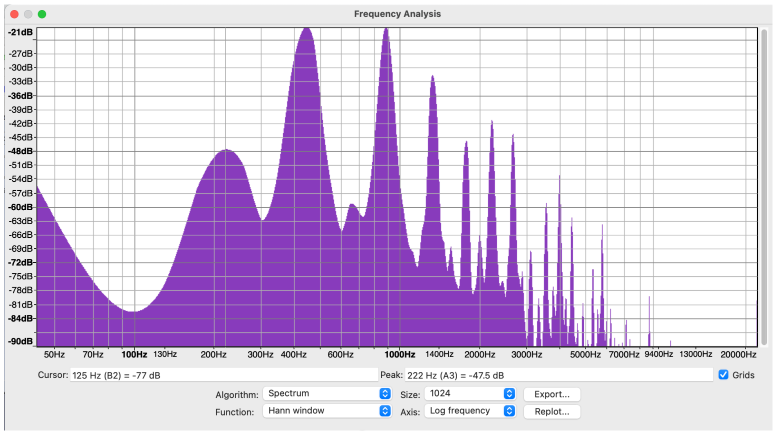
sub-harmonic 110 Hz is markedly absent. There are various ways to attempt to address this issue. Rather than delving into DSP post-processing, we prefer in this paper to first attempt to work with the spline model. This is a good opportunity to make a simple adjustment to the model which is surprisingly cheap and effective. We can represent the sub-harmonic with a quadratic spline, represented in the same basis as the cubic cycle-splines. This requires simply solving for a spline on one cycle of the form $A(1-t)t$ where $A > 0$ represents the appropriate amplitude borrowed from the spectrum of the original waveform. We then mimic a sinusoid on successive cycles by alternating between A and $-A$. The B-spline coefficients of this "parabolic sinusoid" can be added directly to the B-spline coefficients of each cycle-spline. Note that this method preserves the allowance of varying cycle lengths as well. The resulting spectrum is shown in figure 3.

This example brings up a point about audio rendering and mixing with spline models. Given several spline models, which are to be mixed, one can do the mixing prior to the final rendering, as long as the B-spline bases are consistent. This can lead to a significant reduction in computation. By consistent we mean that the models have the same degree d and the same number of subintervals per cycle k. These could be constant, or could vary but still agree per cycle. The two models do not need to have the same key cycles, but if they do there will be a greater reduction in computation.

Cycle shape discontinuities:

The next type of problem is due to the inherent inaccuracies which occur when attempting to break up an audio sample into cycles. This is illustrated in the reference example, with $f_0 = 220$ Hz (guess at fundamental frequency). With the help of our visualization software for cycle interpolation (written with JUCE) we note that a problem occurs in the region around cycle 180. There are 8 prominent stationary points in each cycle starting at cycle 70. The last of these is a relative minimum occurring just before the right endpoint of each cycle, which we will call feature A. We can see that feature A begins to drift upwards in cycles 100 through 170, then finally transitions above the time

Fig. 2. Spectrum of Model Without Sub-harmonic



Cursor: 125 Hz (B2) = -77 dB

Peak: 222 Hz (A3) = -47.5 dB

☑ Grids

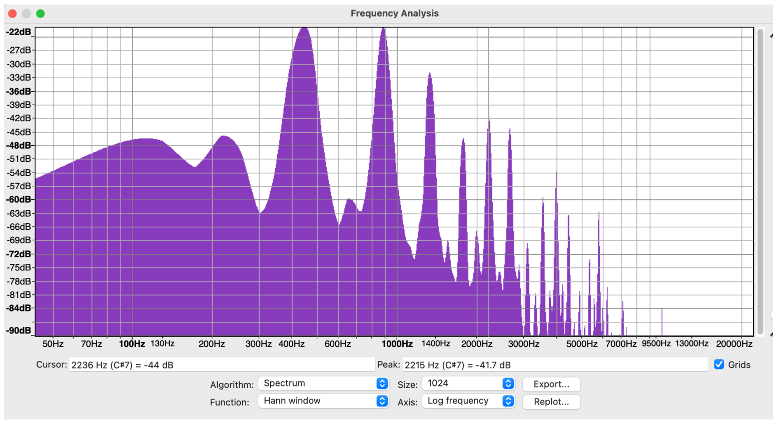
Algorithm: Spectrum

Size: 1024

Function: Hann window

Axis: Log frequency

Fig. 3. Spectrum of Model With Sub-harmonic



Cursor: 2236 Hz (C#7) = -44 dB

Peak: 2215 Hz (C#7) = -41.7 dB

☑ Grids

Algorithm: Spectrum

Size: 1024

Function: Hann window

Axis: Log frequency

axis by cycle 182. This is illustrated in figure 4 around cycle 180. This behavior of the graph is part of the natural evolution of the shape of the curve but it is a problem for cycle interpolation. We will call this problem a *shape discontinuity*, meaning an abrupt change in the shape of consecutive cycles. This is not a property of the original waveform, just of the cycle interpolation model. The original waveform has a smooth progression of cycles, but these cycles are not always best defined by zero crossings.

It is worth noting that this poses no problem for the basic model (without cycle interpolation) since the cycles are each modeled independently. In fact, in the basic model, the cycles could be assigned almost any sequence of lengths and cause no problems. It is also worth noting that the reference example has only one instance of an obvious shape discontinuity of this type.

To see the effect on cycle interpolation, we illustrate in figure 5 a graph which also includes the model (in blue) with regular cycle interpolation value $m = 5$. The shape discontinuity causes the B-spline coefficients of the model to change drastically between key cycles. It is no surprise that this causes audible artifacts in the resulting audio.

There are a few ways to mitigate this problem by adjusting the model. The most obvious is to add key cycles at and around the transition point, for instance by adding key cycles 178 through 182. This method does cause reduction in the audible artifacts, but does not really solve the problem which stems from the shape discontinuity. Another approach is to release the waveform at an earlier point from its obligation to match any more key cycles from the original waveform. In other words, this approach removes rather than adds key cycles. One obvious result is that the tail produced by the model will differ significantly from the original, but this may have a small effect on the outcome. For instance, if we use the Fibonacci cycle interpolation model for the reference example, the initial sequence is: $0, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 301$, with 14 key cycles. If we remove the last two then the tail produced by the model is based on cycle 144, and has only 12 key cycles. If these key cycles use about 1/3 of the samples per cycle (say $k = 63$) then we have a good model which removes most of the audible artifacts. The data in this model can be further reduced by setting the cycle length to a constant, say 200 samples. If we use the simplest type of meta-splines (linear interpolation) then the data can be measured by the number of floats needed to reconstruct the model as a fraction of the original audio data. We include in the model the normalization value of one float per cycle. In this case the percent of the original data stored in the model is about 0.183%.

We have found that cycle interpolation has the potential to form good models of recorded sounds with a small fraction of the original sampled data. There are interesting problems which occur in representing sub-harmonics and avoiding errors inherent in the process of breaking up audio signals into cycles. We are encouraged by the results so far and look forward to seeing these techniques used in the design of new sounds in UPISketch.

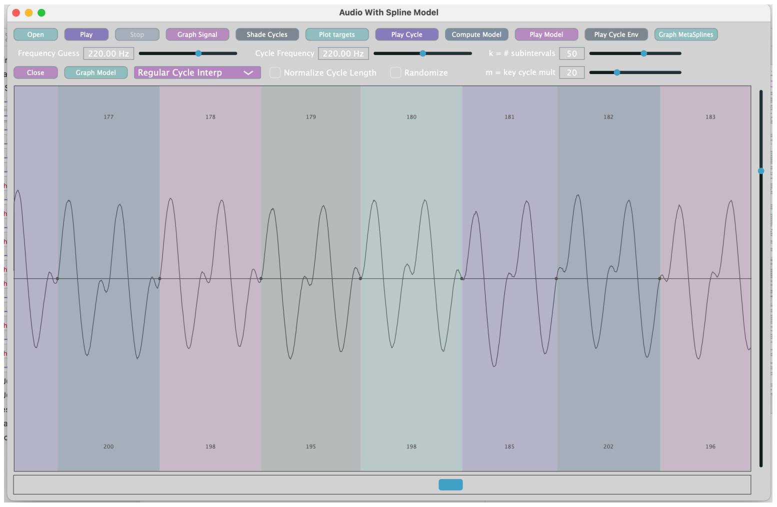
5 Future work

There are many aspects to explore in more detail. A few of these are:

1) From the sound design perspective, use cycle interpolation together with randomization, or cellular automata, to explore new timbres.

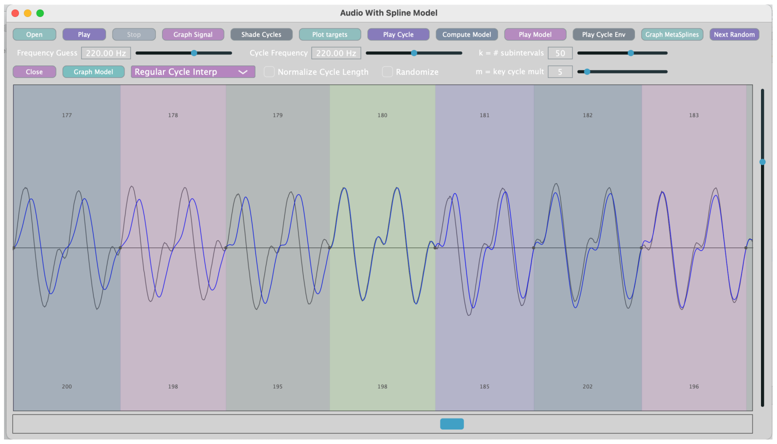
2) Investigate how shape discontinuities arise when cycle interpolation is applied to various musical instrument samples.

Fig. 4. Shape Discontinuity Near Cycle 180



Frequency Guess 220.00 Hz
Cycle Frequency 220.00 Hz
k = # subintervals 50
Regular Cycle Interp
Normalize Cycle Length
Randomize
m = key cycle mult 20

Fig. 5. Cycle Interpolation Near Cycle 180



Frequency Guess 220.00 Hz
Cycle Frequency 220.00 Hz
k = # subintervals 50
Regular Cycle Interp
Normalize Cycle Length
Randomize
m = key cycle mult 5

3) Use cycle interpolation to model phones for concatenative speech synthesis.

4) Reduce the data in cycle interpolation further, taking into account the prioritization of interpolation points based on discrete curvature.

5) Address the shape discontinuity problem by allowing cycles to be defined at points which are not zero-crossings.

References

1. Klassen, M.: Lecture notes on Bezier curves and polynomial splines. http://azrael.digipen.edu/notes/
2. UPISketch: The UPIC idea and its current applications for initiating new audiences to music. Bourotte, R., Kanach, S., Organised Sound, vol 24, no.3, 252-260, 2019.
3. Collins, N.: SplineSynth: An Interface to Low-Level Digital Audio. Proceedings of the Diderot Forum on Mathematics and Music, Vienna, 1999, ISBN 3-85403-133-5, pp 49-61.
4. Ardaillon, L., Degottex, G., Roebel, A.: A multi-layer F0 model for singing voice synthesis using a B-spline representation with intuitive controls. Interspeech 2015, Sep 2015, Dresden, Germany. hal-01251898v2.
5. de Boor, C.: A Practical Guide to Splines, revised edition, Springer-Verlag, New York (1980)