MUS 470/470L Special Topics

Matt Klassen – Fall 2021

Spline Interpolation of Audio Signals

In this document I will discuss some approaches to modeling audio signals with splines and suggest some sample projects. These methods can be similar to audio data compression and decompression if the model is built simply from an audio file by replacing some of the audio data by approximate values coming from splines which interpolate a subset of exact values from the audio file. On the other hand, these methods can be similar to physical modeling if the spline data is designed to fit a more generic model which mimics the physical properties which generated the data. It can be useful to think of our approach as lying between these two extremes, since we can generate models which adhere closely to original audio data, but they are also quite variable and are based on many parameters.

Basic Prototype

We start with a description of a basic program which can be a command line utility which operates on wav files as input/output. The idea is to model a short audio sample (say 1-5 sec) in smaller chunks, which could be for example cycles or periods. If a wave form is approximately periodic, as many sample libraries of musical instrument samples are, it may be possible to break up the short wav file into cycles. We will use a reference example of a guitar pluck with approximate fundamental frequency 450 Hz. At the sample rate 44100 Hz, this would give about $98 = 44100/450$ samples per cycle, if the wave form is periodic. In order to break the wave form into cycles we can use the guess at fundamental frequency $f_0 = 450$. Starting with the first zero crossing in the wav file, and using f_0 to compute the predicated period length, we can then find zero crossings closest to periods iteratively. Each cycle, or period, is then modeled with a basis of B-splines on a sequence of sub-intervals whose union is the given cycle interval. The number of subintervals k can be chosen to remain constant for all cycles. In our reference example we have about 98 samples per cycle, so we would choose some number of intervals less than 98. Since the wave form is not strictly periodic when we refer to cycles, these are now simply a sequence of chunks, or subintervals, of the audio file time line. For the reference example we have about 450 of these per second of audio data. A reasonable starting value for k is 30.

Zero crossings can be computed with float values between samples by using linear interpolation. This way the sequence of cycles is now a sequence of intervals with overlapping endpoints which are given as floats which lie somewhere between the first and last sample values. Note: if this method is being employed for a wave form without a particular fundamental frequency, then the value f_0 can just be taken as a way to compute a cycle length which is used to break the file up into chunks of equal length. This means that the chunks of audio do not have any particular pattern. In this case it is also not necessary to compute zero crossings. We will continue now to assume that indeed a meaningful f_0 is being used and that there will indeed be cycles computed with zero crossings.

When working with one particular cycle, it is useful to normalize the interval to $[0, 1]$ and to then compute the k subintervals simply as $[i/k, (i + 1)/k]$, $i = 0, \ldots, k - 1$. Once the B-spline model is computed on this subinterval it can then be translated and scaled to the particular cycle in order to compute the output values of the model. Next, we need to select input values on the interval $[0, 1]$ which we will used to find the B-spline coefficients for the model on a particular cycle. These input values can be the same for all cycles, normalized to $[0, 1]$. We will use default degree $d = 3$. In this case we need to specify $k + 3$ values in the interval $[0, 1]$ as interpolation points, since $k + 3$ is the dimension of the vector space of C^2 cubic splines on a sequence of k subintervals. The values must be spread out reasonably well, according to the Schoenberg-Whitney Interpolation Theorem: it is sufficient to put input values at the sequence consisting of the sub-interval endpoints i/k, $i = 0, \ldots, k$, and two more values: $1/2k$ and $1 - 1/2k$ for a total of $k + 3$ values. (For higher degrees $d > 3$, a similar approach can be used, inserting points half-way between sub-interval endpoints to bring the total up to $k + d$.)

With the above system of inputs, one computes the output values for a particular cycle using the given inputs and linear interpolation between audio data points. A set of $k + 3$ B-spline coefficients is then computed by solving a linear system. The collection of all sets of B-spline coefficients for each cycle then makes up the model of the audio sample. The setup of the linear system can be done initially with truncated power spline basis, then solved and converted to a B-spline basis. This is done in the sample code. Bases for splines are also discussd below.

In summary, here is the above written as a programming assignment:

Spline Modeling Project 1 - Basic Spline Interpolation

Write a program which takes a short wav file and interpolates the audio data with cubic splines. Use the following input parameters:

1. f_s = the sample rate of the input file
2. f_0 = guess at the fundamental frequency of the audio sample
3. k = the number of subintervals per cycle
4. d = the degree of the splines (default $d = 3$)

Output should be a wav file of the same length as input, with values determined from the spline functions constructed to interpolate the original data. Data should be broken up into cycles based on zero crossings closest to periods, using guess at fundamental frequency f_0. Optionally, the B-spline coefficients making up the model could be written to a text file, one cycle per line.

Bases for splines

Two bases are commonly used: the truncated power basis, and the B-spline basis. Each can be constructed for any sequence of k sub-intervals, say $[u_0, u_1, \ldots, u_k]$, and a degree d for the polynomial pieces on each interval, and an order of continuity r. For simplicity, assume the intervals are of uniform width, say $w = u_i - u_{i-1}$. Instead of one order of continuity r, a vector of continuity conditions can be specified, such as $\mathbf{r} = (r_1, r_2, \ldots, r_{k-1})$ so that the spline functions agree to order of continuity r_i at the break-point u_i, which excludes the endpoints. The default order of continuity will be $r = d-1$, which is the maximum finite r for piecewise polynomials of degree d. To achieve $r = d$ immediately elevates r to infinity, which means the piecewise polynomial is just a single polynomial across all the intervals.

The vector space $V = P^k_{d,r}[u_0, u_1, \ldots, u_k]$ of all polynomial splines of degree d with order of continuity $r = d-1$ on the intervals $[u_0, u_1, \ldots, u_k]$ has dimension $n = d + k$. So both of the bases will have n functions in them.

The truncated power basis functions are given by first choosing a knot sequence

$$\mathbf{t} = \{t_0, t_1, \ldots, t_{n-1}\}.$$

The simplest default should be to use the sequence u_i with d numbers appended to the left end of the sequence, so that for $d = 3$, for example, we get $t_0 = u_0 - 3w$, $t_1 = u_0 - 2w$, $t_2 = u_0 - w$, $t_3 = u_0$. Then the sequence continues with $t_{3+i} = u_i$, up to $i = k-1$. So the final value is $t_{n-1} = u_{k-1}$. Also, $n = k + d - 1 = k + 2$ for $d = 3$. With this simple default we also get that $r = d - 1 = 2$.

The truncated power function is defined for integer $k \geq 1$ as:

$$(t-c)^k_+ = \begin{cases} 0, & t < c \\ (t-c)^k & t \geq c \end{cases}$$

The truncated power basis functions attached to the knot sequence are obtained by wrapping the knot values by the form $(t-c)^{d-1}_+$ to give the set:

$$\{(t-t_0)^{d-1}_+, (t-t_1)^{d-1}_+, \ldots, (t-t_{n-1})^{d-1}_+\}.$$

The B-spline basis for the same vector space V described above is obtained from another knot sequence, similar to $\mathbf{t}$ above, but this time appending d values on the right as well as the left of the sequence. The simplest case is to use

$$\mathbf{t} = \{t_0, t_1, \ldots, t_N\},$$

with same starting values $t_0, t_1, \ldots, t_{n-1}$, but continuing with $t_{3+k} = u_k$, and $t_{3+i} = u_k + (i-k)w$ for $i = k+1$ up to $i = k+d-1$. So the final value is now $t_N = u_{k-1} + (d+1)w$.

A more general form for the knot sequence that gives a B-spline basis can be described by:

$$\{\alpha_0, \ldots, \alpha_d, u_1, \ldots, u_{k-1}, \beta_0, \ldots, \beta_d\},$$

where $\alpha_i \leq u_0$ and $\beta_i \geq u_k$, for $i = 0, \ldots, d$. (Check that the example cases described above adhere to these requirements.)

The B-splines associated to the knot sequence $\mathbf{t}$ above can be labeled

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

Solving for an explicit form of any B-spline can be tedious, but it turns out that we can find the coefficients for a spline in terms of these B-splines and then evaluate a linear combination with those coefficients, without ever writing down any explicit forms. This is done through the use of the DeBoor algorithm for B-spline evaluation. In general it is not necessary for the knot sequence to have uniform separation w between values. The knot sequence can also have consecutive equal values, as long as the multiplicity does not exceed $d+1$. In the case of multiple values, the basis splines are chosen with consecutively lower degrees, which accomodates representation of functions with lower orders of continuity at the break points.

Solving for an interpolating spline

Let $\{f_0, \ldots, f_{n-1}\}$ be a basis of splines of one of the two types above, for a vector space of splines on a sequence of intervals as above. Now suppose we want to solve for a function f as a linear combination of these basis functions which passes through a given set of data points. If there are n data points (x_i, y_i), for $i = 0, \ldots, n-1$, then it may be possible to solve for such an f. In fact, according to the Schoenberg-Whitney Theorem: as long as we have $t_i < x_i < t_{i+d+1}$, for each input x_i, where t_j refers to the B-spline knot sequence $\mathbf{t}$, then it will be possible to solve for f with a linear system approach. (More precisely, there exists a unique solution to the linear system below.) Let f be defined as:

$$f(t) = a_0 f_0(t) + a_1 f_1(t) + \cdots + a_{n-1} f_{n-1}(t).$$

Then we form each row of the linear system by plugging in each of the inputs x_i and setting equal to the output value y_i. Each row of the linear system then has the form:

$$a_0 f_0(x_i) + a_1 f_1(x_i) + \cdots + a_{n-1} f_{n-1}(x_i) = y_i.$$

Extracting the constants $f_j(x_i)$ as coefficients of the a_j, we can convert this to a row of an augmented matrix:

$$[f_0(x_i) \;\; f_1(x_i) \;\; \cdots \;\; f_{n-1}(x_i) \;\; | \;\; y_i].$$

The entire augmented matrix looks like this:

$$\begin{pmatrix} f_0(x_0) & f_1(x_0) & \cdots & f_{n-1}(x_0) & | & y_0 \\ f_0(x_1) & f_1(x_1) & \cdots & f_{n-1}(x_1) & | & y_1 \\ \vdots & \vdots & \vdots & \vdots & | & \vdots \\ f_0(x_{n-1}) & f_1(x_{n-1}) & \cdots & f_{n-1}(x_{n-1}) & | & y_{n-1} \end{pmatrix}.$$

For example, if we set $d = 3$, then we can use as inputs x_i the set of endpoints and breakpoints of intervals as a starting point. But this is only $k + 1$ values, and we have $n = k + d = k + 3$. So we can insert two more values into this list. It is convenient to assign those values as $x_1 = u_0 + \frac{1}{2}w$ and $x_{n-2} = u_k - \frac{1}{2}w$. The rest of the sequence then uses the u_j, so that $\{x_0, x_2, \dots, x_{n-3}, x_{n-1}\} = \{u_0, \dots, u_k\}$. This sequence then satisfies the requirement for a unique solution, which is $t_i < x_i < t_{i+d+1}$.

Data Reduction and Cycle Interpolation

Once a set of B-spline coefficients is chosen for a model of an audio signal, it can become a starting point for further modeling and reduction of data. One way to reduce the amount of data stored is to remove data points from a particular cycle based on some measure of significance to the model, such as local curvature (discrete or continuous version). For example, if the computed curvature at three consecutive data points falls below a threshold, the middle data point could be removed and the model recalculated. This would not only reduce the data but also change the interval sequence to be non-uniform, and change the B-spline basis to be used. Since one break-point between intervals contributes one knot value to the knot sequence, which corresponds to one truncated power basis function, it is straight-forward to recalculate the spline basis and solve for the coefficients. One disadvantage to this approach is that it does prioritize keeping models between cycles compatible, as removal of a data point in one cycle might not be practical in another cycle. If spline models are treated as a progression through a sequence of cycles, then it might make more sense to keep them uniform.

Another approach to reduction in the amount of data for a B-spline model is to replace some cycles with B-spline coefficients which are generated by interpolation between cycles. For instance, the B-spline coefficients for every other cycle could be calculated by linearly interpolating the B-spline coefficients from the cycle before and after the given one. There are many such schemes which generalize this one. For example, suppose that only every tenth cycle's B-spline coefficient set is saved, and also the lengths of the in-between cycles (given that this can be non-uniform based on the original audio data). The in-between cycles could be reconstructed by interpolating the B-spline coefficients one by one over many cycles. For example, suppose there are 700 cycles, and each cycle is broken into $k = 20$ subintervals and $d = 3$ so that there are $k + d = 23$ B-spline coefficients per cycle $b_0, \dots, b_{22}$. There are $700/5 = 140$ saved cycles. The b_0 values could be used to form an interpolating cubic spline on some sequence of intervals, and then the b_0 values for the missing cycles could be calculated from this spline. The same could be done for each of the b_i values. In order to minimize variation between data points, one could use the natural cubic spline which has second order derivative zero at the endpoints. We call this method *cycle interpolation.* It is possible to do cycle interpolation between evenly spaced *key cycles* or to choose some special sequence of key cycles and interpolate all other cycles. For example, if an audio sample represents a plucked string sound, it may be that in the attack phase there are many abrupt changes and it is better not to do cycle interpolation, but in the middle or tail of the sample it is increasingly more appropriate.

Spline Modeling Project 2 - Regular Cycle Interpolation

Write a program which incorporates a B-spline model for a short wav file as described above, and forms a UI with options to play back the input wav file or the output wav file computed from the model. User can then recompute the model with changes in parameters k, d, and f_0. User can also opt to do data reduction by selecting every m^{th} cycle, called key cycles*, to leave intact, and then to interpolate B-spline coefficients to compute the model values in the other cycles.*

1. *s = the sample rate of the input file*
2. *f_0 = guess at the fundamental frequency of the audio sample*
3. *k = the number of subintervals per cycle*
4. *d = the degree of the splines (default $d = 3$)*

Output should be a wav file of the same length as input, with values determined from the spline functions constructed to interpolate the original data. Data should be broken up into cycles based on zero crossings closest to periods, using *guess at fundamental frequency f_0. Optionally, the B-spline coefficients making up the model could be written to a text file, one cycle per line. Since the entire model can be computed from the set of key cycles, the amount of data saved as a percent of the original data should be calculated and displayed in the UI with a label such as "reduced data size".*

Spline Modeling Project 3 - Irregular Cycle Interpolation

Use the previous project as starting point. This project allows the user to choose particular cycles as key cycles through a UI, and to graph the model. A custom view with graphs should be implemented, with the following:

1. *graph of audio signal as piecewise linear spline*
2. *interpolation points, marked on time axis and on audio graph*
3. *graph of model as cubic interpolating spline*
4. *highlight cycles currently selected as key cycles*
5. *toggle display on/off for any of the above*

Further Details:

- *View of signal should scroll sideways and zoom in or out.*
- *Mark interpolation points with a different shape or color to distinguish from sample points*
- *Include a button to mark the cycles which are multiples of m, say C_{mj} for integers j, as key cycles (not to be interpolated).*
- *Include a button to mark currently selected cycle as target for interpolation or as a key cycle. This way the model can be forced to use any sequence of key or interpolated cycles.*

Spline Modeling Project 4 - Detailed Model Adjustments

Use the previous project as starting point. Now toggle the interpolation points on or off. Since the interpolation points are not necessarily sample points, but are computed by linear interpolation between samples, they could be graphed with a different color or point shape to distinguish. Also, a UI toggle or button should be able to set whether cycle interpolation is in effect or not. Add the following abilities for the UI:

1. *to delete an interpolation point*
2. *to add a new interpolation point*
3. *to move an interpolation point left/right in the timeline*
4. *to move an interpolation point inside a given cycle up or down*

Further Details:

- *Deleting an interpolation point: this implies a change in the spline basis. The simplest way to adjust the bases is to delete the knot value coinciding with the data input value x_j, for some j with $2 \leq j \leq k-2$, and then renumber the knot values and input values. Additionally, one could implement the selection and deletion of a sequence of adjacent interpolation points.*
- *Adding a new interpolation point, by clicking on the timeline, requires insertion of a new knot value in the knot sequence, which in turn forces a new basis function to be used.*
- *Moving an interpolation point left/right in the timeline: this is best to do by clicking on the time axis, near enough to the t coordinate of an interpolation point to flag its selection, then dragging it left and right. This should move the interpolation point along the piecewise linear graph of the audio data. The point should move up to some minimum distance to neighboring data points, but not over them.*

- *Moving an interpolation point up/down: this moves the y-coordinate only of an interpolation point off the piecewise linear graph connecting the audio data. If this action is followed by then moving this point left/right in the timeline, the point should move with constant y-value until it comes in contact with the piecewise linear graph, after which it merges again with this graph.*
- *If cycle interpolation is in effect, the first two of the above changes (deletion and left/right movement) to interpolation points should also be mirrored in subsequent cycles, following the cycle in which the change is being made. Changing the y-value of an interpolation point would then only affect the current cycle.*
- *A further control option could be given in the UI to select cycles which must maintain consistent changes to y-values. For example, let's say that cycles* 10*,* 11*, and* 13 *are flagged as being consistent. Then when a y-value is changed in one of those cycles, it is also changed in the others. The amount of change could be calculated as a percent increase toward* 1.0*. So if a y-value is changed from* 0.5 *to* 0.6 *in cycle* 11*, (a* 20% *increase toward* 1.0*) and the corresponding value is* 0.8 *in cycle* 10*, then the new value in cycle* 10 *should be* 0.84 *since this matches* 20% *of the distance to* 1.0 *as in cycle* 11*. The UI should perform changes in consistent cycles simultaneously as they are shifted in one cycle.*
- *As changes are made to the data, the model is not updated. When the user wants to update the model using all of the current data, they should click a button to update. This should recalculate the entire model, also doing cycle interpolation where indicated, and graph it. The new model graph is a B-spline graph on each cycle, passing through the required interpolation points.*
- *Data should be forced to remain consistent. For example, if interpolation between cycles is required, the beginning and ending cycles should have the same value for k, or number of subintervals. Coping with inconsistent data can be handled by giving error messages and not computing the model, or by filling in with various types of data. For example, if cycle* 10 *has* $k = 20$ *and cycle* 15 *has* $k = 25$*, and cycle interpolation is selected between these two cycles, then since this in not possible, there could be an automatic filling of cycles* 11 *through* 14 *by using the same B-spline coefficients as for cycle* 10*.*

With these detailed model adjustments, we are departing from the original data more significantly. This could allow a type of synthesis, generating new sounds which are inspired by the original audio data set, but not bound to approximate it. We are also allowing for cycle interpolation which can change several times during the progression of cycles, allowing for basis changes, and sporadic changes in target values which differ from original audio data.

Adjustments to the Spline Bases and Knot Sequences

Suppose a knot sequence $\mathbf{t}$ and interpolation points x_i are chosen, as in the section on Bases for Splines, given as $\mathbf{t} = \{t_0, t_1, \ldots, t_N\}$, with x_i given in the list:

$$x_0 = u_0, x_1 = u_0 + \frac{1}{2}w, x_2 = u_1, x_3 = u_2, \ldots, x_{n-3} = u_{k-1}, x_{n-2} = u_k - \frac{1}{2}w, x_{n-1} = x_k.$$

The knot sequence is also structured to be uniform, so that $t_j - t_{j-1} = w$, $j = 1 \ldots, N-1$. Further, the middle values coincide with the subinterval endpoints, so that:

$$t_d = u_0, t_{d+1} = u_1, \ldots, t_{d+k} = u_k.$$

If an interpolation point is deleted, this can be compensated for by reducing the spline basis by removing one basis function. If the interpolation point is $x_i = u_j$ for some j, with $a \leq j \leq k-1$, then we should remove the basis function $(t - u_j)_+^{d-1}$, which is equivalent to removing the knot value $t_{j+d} = u_j$. The knot sequence then becomes nonuniform, with one gap of length $2w$. One then can check that the interpolation points still satisfy the Schoenberg-Whitney conditions, which in turn guarantees that the new interpolation conditions and new spline basis give a unique solution. So we propose that removal of data points should be restricted to x_j with $2 \leq j \leq k-2$. This avoids the first two data points at the ends of the sequence. It is natural to avoid the end points, since when we are modeling cycles we are assuming zero crossings at the end points. It is also natural to avoid the data points positioned at $\frac{1}{2}w$ from either endpoint, in particular because they do not have an associated basis function, which makes the choice of basis modification more complicated. Also those two points can be considered as data which helps to match the slopes of the signal model near the cycle endpoints.

When the spline model has changes to parameters, such as k, from cycle to cycle, these changes need to be stored in some metadata or header information to the model. Such header information could be, for instance: $M =$ number of cycles, $d =$ degree of splines, then a vector of length M of integers k_i giving the parameter k on each cycle. A different flag could also be set to indicate that all k values are the same, or to give some simpler pattern such as the difference in k_i from the previous cycle, etc.

Data Reduction based on Approximate Curvature

We begin this section with the assumption that we have a spline model of an audio sample, as in project 1. This means we have chosen k, and if we assume also that $d = 3$, then we have $k+3$ interpolation points per cycle. Focusing for the moment on one cycle, we would like to reduce the number of interpolation points based on the notion that points with small local curvature are less essential in forming a good model. To do this, we use an approximation method to detect small values of local curvature on the piecewise linear path of the audio data, as follows.

Let (t, y) be an interpolation point with nearby sample values (i, y_i), $(i+1, y_{i+1})$, $(i+2, y_{i+2})$, $(i+3, y_{i+3})$, satisfying: $i + 1 \leq t < i + 2$. Let $m_j = y_{j+1} - y_j$ be the slope of line segment starting at sample point (j, y_j). Let M_t be the maximum of the absolute differences $|m_{i+1} - m_i|$, $|m_{i+2} - m_{i+1}|$, and $|m_{i+1} - m_i|$. Clearly M_t is constant on line segments of the piecewise linear graph of the audio data. We will say that the interpolation point (t, y) has approximate curvature at most ϵ if $M_t < \epsilon$.

One way to reduce the model data approximating one cycle is to choose a small ϵ and remove all interpolation points with $M_t < \epsilon$. This leaves the question of how many interpolation points will remain. If the goal is to keep this number consistent across cycles, then it may be better to remove interpolation points one at a time, in order of size of M_t. This way it is possible to keep k constant across cycles.

Since we are attempting to make the various schemes (for forming interpolating spline models of audio data) work together, we should address how this type of data reduction could fit in with cycle interpolation. The simplest obstacle to cycle interpolation is changing values of k from one cycle to another. Suppose we keep k consistent by the method indicated above. The next obstacle is changing the distribution of subintervals.

Suppose we have two key cycles with B-spline models based on the same k and $d = 3$. Also assume that the subintervals are not the same, so we have in the first cycle $u_0 < u_1 < \cdots < u_k$ and in the second cycle $u'_0 < u'_1 < \cdots < u'_k$. We can normalize both intervals to assume $u_0 = u'_0 = 0$ and $u_1 = u'_1 = 1$, but the other values can still be different. Suppose also that we want to do cycle interpolation for 10 intermediate cycles between these two. Then we can once again invoke spline interpolation, this time to determine the subinterval points. If there are only two cycles to interpolate between, then we would use linear interpolation between u_i and u'_i to determine the 10 intermediate values. If there are more key cycles, with intermediate cycles to be interpolated, we can forma a cubic spline through the key cycles for each u_i. This can be done in the same manner as for B-spline coefficients.

Introducing Randomness into the Spline Models

Since these spline models for audio data contain many parameters, it is possible to allow some parameters to be controlled by random processes. For example, the lengths of cycles can be fixed to a constant value $L = 1/f_0$, for some fundamental frequency f_0, or can be varied according to the lengths computed from actual audio data. Or they can be chosen randomly, say $L + \epsilon$, for some fundamental frequency f_0 and small error (random variable) ϵ according to some chosen distribution.

A list of parameters that can be randomized:

1. cycle length (as above)
2. key cycle indices (which cycles are to be used as key cycles)
3. subinterval lengths (values $u_0, \ldots, u_k$ on a specific cycle or all cycles)

4. B-spline coefficients for cycle interpolation
5. B-spline coefficients for model output on one or many cycles

In the last case, the model is changed quite drastically if many or all of the coefficients are randomized. As the number of B-spline coefficients increases for the model of one cycle, this would approach random noise, which is the result of randomizing all samples on one cycle. The amount of randomness can be controlled in many different ways. For example, suppose that there 10 B-spline coefficients per cycle, and the model has f_0 which yields 100 samples per cycle. Then suppose that B-spline coefficients b_2, b_4 and b_8 are randomized to be within 0.1 of the original value that was assigned by the model of some audio data. How much will this affect the resulting sound and in what way?

In the previous case, the B-spline is a "meta-spline" affecting the progression of model coefficients between cycles. For example, suppose that every tenth cycle is a key cycle and k (number of subintervals) is constant over the entire model, which has 1000 cycles. One can then use a piecewise linear spline between cycles to linearly interpolate a model coefficient, say b_i, over the all non-key cycles. Or, a smooth spline could be used, say a C^2 cubic spline, which computes the model coefficients in between cycles. What happens if we begin to randomize these meta-splines? The first thing to note is that the key cycles will no longer be maintained, for any such model coefficient. If the meta-spline coefficients are modulated, or allowed to deviate from their original values by some small amount, how does this affect the sound of the entire model?

Linearlity and Periodicity in Spline Models

In the first prototype model above, we computed the lengths of cycles based on real audio data. These lengths can vary due to nonlinearity of the sound generators, such as plucked string, vibrating plates, etc. Nonlinearlity here means that the physics of motion for the vibrating objects is governed by nonlinear partial differential equations, and hence has solutions which can be nonperiodic. This happens, for example, if a string is plucked more vigorously, increasing the error in the small angle approximation used to derive the wave equation. For a guitar string playing $A440$ (first string fifth fret) recorded at sample rate 44100 has around 100 samples per cycle. This number can vary up to around 10 samples per cycle if the string is plucked more vigorously, as opposed to a very soft pluck yielding almost constant cycle length.

A question for modeling comes up naturally: Can we hear the difference in the above nonlinearlity? For example, if we create a cubic spline model based on constant cycle length, and compare this to the model with slightly varying cycle lengths as originally obtained from an audio recording, how does this affect the overall sound?

Spline Models of Musical Gestures

A musical gesture can be thought of in many ways, but some common examples involving pitch, rhythm, and timbre are:

1. a phrase or melody made up of notes from a scale
2. a glissando (ascending or descending pitch from a starting note)
3. vibrato (pitch modulation around a starting note)
4. a percussive sound
5. a rhythmic sequence (independent of pitch)
6. a change in timbre (for emphasis)
7. a smoothly varying pitch based on some curve

Model Hierarchy and Level of Detail for Audio

Model hierarchy refers to the idea of having a sequence of models which use different amounts of data but represent the same audio sample. This could be useful, for example in game audio, in situations where there are many sounds of various different priorities being rendered simultaneously. If the low priority sounds can be represented with smaller amounts of data, or lower LOD (Level Of Detail), it could save memory or processing. The goal would be to have sounds which are still useful, not too distorted or noisy, and still bear enough resemblance to the original sound to be adequate in a mix.

If the models of sounds at different layers in the hierarchy can fit consistently together, then it could also be possible to mix sounds from different layers prior to rendering. Mixing prior to rendering means simply to add the B-spline coefficients of several models of sounds together before producing the model of the joint sound, or mix. Usual mixing is *post-rendered* meaning that samples from two audio streams are added together (and normalized or averaged). So we can also use the term *pre-rendered* mix to refer to a model which incorporates two other models which have been added together (or averaged) by adding their B-spline coefficients. Further, if the B-spline coefficients are to be determined with cycle interpolation, or other techniques, then these calculations would be done in order to generate the coefficient values prior to adding them.

Meta-spline coefficients can also be added (and normalized or averaged) prior to producing B-spline coefficients of two signals which are to be mixed. For example, suppose two signals each have 100 cycles, $C_0, \ldots, C_{99}$, and each cycle has 30 B-spline coefficients. Suppose also that only every $10th$ cycle is a key cycle, and all others are interpolated, and that the interpolation is done for each B-spline coefficient with 10 key coefficients. If the first signal has B-spline coefficients $b_{i,j}$, and the second signal has B-spline coefficients $b'_{i,j}$, $i = 0, \ldots, 29$, $j = 0, \ldots, 99$, then we can use a meta-spline $f_i(t)$ which interpolates the coefficients $b_{i,j}$, for fixed i and another meta-spline $g_i(t)$ which interpolates the coefficients $b'_{i,j}$, also for fixed i. This means that each of f_i and g_i will have 10 B-spline coefficients, let's call them c_m and c'_m, $m = 1, \ldots, 10$. So, for instance, $f_0(t)$ would have 10 evenly spaced interpolation points with target output values $b_{0,j}$, $j = 0, 10, 20, \ldots, 90$, which can be used to fill in the values of all the other $b_{0,j}$. Since the addition of f_i and g_i is simply given by the addition of c_m and c'_m, this can be done prior to computing the b_i which in turn can be done prior to computing the final mixed audio stream.

What does it mean to add B-spline models together from different layers? For this to work, we need to have the B-spline basis for one layer be a subset of the B-spline basis for another layer. This can be accomplished by using refinements of subinterval sequences. For example, if model A has subinterval sequence $u_0, \ldots, u_k$, and model C has subinterval sequence obtained from this one by eliminating every other value, say $u_0, u_2, \ldots, u_{k-2}, u_k$, then B-splines used for model C are a subset of the functions formed by the B-splines formed with model A. So it makes sense to form a sum which has the same B-spline basis as model A. This can also apply to a sequence of layers, as long as the bases from one layer to the next follow the same containment, which can be guaranteed if the subinterval sequences are formed by a systematic deletion of points.

Which Sound Types fit with Which Model Types?

Here are some sound types which are good examples for exploration of the techniques above:

1. Harmonic Tones (instrument samples)
2. Human Speech (varying pitch, more complicated models)
3. Car Engines
4. Footsteps on different surface textures

Filters Applied to B-spline Models

Since B-spline models are used to generage PCM audio data, one can of course apply filters to the output audio stream. However, one can also ask if the model parameters can be filtered, or if they can be modified in such a way as to approximate the effects of various filters.

Network Transmission of Audio Data and B-spline Models

Since audio transmission of spoken voice can be subject to bandwidth fluctuations, it can help to have lower resolution models which allow for reconstruction of a usable approximate model while bandwidth is too low to support full transmission. This requires realtime spline model formation, as well as data transmission protocols to support the reconstruction at the other end.