Probabilistic Audio Resynthesis

Craig Prince, Kevin Wampler and **Katarzyna Wilamowska**
Department of Computer Science and Engineering
University of Washington
Seattle, WA 98195
{cmprince, wampler, kasiaw}@cs.washington.edu

Abstract

We consider the problem of synthesizing an audio signal given an incomplete set of features associated with the note to be generated. This is achieved through a system which learns a joint probability distribution over various signal metrics from waveforms and uses this as a prior for generating new waveforms. We experiment with two methods for resynthesizing a waveform given this joint distribution: a greedy approach, and an approach using particle filters. We found that, in general, our approach is able to reproduce the waveform for both audio it has and has not seen previously; however, because of noise introduced through the sampling process the resynthesized audio has a distinctly electronic timbre.

Introduction

Most current techniques for automatic synthesis of music rely on either recording and replaying of musical notes from real instruments or on hand-built waveforms representing the instrument being played. Collecting/building this data is a costly and time-consuming process. Instead, we would like to be able to create a data-driven audio synthesis model for learning exactly what an instrument "sounds" like.

With such a model it would then be possible to recreate any tone and variation that such an instrument was capable of producing. It would even be possible to do things atypical of current synthesizers, such as create new instruments that are a combination of other instruments or to add features to one instrument that are part of another.

With this broader goal in mind, in this work we aim to simply to show that we can resynthesize waveforms with high quality given a set of feature signals corresponding to the waveform. We would do this after having trained on a different set of waveforms and their corresponding feature signals.

We develop a novel approach to audio resynthesis which uses a set of input feature signals to generate a joint probability distribution over all audio signals. We also test several different training sets for building this distribution to determine which settings are optimal for audio resynthesis. Thirdly, we test two different methods of audio resynthesis – one a greedy approach, and one using particle filters – to determine which is best for audio resynthesis in our model.

We begin with a discussion of related work in the field of audio synthesis. The next two sections then continue with a discussion of how we build our kernel model from a set of training signals and how we can use it to build joint probability distributions – used for inference. This is followed by a discussion of how we finally resynthesize a new waveform from a set of input signals. The next section outlines the experiments we conducted to validate our method and discusses our results. Finally, we end with a section discussing future work and conclusions.

Copyright © 2004, Association for Awesome Audio Inference (www.aaai.org). However bad our programs my be, they still sing better than we do.

Related Work

Using computers for the creation of music has long been a goal of computer science. One of the first and more successful attempts at computer synthesis of music was the Musical Instrument Digital Interface (MIDI) (Int 1983). MIDI was designed to provide a digital specification of musical sound. It includes parameters such as instrument, pitch, volume, note, etc. Each of these can be independently adjusted to produce a tone faithful to the actual tone produced by a real instrument. Most MIDI synthesizers today work by taking a set of known instrument tones and performing digital signal transformations to produce the correct output. Unfortunately these synthesizers are limited in that they do no learning – simply producing using digital signal processing algorithms to manipulate a single waveform.

Our work most closely resembles the previous work in the field of audio analogies. Audio analogies are described in (Roads 1985) and (Levitt 1983). Namely given some set of signals A and some audio A', the system tries to learn the correspondence between A and A'. Then when presented with a new set of features B, the goal is to generate some new audio B' in the same "style" as A'. Our work applies newer AI techniques – e.g. particle filters and metric spaces – to try to solve this problem.

More recent work leverages progress made in image analogies and applies it to audio (Simon *et al.*). However, this work is concerned mostly with recomposition of MIDI segments to capture musical style as opposed to individual waveforms themselves and is thus not directly applicable.

Audio Representation

In order to resynthesize an instrument, it is necessary to build a probabilistic model of the behavior of the instrument. In particular, we would like a model which we can train with a correlated version of a score and an audio rendition of it, and then use this model to take new score and compute a corresponding audio version. It is thus necessary to define not only the features of the audio to store, but also which features of the score should be considered. For the time being we shall look only at the general form these features must take, but shall shortly see precisely what features we associate with scores and audio.

It is clear that both the features of the score and the features of the audio are dependent upon time. At any particular time, a note in the score has some frequency, amplitude, etc., and likewise an audio stream has values which vary over time. To account for this structure we represent both the score and the audio as *signals*. We define a signal as a function

$$s : \mathbb{R} \rightarrow \mathbb{M} \tag{1}$$

such that at any time t, $s(t)$ is the value of the signal s at time t. We further allow that a signal may not be defined at some times, and that signals can depend on other signals provided this dependency is acyclic.

There is a rather serious question as to what restrictions should be placed on the values that a signal can take. Our use of signals to represent both the features of the score as well as the features of the audio necessitate a general definition. It is still necessary, however, to force the values that signals can take to have enough structure to allow the resynthesis of audio given a score. It turns out to be sufficient for our purposes to require that the range of each signal be a metric space. That is, for each signal we define a function

$$dist : \mathbb{M} \times \mathbb{M} \rightarrow \mathbb{R}^{+} \tag{2}$$

which serves a measure of similarity between two values of a signal. Thus for a signal s and times t_1 and t_2, $dist\,(s(t1), s(t2))$ gives a positive real number indicating how similar the values of s at t_1 and t_2 are. As $\mathbb{M}$ is a metric space we require $dist$ to satisfy the following properties.

1. $dist\,(a, b) \geq 0$
2. $dist\,(x, y) = 0 \Leftrightarrow x = y$
3. $dist\,(x, y) = dist\,(y, x)$
4. $dist\,(x, z) \leq dist\,(x, y) + dist\,(y, z)$

Essentially these properties impose enough structure on the values that signals can take to allow us to define consistent probability distributions over a set of signals.

A score will be represented by some set of signals. For notational convenience we denote this set by $\mathbf{S}$ and each individual signal of the score as $\mathbf{S}_i$:

$$\mathbf{S} =_{df} \{\mathbf{S}_1, \ldots, \mathbf{S}_n\} \tag{3}$$

Individual values which these signals may take are denoted by s or s_i for values of all signals or of an individual signal respectively. We similarly denote the audio signal by $\mathbf{A}$ and values which it may take as a. We restrict our attention to the resynthesis of a single audio signal, as the interdependence of multiple audio signals (such as exist in stereo recordings) complicates the algorithms involved.

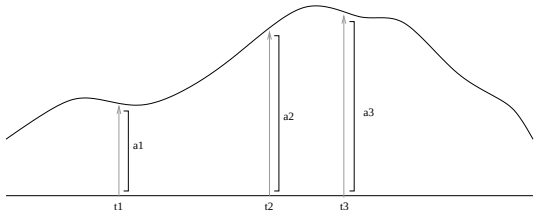

Figure 1: The amplitude of an audio wave taken at different times. The set of all such samples forms the audio signal used in our model.

The task of audio resynthesis can now be stated as the process of, given a collection of signal-audio pairings, taking a set of signals, s, representing an input score and generating the audio signal, a, such that the sequence of values taken by $\{s, a\}$ occurs with high probability in the training data. That is, compute the audio which based on the training data is likely based on the input score.

For simplicity we have chosen a somewhat minimal set of signals to represent a score and a somewhat simplistic audio representation. We assume that each score consists of a single note and that the waveforms generated by the instruments in the audio rendition of this score are approximately homogeneous. That is, there are no large-scale temporal features to the note (such as attack, decay, vibrato, etc.). We also model the audio signal as simply an amplitude of the audio waveform at a given time (Figure 1). We can then give a reasonably full representation of a score by the signals:

frequency the frequency (in hertz) of the note being played

phase the phase of the fundamental frequency of the note being played

lag the audio signal some fixed multiple of the phase in the past

The frequency signal allows the timbre of an instrument to depend on the note being played. The lag signal is an attempt to capture some simple aspects of the evolution of an audio signal over time. The phase signal is critical, even given the frequency signal, and is intended to capture the periodic nature of the audio waveform. Without it, even though the lag signal could induce variation over time, the resulting audio would likely be non-periodic, and thus sound essentially like static. This representation is essentially a metric-space analog with continuous time of the dynamic Bayes net illustrated in Figure 2.

Computing Probabilities

In order to be able to actually compute a likely audio given a set of signals, it is necessary to be able to represent information about the probabilities of the audio and signals. This matter is somewhat complicated by the possibility that some of the signals will not be defined at certain times. This possibility is in fact more than a mere theoretical musing, the lag

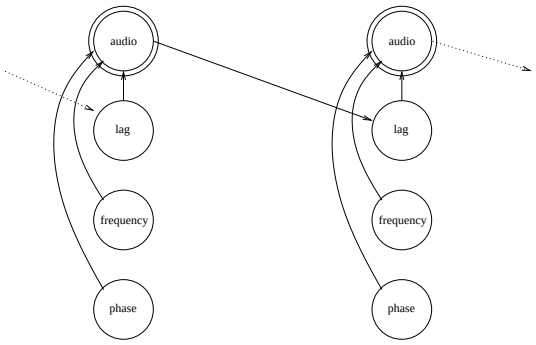

Figure 2: A representation of our signal dependencies as a dynamic Bayes net. Note that this is somewhat misleading, as time is continuous in our model.

signal for example does not have a value before some fraction (which is equal to the amount of the lag) of the period of a note has been synthesized.

We build a model in which it is possible to sample the joint probability distribution $\mathbf{P}(\mathbf{X})$ for any $\mathbf{X} \subseteq \{\mathbf{S} \cup \mathbf{A}\}$. This allows us to sample joint probabilities at points defined by assignments of any subset of our signals (typically all the signals which are defined at a given time). Furthermore since $\mathbf{P}(\mathbf{X}|\mathbf{Y}) = \frac{\mathbf{P}(\mathbf{X},\mathbf{Y})}{\mathbf{P}(\mathbf{Y})}$, we can use this model to sample the conditional probabilities for any set of signals.

To achieve this flexibility of sampling the joint probability distribution for any set of signal values and because the independence of the signals from each other is not known beforehand we have chosen a non-parametric representation to compute these joint probabilities. This representation is a generalization of a kernel model which works in metric spaces and allows sampling along an arbitrary hyperplane defined by assignments to some subset of the full set of axes in the model (as opposed to only allowing sampling at a point defined by assignments to all axes). The model also allows kernel functions to be added for arbitrary axis-aligned hyperplanes, although we do not ever make use of this ability.

The techniques to implement these requirements are surprisingly simple. We define the kernel space as the space of joint probability distributions of values of the signals and the audio.

$$K =_{df} range(\mathbf{A}) \times range(\mathbf{S}_1) \times \cdots \times range(\mathbf{S}_n) \quad (4)$$

We further define points in this kernel space as assignments of values to some subset of the signals:

$$k = \langle s_i, \ldots, s_j \rangle \text{ where} \quad (5)$$

$$s_i \in range(\mathbf{S}_i), \ldots, s_j \in range(\mathbf{S}_j) \text{and} \quad (6)$$

$$\{\mathbf{S}_i, \ldots, \mathbf{S}_j\} \subseteq \mathbf{S} \cup \mathbf{A} \quad (7)$$

Note that due to the requirement that the values of each signal lie in a metric space the distance between two points in the kernel space can be computed along any axis. We can leverage this to create a general definition of a kernel function, f_k, for a point k in the kernel space as a function [1] mapping distances from k along some subset of the axes of K into a real number:

$$f_k : \mathbb{R}^{+m} \rightarrow \mathbb{R}^+ \quad (8)$$

The strength of a kernel at a point p is calculated by computing the distance of k from p along each of the axes they have in common and then evaluating the kernel function for these distances.

The kernel space will consist of many of these kernel functions and the strength of the kernel model at a point is defined to be the sum of the strengths of all of the kernel functions at that point. Provided that the functions are well chosen this kernel space will provide a reasonable estimate for the probability density of the joint distribution for any subset of assignments to $\mathbf{S} \cup \mathbf{A}$ (particularly around dense portions of the kernel space).

For the actual kernel functions used in our model we have chosen multidimensional axis-oriented Gaussian functions. Each such kernel is associated with a weight w, a point μ in K as a mean and a variance in $\mathbb{R}^+$ associated with the range of each signal in this mean. Let us denote these variances by $\sigma_1, \ldots, \sigma_m$ and the distances from μ to a point at which the strength of the kernel is being evaluated, p, by $d_1, \ldots, d_m$. The strength of the kernel at p is then expressed as:

$$\frac{w}{2\pi \prod_{i=1}^{m} \sigma_i} \cdot e^{\sum_{i=1}^{m} \frac{d_i^2}{2\sigma_i^2}} \quad (9)$$

It is useful to note that such a function will always have a single maximum at μ and the integral of its strength along any set of axes will always be w.

As a final note, we mention that given the loose structure of the kernel space it can be a non-trivial problem to find points to sample it at. For signals which are known to take values from the real numbers, for example, this is straightforward, but in general this is not always possible. Note however that at the very least it is possible to sample the space at points generated from combinations of the coordinates of the points used to generate the kernel space itself. In most cases, however, it is possible to compute convex combinations of points in the kernel space. In this case we can represent new sampling points by convex combinations of preexisting points in the kernel space. This limits the possible points that can be expressed to the convex hull of the set of sample points, but in general this is a reasonable restriction as the single maximum of the kernel functions ensures that all maxima in the probability distribution will occur within this hull.

Resynthesizing

After constructing the kernel space from the training signals, the next step is to recreate a likely output signal given only a set of input signals describing this output signal. We call this process resynthesis. One common scenario for resynthesis is to recreate some output waveform given a set of

[1] Note that this is actually a template for many functions, one for each possible subset of the axes of K rather than a single function.

signals describing the features of this waveform; however, our approach will work for regenerating any missing signal given some set of other related signals. Although our goal can be any signal, for convenience throughout this section we will refer to the signal being reconstructed as the audio or waveform.

We have implemented two different algorithms for generating an output waveform from signals. The first is a greedy approach which always chooses the"best" value for each time step. The second approach uses particle filters to take into account the previously generated waveform during resynthesis.

Greedy Resynthesis

Our first method for resynthesis is to use a greedy algorithm to give a good approximation of the waveform we are trying to construct. The basic idea of this approach is to select the next value of our waveform by taking a weighted average of all the possible candidate values of the waveform (though could be a discrete or continuous space). In this way, our greedy approach is similar to the Bayesian estimator in that it finds the value that gives equal weight to both sides of the probability distribution of candidate values. The probability estimate is calculated by sampling the kernel space probability distribution as described above.

The value calculated by weighted averaging becomes the value of the waveform at this time step. For each new time step, we simply calculate the new waveform value as described above.

As one might imagine, this approach seems somewhat flawed by taking a weighted averaging across all samples as this can overly smooth the resulting audio. We believe a better approach would be to find the most likely sequence of waveform values for a sequence of input signals instead of averaging.

Resynthesis Using Particle Filters

In the greedy resynthesis method above our approach to reconstruction took a weighted average over the distribution to calculate new waveform values. However, this can introduce smoothing, to see how this affected the results we also implemented a different algorithm for resynthesizing the output waveform using particle filters. This approach allows us to approximate the best sequence of output audio amplitudes given a set of input signals.

The optimal approach to generating the samples of a waveform would be to use a Viterbi-style algorithm because it gives the optimal sequence of "states" (waveform amplitudes) given a sequence of observations (input signals); however, the Viterbi algorithm only works over a discrete set of states yet we are interested in finding the optimal values over continuous metric spaces. Using particle filters allows us to get a good stochastic approximation to Viterbi for a continuous space.

Particle Filter Implementation Our particle filter implementation takes a very standard approach as described by Russell and Norvig (Russell & Norvig 2003). Each particle represents a belief in a certain audio waveform amplitude at a given time. We then update each particle's position based on the probability distribution:

$$\mathbf{P}\left(X_{t+1}|X_t\right) \tag{10}$$

Where X_{t+1} is the new waveform amplitude and X_t is the current waveform amplitude. This distribution is exactly the transition probability function for each particle. Notice that because we are in a continuous space, this probability distribution is different for every particle and must be recalculated each time step. We are able to get these probabilities by sampling the Kernel space produced above (see *Computing Probabilities* above).

After updating the particle's "position" we calculate the likelihood of each particle based on the probability that the given set of signals would support that particle's current value:

$$\mathbf{P}\left(X_{t+1}|Obs_{Signals}\right) \tag{11}$$

Where X_{t+1} is our new amplitude and $Obs_{Signals}$ are the input signals at $t+1$, i.e. our observations. Note that this is somewhat contrary to the general particle filtering algorithm, which reverses the conditional probability so that it is:

$$\mathbf{P}\left(Obs_{Signals}|X_{t+1}\right) \tag{12}$$

Because we are unable to precisely calculate this quantity, we make the assumption that $\mathbf{P}\left(Obs_{Signals}\right)$ is uniform and thus the conditional probabilities in equations 11 and 12 are proportional. Now, given the weighted particles we take the value of the particle with the highest weight and use it as the value for the resynthesized waveform at time $t+1$.

Once we've weighted the particles we then need to remove low probability particles and replicate high-probability particles. This is accomplished by resampling the particles. We draw a new set of particles from the existing set, but use the normalized weights as the probabilities by which we draw each particle. Note that particles are drawn with replacement so that the higher-weighted particles will be drawn more often. We perform this sampling after each time step, keeping the total number of particles constant and resetting all weights to zero.

We generate probability distributions by uniformly sampling the kernel space over a given multi-dimensional hyperplane. We then linearly interpolate these samples to generate a smooth, complete probability density function (PDF). Given a PDF we are then able to directly draw samples from this distribution with the probabilities defined in the PDF. We take a shortcut when drawing samples so that instead of transforming the PDF into a cumulative density function (CDF) and taking it's inverse, we can instead calculate the value of a sample directly from the PDF. This is simply a convenience and does not affect our results.

Results

Testing Methodology

The problem of comparing waves is difficult since ideally the comparisons would be qualitative instead of quantitative in nature. In our attempt to generate some methods of comparisons between waves we relied on two different distance

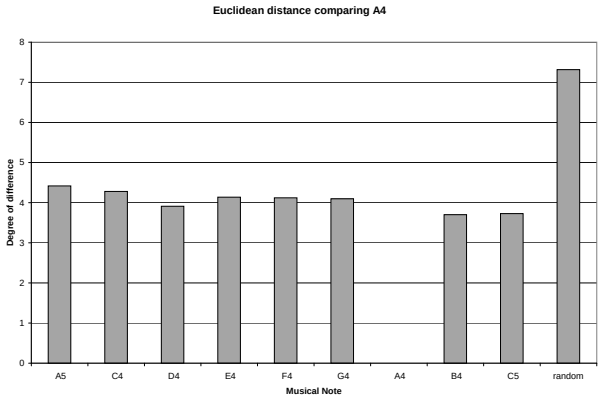

Figure 3: Euclidean distance: difference values between waves created by notes A_4, the $C_4 \rightarrow C_5$ octave, A_5, and a randomly generated wave.

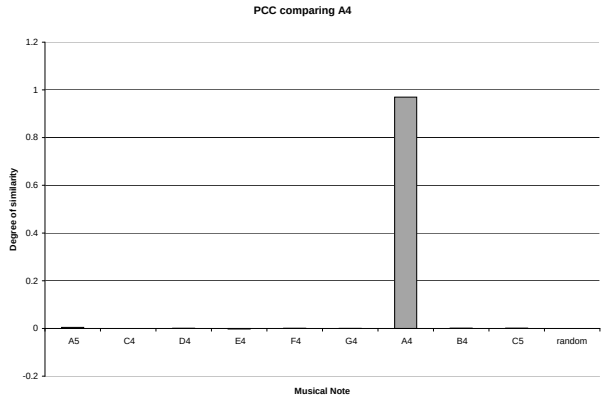

Figure 4: Pearson's correlation coefficient: similarity values between waves created by notes A_4, the $C_4 \rightarrow C_5$ octave, A_5, and a randomly generated wave.

measures: *Euclidean distance* and *Pearson's correlation coefficient*. Initially each wave represented in a WAV file is translated to an array of sampling values. These arrays are then used in the comparison process. Our experiments involved determining:

1. how close we could replicate a wave that had already been learned
2. whether we could infer new/untrained waves based on learned waves
3. whether learning more samples allowed us to create a more likely wave

In the following section we will answer these questions and explain some of the calculations necessary to determine wave distance/correlation.

Euclidean distance The Euclidean distance between two waves w_1 and w_2 is calculated as follows.

$$d_e(w_1, w_2) = avg\sqrt{\sum_{i=1}^{n}(w_{1i} - w_{2i})2} \tag{13}$$

where, the average distance value d_e is calculated over the possible horizontal shift between the two waves. The closer the difference value is to zero, the more identical the waves. Figure 3 shows a comparison between note A_4, the $C_4 \rightarrow C_5$, A_5 and a randomly generated wave. Obviously, A_4i s most similar to note A_4, while least similar to the randomly generated note.

Pearson's correlation coefficient The Pearson's correlation coefficient between two waves w_1 and w_2 is calculated as follows.

$$a = \sum_{i=1}^{n} w_{1i} \tag{14}$$

$$b = \sum_{i=1}^{n} w_{2i} \tag{15}$$

$$c = \sum_{i=1}^{n} (w_{1i} w_{2i}) \tag{16}$$

$$d = \sum_{i=1}^{n} {w_{1i}}^2 \tag{17}$$

$$e = \sum_{i=1}^{n} {w_{1i}}^2 \tag{18}$$

$$d_{pcc}(w_1, w_2) = \frac{c - \frac{(a)(b)}{n}}{\sqrt{(d - \frac{a2}{n})(e - \frac{b2}{n})}} \tag{19}$$

With this method of measurement, the higher values correspond to better matches between waves. In Figure 4, outside of note A_4, no matches are visible. By these two initial experiments with wave files not produced by our system, we are convinced that the measurement techniques chosen for our wave comparison purposes will be sufficient.

Inferring learned and unlearned notes

If we were to store the training data samples in a look-up table and then attempt to resynthesize a wave with identical frequency, the values returned would be just the memorized value of the initial training wave. However, our system, using machine learning techniques, infers the new wave based on all the values it has trained on, with added weight to samples with similar frequencies. For example the note C_3 was

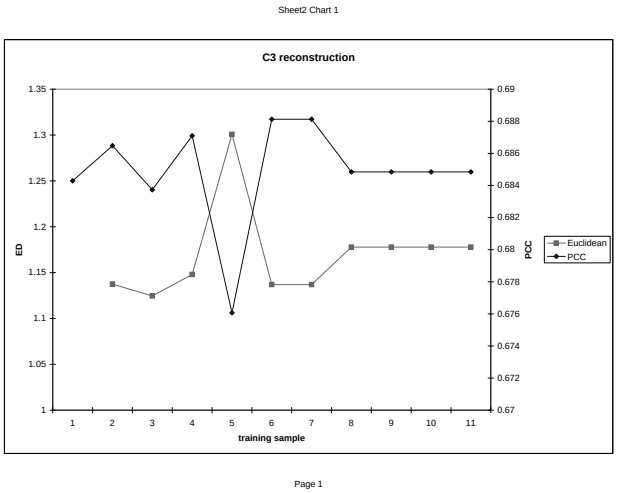

Figure 5: Reconstruction of note C3, based on training sample size. C3 is always included in the training set.

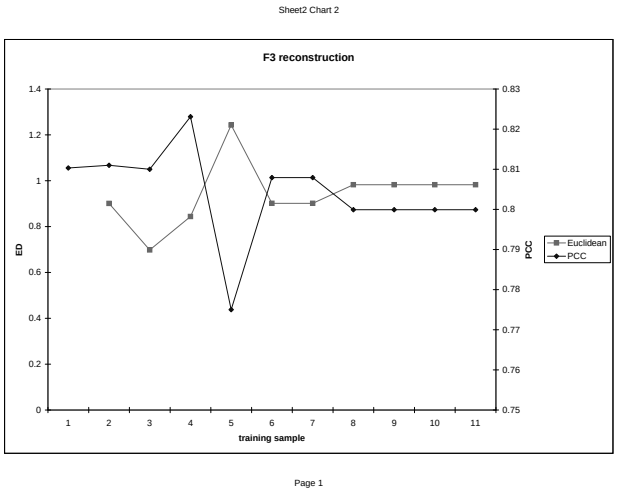

Figure 6: Reconstruction of note F3, based on training sample size. F3 is not in the training set until training sample sizes 6 and above.

used in all the training samples and in Figure 5 we can see that although known at the beginning the distance between the original wave and the generated wave decreases as the sample size increases. In comparison, note $F3$ did not enter the training sample until a sample size of 6, and we can observe that the distances are much larger, Figure 6. The distances for either note are still quite informative as to our success to resynthesize both a training and non-training sample note.

Training sample size

As can already be deduced from Figure 5 and 6, as the training sample size increases, the distance between two compared waves decreases. In order to obtain a more general view, the averages over all training samples were calculated and presented in Figure 7. As the sample size increases, the Euclidean distance between two waves shrinks and the Pearson's correlation coefficient grows.

Different Methods of Audio Resynthesis

As mentioned we tested both a greedy resynthesis method and an approach based on particle filters. Due to the prohibitive computational time associated with the particle filtering approach we have not been able to generate a large set of results for it. Our preliminary results indicate that it gives a similar result to the greedy method, but with a low number of particles introduces a significant amount of noise. This can be combatted with the use of more particles, though at the cost of computational time.

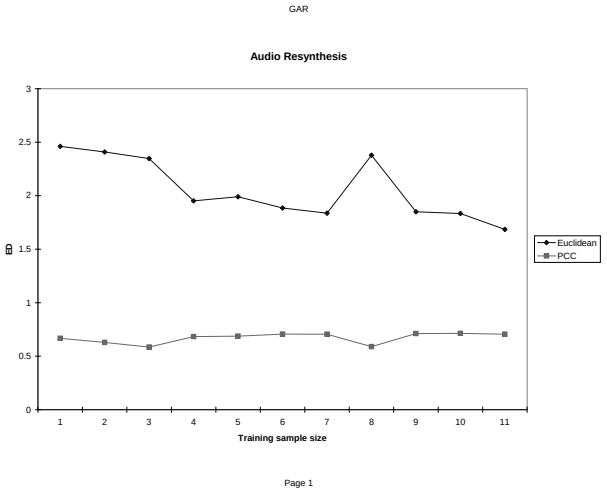

Figure 7: Average distances and correlation coefficients for all notes based on unchanging set of training samples in any given size.

Conclusion and Future Work

As is illustrated by the testing data and a qualitative side-by-side comparison of a audio signal and a resynthesized version of it (Figure 8), the method described in this paper is an effective way of reproducing the wavefrom of an audio

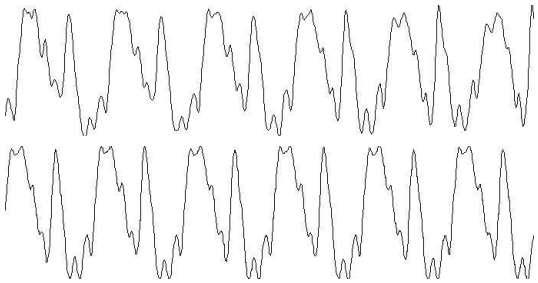

Figure 8: Original violin waveform (top) and a resynthesized version (bottom).

signal. Furthermore, because of the general nature of the algorithms employed these methods can likely be applied to a wide range of other phenomenon beyond audio resynthesis. Despite these successes, the method still falls short of effective audio resythesis as the actual tones generated have a distinctly electronic timbre. We speculate that this is due to a small amount of high frequency noise added by a slight jittering created by the sampling of the kernel model used to resynthesize the signal.

As audio synthesis in general is an extremely complex problem, there is a great deal of work which remains to be done along the lines of resynthesis. This work lies in both the quality of the resynthesis and in data acquisition. With regard to the former of these issues it would likely be very useful to represent an audio signal as a series of very short audio clips (similar to granular synthesis) instead of single samples. This would help to reduce the effect of jitter introduced by the approximate nature of the resynthesis, as well as to better express longer term evolution of an instrument and possibly greatly improve the speed of resynthesis by computing the audio samples more sparsely. In addition it may be fruitful to represent these samples in a basis other than the waveshape which has a more direct correspondence with the perception of the sound, such as a frequency spectrum. Note that since we only require that the values of signals lie in a metric space this extension should work well with the model presented in this paper.

With respect to data acquisition there is a great deal of work that can be done. Most obvious is that scores are currently limited to single notes. Ideally a score would be any combination of potentially multiple instruments playing multiple notes at different times. The learning of the sounds of individual instruments from such a combination is a very tricky task, but given that we have an explicit probabilistic model it may be possible to use an EM type algorithm to infer sound of the individual notes.

References

[1] The International MIDI Association (IMA), Sun Valley, California. 1983. *Musical Instrmument Digital Interface (MIDI) Specification 1.0*.

[2] Levitt, D. 1983. Learning music by imitating. Unpublished manuscript.

[3] Roads, C. 1985. *Research in music and artificial intelligence*, volume 17. ACM Press.

[4] Russell, S., and Norvig, P. 2003. *Artificial Intelligence: A Modern Approach*. Prentice-Hall, Englewood Cliffs, NJ, 2nd edition edition.

[5] Simon, I.; Basu, S.; Agarwala, M.; and Salesin, D. Audio analogies. Unpublished manuscript, University of Washington.

Appendix A

Work done by Craig Prince

- particle filter resynthesis

Work done by Kevin Wampler

- conceptual framework
- code for audio I/O, kernel model, and greedy resynthesis

Work done by Kasia Wilamowska

- generation of test tones
- testing of algorithms

Work done by all authors

- suffering through a very long Sunday night/Monday morning