AUDIO ANALOGIES: CREATING NEW MUSIC FROM AN EXISTING PERFORMANCE BY CONCATENATIVE SYNTHESIS

Ian Simon
University of Washington

Sumit Basu
Microsoft Research

David Salesin
Microsoft Research

Maneesh Agrawala
Microsoft Research

ABSTRACT

This paper describes a method for creating new music by concatenative synthesis. Given a MIDI score and an audio recording of an example piece of monophonic music, our method synthesizes audio to correspond with a new MIDI score. The algorithm we use is based on concatenative synthesis, commonly used for generating speech. Two versions of our algorithm are explored, one in which individual notes from the example piece are concatenated, and one in which pairs of adjacent notes from the example piece are concatenated. We examine the range of example pieces and target scores for which each version of our algorithm yields good results. Our underlying framework remains general enough to be applicable to other problems, such as rendering a stylized version of the target score, and other types of sound analogies.

1. INTRODUCTION

Techniques for synthesizing sound can be split into two categories of approaches. *Model-based synthesis* uses a recipe for creating sound from scratch, and generates new waveforms with different qualities by modifying the parameters in the recipe. *Concatenative synthesis* instead uses a database of existing sound, divided into units, and generates new waveforms by placing these units in a new sequence.

Our system, which we call Audio Analogies (Figure 1), provides a convenient method for controlling the creation of new sounds. Given an example pair (A, A') specifying the desired creation mechanism, we apply this mechanism to new input B to produce output B'. Although the framework is general enough to handle a wide variety of operations, we have currently only explored the case in which A and B are MIDI scores, and A' is a raw sound clip. Here, A is a transcription of musical performance A' (see Figure 2), and our goal is to create a new raw sound clip B' that is the realization of MIDI score B.

The above operation can be used for the following two applications, of which we focus almost entirely on the first one:

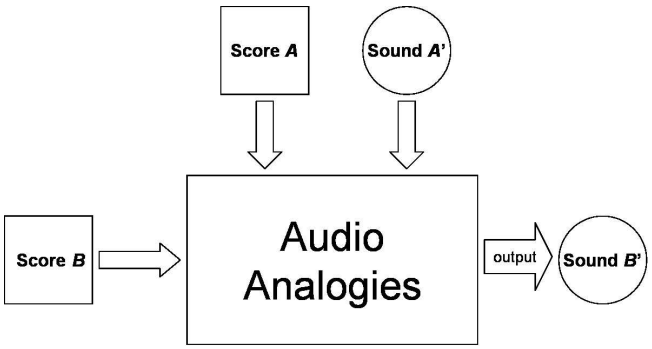

Figure 1. *Our system takes MIDI scores A and B and raw sound A' as input, and produces a new raw sound B', such that the relationship between A and A' is "the same as" the relationship between B and B'.*

- *improved music synthesis* — Music synthesized by analogy can have a more human feel than music created by a standard MIDI synthesizer, as any nuances of example sound A' can be preserved in B', without the use of complicated sound models. Audio analogies allow us to play the notes of MIDI score B exactly, but with more natural instruments. We focus on this application for this paper.

- *music stylization* — Instead of playing score B exactly, we can attempt to preserve more of the musical structure of A by using note sequences that are similar to those present in A. Rather than realizing a MIDI score on a particular instrument, we can realize a modification of that MIDI score that reflects the phrasing of a certain musical style or performer.

In the next section, we describe the problem of computing analogies between musical scores and waveforms. In Section 3, we discuss related work, in the areas of both analogies and audio synthesis. In Section 4, we give some intuition behind our algorithm, and in Section 5, we present two versions of the algorithm itself. In Section 6, we provide some current results, which can be accessed at `http://www.cs.washington.edu/homes/iansimon/audio_analogies/`. We discuss possible applications of our algorithm in Section 7, and future expansions of our technique in Section 8.

2. PROBLEM

Our problem can be stated as follows:

> Given musical score A, waveform A', and score B, create a new waveform B' such that $A : A' :: B : B'$.

That is, the relationship between A and A' should be "the same as" the relationship between B and B'. In order to evaluate the degree to which this statement holds, we need to decide what it means for the relationship between A and A' to be "the same as" the relationship between B and B'. Such a high-level concept as this depends greatly upon human perception, both in terms of the accuracy of the transformation between B and B', and the coherence of B' with respect to A'. Our approach borrows from concatenative speech synthesis, in which speech is generated from text by concatenating phonemes from a large library of examples.

In this approach, there is a fundamental tradeoff between accuracy and coherence. The more faithful B' is to B, the less likely it is that B' is coherent with respect to A'. The more coherent B' is with respect to A', the less likely it is that B' is an accurate transformation of B. We use a value α, between 0 and 1, to express this tradeoff. Values closer to 1 mean that B' should match B more closely, while values closer to 0 mean that B' should incorporate more of the style of A'. So, at the most general level, the input to Audio Analogies is an example pair (A, A'), a new score B, and parameter α, and the output is a new waveform B'.

Though the analogy framework is general enough to handle different types of sound objects, in this paper we will deal exclusively with the case in which A and B are MIDI scores, and A' and B' are raw sounds. This is the only case we have explored, and though the Audio Analogies system could be expanded to do much more, we feel this case is the most interesting. We discuss some other cases in Section 3. A MIDI score is usually thought of as simple and mechanical, while a musical performance is more human and full of artistic expression. The process by which a score is transformed into a musical performance is therefore intriguing, and providing an automated mechanism for it is a desirable goal.

In addition, we limit ourselves to monophonic music for both A and B.

3. RELATED WORK

Our work is inspired most directly by Image Analogies [3]. However, when adapting their technique to audio, we found it necessary to borrow ideas from the realm of sound synthesis. The concatenative approach has been quite successful in the realm of speech synthesis. In addition, Schwarz [8] used concatenative synthesis to generate sound that was the realization of a musical score, played using sounds from a large database. Our focus is on high-quality synthesis with a single example instrument, often with only a few minutes of example data. We also allow for the modification of database units.

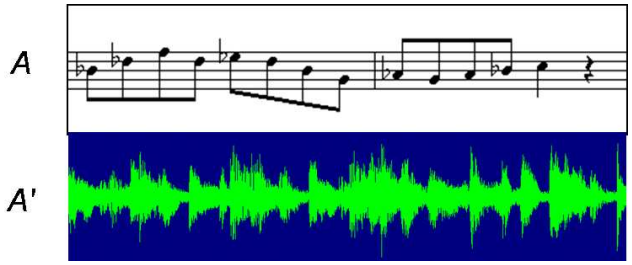

Figure 2. *An example pair (A, A') consists of a MIDI score and a waveform captured from a human performance of the score.*

Zils and Pachet [9] developed a system called Musical Mosaicing, which uses concatenative synthesis to automatically sequence snippets of existing music from a large database to match a target waveform. They focus on generation of music from the techno genre, and designed their system as a composer's tool.

Also, some model-based techniques have explored the notion of musical style. The Saxex system [1] generates expressive performances of melodies from a model derived from examples of human performances. Derenyi and Dannenberg [2] synthesize trumpet performances by using both a performance model, which generates a sequence of amplitudes and frequencies from a score, and an instrument model, which models the sound timbre.

Score alignment is an important component of concatenative music synthesis, since MIDI score A must be as accurate a transcription of performance A' as possible. Orio and Schwarz [5] use dynamic time warping to find the best global alignment of a score and a waveform. Raphael [6] uses a hidden Markov model to segment a waveform into regions corresponding to the notes of a score.

4. OUR APPROACH

Given example pair (A, A') and new score B, we need a technique for creating a new waveform B' that is a performance of B, using sound from A'. Concatenative synthesis has been widely used to solve the corresponding problem in the realm of speech. In this approach, B' is constructed by concatenating pieces of A'.

The analogy framework should be able to capture any local transformation. That is, a *frame* of A' (or B') should only depend on the corresponding frame in A (or B) and its local neighborhood. A frame can be thought of as a basic building block of sound. Our system uses either single notes or pairs of adjacent notes as frames. We construct B' using a sequence of frames from A'. Each frame in the sequence should match the corresponding frame in B, and the sequence should be coherent with respect to A'. Given cost functions for these two objectives, and the value α, the optimal sequence is well-defined, and can be computed with a dynamic programming algorithm.

Also, A and A' need not be a single example pair, but could be a database of many examples. This increases the likelihood that each neighborhood in B will be similar to a neighborhood in A, which yields much better results. However, increasing the quantity of example data also leads to an increase in running time. In our current algorithm, the increase is quadratic. This is because we allow all frames in A' to be possible matches for each frame in B'. Our databases tend to be small, since we are often trying to recreate the style of a single performance.

To better utilize the small amount of example data we have, we allow B' to use modified frames from A'. Using SOLA (synchronized overlap-add) [7], we can independently modify the time and pitch of an example waveform to more closely match a target note in B.

5. AUDIO ANALOGIES

First, we define some terms that will be used in the rest of the paper (Section 5.1). Then, we describe the data structures needed (Section 5.2), followed by the algorithm itself (Section 5.3). We first present the version of the algorithm in which a frame is a single note, and later discuss the alternate version in which a frame is a pair of adjacent notes.

5.1. Definitions

The terms defined in this section represent structures and values necessary for describing our algorithm.

(A, A') is the input example pair. A is a MIDI score, and A' is the corresponding waveform.

B is the input target score.

B' is the output waveform.

$|A|$ is the number of frames that make up A.

a_i is the i^{th} frame of A. a'_i, b_i, and b'_i are defined similarly for A', B, and B', respectively.

z_i^j is the j^{th} candidate for matching frame b_i. A candidate is a frame from A', possibly altered to better match b_i.

$r(i, j)$ is the index of the frame in A' that is used to construct candidate z_i^j. That is, z_i^j is constructed from $a'_{r(i,j)}$.

κ is the number of candidates for each frame b_i.

$c_{\text{match}}(i, j)$ is the cost of matching candidate z_i^j with frame b_i in B. This is the *match cost* of using z_i^j as the i^{th} frame of B', independent of all other frames in B'.

$c_{\text{transition}}(i, j, k)$ is the cost of placing candidate z_{i+1}^k directly after candidate z_i^j in B'. This is the *transition cost* between these two frames.

α is the weight given to match costs as opposed to transition costs, between 0 and 1.

The match and transition cost functions have many components, and will be discussed in Section 5.3.3.

5.2. Data Structures

In addition to the three inputs, A, A', and B, we will need a few intermediate structures in our computation:

- M_{cost} is a $|B|$-by-κ matrix of costs, used for dynamic programming. $M_{\text{cost}}[i, j]$ represents the total cost of the optimal sequence of frames 1 to i of B' in which $b'_i = z_i^j$.
- M_{index} is an n-by-κ matrix of indices, used for dynamic programming. $M_{\text{index}}[i, j]$ holds the index k for which $b'_{i-1} = z_{i-1}^k$ in the optimal sequence of frames 1 to i of B' in which $b'_i = z_i^j$. That is, z_{i-1}^k is the predecessor of z_i^j in such an optimal sequence.

These data structures are used by the dynamic programming algorithm that computes the optimal frame sequence.

5.3. Algorithm

The algorithm can be broken up into six stages:

1. Segment A, A', and B into frames (Section 5.3.1).
2. Choose candidates z_i^j for each frame b_i (Section 5.3.2).
3. Compute c_{match} and $c_{\text{transition}}$ (Section 5.3.3). This presumes the existence of a method for computing match and transition costs between frames, which we will discuss later.

4. Using a Viterbi algorithm, compute the two matrices M_{cost} and M_{index} (Section 5.3.4). From these two matrices, compute the globally optimal sequence S of frames from A'.

5. Construct the waveform for each frame of S (Section 5.3.5).

6. Construct B' by concatenating the waveforms (Section 5.3.6).

All of these stages are important in terms of their effects on the algorithm's output, so we will discuss them in more detail in the following sections.

5.3.1. Segmentation

Each sound must be broken into discrete frames. It is important to distinguish between three different types of frames:

- *score frames* — These are the original frames from A and B. Each one is simply a vector of note properties (see Figure 3), most importantly the duration and pitch.
- *candidate frames* — These are similar to score frames, but are used as potential matches for the score frames of B. Each candidate frame contains a vector of note data, as well as a reference to a score frame in A.
- *wave frames* — This is the type of frame we use when actually constructing B'. It consists of a set of raw sound samples.

In this version of the algorithm, a single frame corresponds to a single note (or rest, which we treat no differently from a note). This leaves us with the problem of making sure that the notes of A align properly with the notes of A'. A near-perfect alignment is necessary, since we will be using the waveform corresponding to a single note of A when constructing the waveform corresponding to a single note of B. If sound data from other notes in A manages to seep in, audible "grace note" artifacts will be heard in the output waveform. (The second version of our algorithm uses an alternate method of addressing this problem.)

Another problem that arises is that sometimes, even in the case of monophonic music, a short amount of time exists during which two adjacent notes can be heard simultaneously. This *coarticulation* can be caused by reverberation in the recording environment, or even inside the instrument. When this scenario occurs, a perfect alignment is not possible.

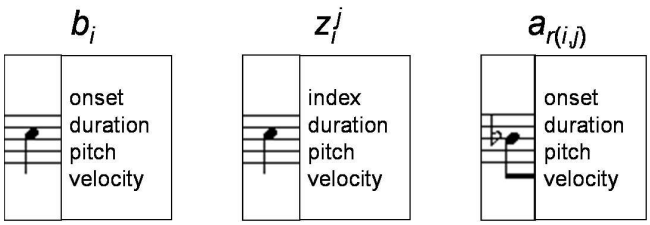

Figure 3. *Each score frame b_i corresponds to a single note. The data associated with the frame is the note onset time (in milliseconds), the note duration (in milliseconds), the pitch, and the velocity (a MIDI parameter representing how hard the note is struck). A candidate note z_i^j contains its duration, pitch, velocity, and a reference to frame $a_{r(i,j)}$, whose corresponding waveform $a'_{r(i,j)}$ will be used to construct this candidate. Frame $a_{r(i,j)}$ may have different note parameters from b_i.*

After the segmentation has been computed, we perform one more preprocessing step in which B is modified to make matches with A more likely. First, we attempt to transpose B so that it is in the same key and pitch register as A. This is as simple as trying all possible transpositions of the notes of B, and keeping the one which has the most pitch overlaps with A.

Second, we change the tempo of B uniformly so that the median note duration of B and A is the same. This isn't necessarily optimal, in the sense that there are score distance metrics for which another uniform tempo change would be better, but this technique seems to work well on our test data.

5.3.2. Candidate Selection

After the input sounds have been segmented, we need to choose the candidates z_i^j for each target frame b_i. We construct z_i^j from note a'_j for all j. That is, we use $\kappa = |A|$ candidates z_i^j for each b_i, and $r(i,j) = j$. We also change the pitch and duration of each candidate to match the pitch and duration of b_i.

As we get larger and larger examples (A, A'), using all example frames as candidates becomes less feasible. For examples that are several minutes long, as ours are, computing the optimal sequence takes only a few minutes. However, for example databases containing hours of music, this approach no longer works in a reasonable amount of time. In such cases, we can choose some predefined value for κ and pick the best κ candidates for each frame, with respect to c_{match}.

5.3.3. Cost Computation

Once we've broken each sound into frames, we need to compute the values of c_{match} and $c_{\text{transition}}$. In order to compute the entries in these matrices, we need a method for evaluating the cost of using SOLA and resampling to change the duration and pitch of a waveform, and a method for evaluating the cost of placing two candidate frames in succession.

Suppose we have defined functions $d_{\text{transform}}(s_1, s_2)$, and $d_{\text{transition}}(s_1, s_2)$ (for score frames s_1 and s_2), representing the cost of transforming one frame to another (using SOLA and resampling to change a note), and placing two frames in succession, respectively. Then, we can compute c_{match} and $c_{\text{transition}}$ as follows:

$$\begin{aligned} c_{\text{match}}(i,j) &= d_{\text{transform}}(a_{r(i,j)}, z_i^j) \\ c_{\text{transition}}(i,j,k) &= d_{\text{transition}}(z_i^j, z_{i+1}^k) \end{aligned}$$

It remains to define $d_{\text{transform}}$ and $d_{\text{transition}}$. Let

$$\begin{aligned} d_{\text{transform}}(a_{r(i,j)}, z_i^j) = \; & \beta |pitch(z_i^j)\log(duration(z_i^j)) \\ & -pitch(a_{r(i,j)})\log(duration(a_{r(i,j)}))| \\ & +\gamma |pitch(z_i^j) - pitch(a_{r(i,j)})| \end{aligned}$$

The first term in the sum is the cost of changing the duration of a note using SOLA and is proportional to the logarithm of the ratio of the durations. (Note that pitch terms are also included, since we change the pitch before applying SOLA.) The second term is the cost of changing the pitch of a note using resampling and is proportional to the difference in pitch (or the logarithm of the ratio of the frequencies). Again, the β and γ terms allow the user to place relative weights on SOLA and resampling.

Let

$$d_{\text{transition}}(z_i^j, z_{i+1}^k) = \begin{cases} 0 & \text{if } r(i+1,k) = r(i,j)+1 \\ & \text{and } pitch(a_{r(i,j)}) - pitch(z_i^j) \\ & = pitch(a_{r(i+1,k)}) - pitch(z_{i+1}^k) \\ \lambda & \text{if } r(i+1,k) = r(i,j)+1 \\ & \text{but } pitch(a_{r(i,j)}) - pitch(z_i^j) \\ & \neq pitch(a_{r(i+1,k)}) - pitch(z_{i+1}^k) \\ \lambda + \mu & \text{if } r(i+1,k) \neq r(i,j)+1 \end{cases}$$

The transition cost is simple. If the two candidates do not come from two consecutive frames of A, a cost $\mu + \lambda$ is incurred. If the two candidates come from consecutive frames but must be resampled at different rates to match the target pitch, a cost λ is incurred. If the two candidates come from consecutive frames and are transposed by the same interval, no cost is incurred. This cost function causes the algorithm to prefer to choose a sequence of frames with as few coarticulation artifacts as possible.

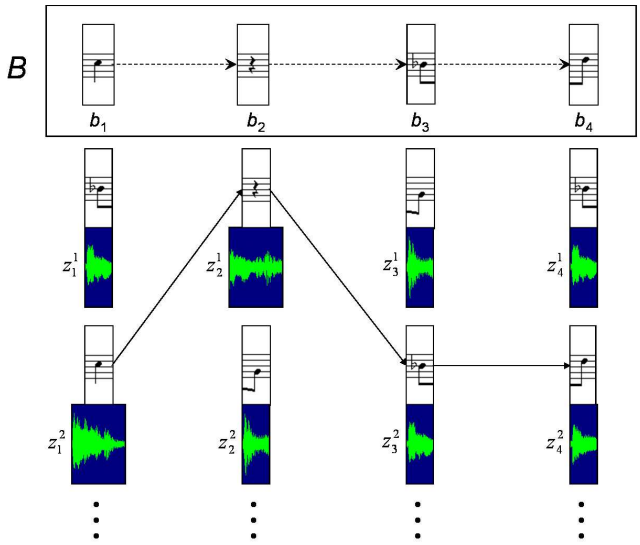

Figure 4. *Each frame in B has a number of candidate frames constructed from A' that can map to it. For each frame in B, we compute the lowest cost sequence ending in each of its candidates. Then, starting with the frame $|B|$, we compute the optimal sequence in reverse.*

5.3.4. Sequence Computation

Given C_{match}, $C_{\text{transition}}$, and the value α, we can compute the optimal sequence S of frame indices from A'. The optimal sequence minimizes the following quantity:

$$\alpha \sum_{i=1}^{n} c_{\text{match}}(i, S_i) + (1-\alpha) \sum_{i=1}^{n-1} c_{\text{transition}}(i, S_i, S_{i+1})$$

This type of minimization problem can be solved by a Viterbi algorithm, in $O(\kappa^2|B|)$ time and $O(\kappa|B|)$ space. For each frame b_i in B, we compute the cost of the optimal sequence of candidates to match b_1 to b_i for each possible candidate z_i^j to end the sequence. This takes $O(\kappa)$ time per candidate, since the optimal sequence may pass through any of the κ candidates for b_{i-1}. Then, we compute the optimal sequence to match b_1 to b_n in reverse. This is a dynamic programming algorithm that can be thought of as finding the optimal path through the candidates, using one candidate per target frame (see Figure 4).

We use all frames from the example pair (A, A') as candidates for each frame b_i, so the algorithm takes $O(|A|^2|B|)$ time.

5.3.5. Waveform Construction

Given selected candidate z_i^j for frame b_i, and example frame $a'_{r(i,j)}$ from which the sound data is to be taken, we need to transform the sound data to match the pitch

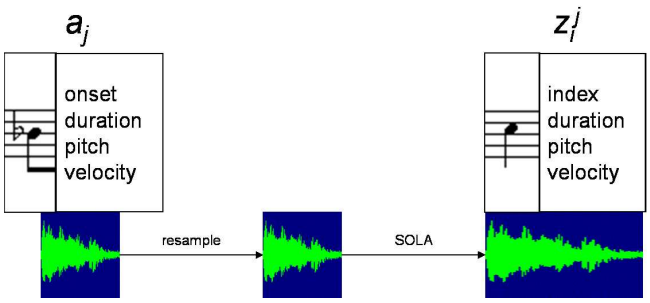

Figure 5. *To construct the waveform for candidate z_i^j, we take the frame $a'_{r(i,j)}$ corresponding to z_i^j, resample it to the desired pitch, and modify its duration using SOLA.*

and duration specified by the candidate. To transform the pitch, we resample the waveform. To change the duration, we use SOLA.

SOLA is a technique for changing the time of a signal independent of the pitch. The signal is broken up into overlapping chunks, which are then shifted relative to each other and added back together. The desired signal length determines the amount by which the chunks are shifted. In addition, the chunks should be shifted so as to align the signal optimally, which can be measured by cross-correlation.

SOLA yields good results as long as the ratio of original signal length to new signal length is not too large or small. Generally, any ratio between 0.5 and 2 sounds good. With a large enough (A, A') pair, it should always be possible to find a candidate whose original signal length is close enough to the target signal length. In addition, SOLA results can be improved by stretching some portions of a note while leaving others alone. The attack of a note, in particular, should often be left alone, as it contains energy at too many frequencies for good signal alignments to be found after shifting.

5.3.6. Concatenation

When a database unit corresponds to a single note, we construct the output waveform by simply concatenating the waveforms for each candidate in the optimal sequence.

5.3.7. Using Adjacent Note Pairs

Our algorithm, as previously described, requires a near-perfect (within about 5 ms) alignment of example score to example audio. When the alignment is poor, displeasing coarticulation artifacts can be heard. While the transition cost function penalizes sequences likely to produce these artifacts, there is often not enough example data to choose a good sequence of notes.

We created a second version of our algorithm that addresses this problem. The key observation is that while it is difficult or impossible to identify the transition point between two notes within 5 ms, it's easier to identify a larger window in which the transition occurs. Then, after the transition into a note, and before the transition out of the note, we can guarantee that only a single note can be heard.

Instead of concatenating single notes from the example, we concatenate pairs of adjacent notes, and blend the second note from the first pair into the first note from the second pair. For many instruments, note blending creates less noticeable artifacts than coarticulation. In some cases, an interval in the target score is not present in the example data, in which case we have no choice but to concatenate single notes, possibly creating coarticulation artifacts.

The steps of the algorithm as listed in Section 5.3 remain the same. However, the details of each step now reflect the use of note pairs as units.

The **segmentation** step still requires aligning the example score with the example audio. The alignment need not be perfect, but we must be able to bound the regions in which note transitions occur. When constructing the optimal sequence, we will try not to break up any of these regions.

The **candidate selection** step works almost exactly as before. For each pair of adjacent notes in the target score, we consider as candidates each pair of adjacent notes in the example. In addition, we continue to consider single notes as candidates, which can be used when the target interval between two adjacent notes is not present in the example (or if it is present, but the pair of notes is otherwise a poor match).

There are a few extra costs to factor into the **cost computation** step. Most importantly, we need to set $d_{\text{transform}}$ to ∞ for candidate intervals that do not match the target interval. We also must modify $d_{\text{transform}}$ to incur cost for the amount of SOLA and resampling done to both notes of the pair, instead of just a single note.

For $d_{\text{transition}}$, we add a constant value for each transition between pairs of notes that are not adjacent in the example, to account for the blending between the second note of the first pair and the first note of the second pair. Transitions between two single notes, as well as between a single note and an interval, are computed as in the other version of the algorithm.

The **sequence computation** step remains unchanged, and still takes time $O(|A|^2|B|)$. The **waveform construction** step also remains unchanged, except that we perform SOLA and resampling on both notes of a candidate pair.

The **concatenation** step is more difficult in this version

of the algorithm. We need to concatenate a sequence of candidate pairs (possibly including single notes as well) for which each two adjacent candidates share a note. To create the shared note, we blend smoothly from the second note of the first pair to the first note of the second pair. Both of these have already been transformed by SOLA and resampling, and have the correct pitch and duration. We blend the two waveforms using the following two-step procedure:

1. *Line up the waveforms.* Using cross-correlation, we compute the optimal relative shift. This aligns the periods of the waveforms.
2. *Blend the waveforms.* We perform a linear blend between the two waveforms.

This simple procedure works moderately well. In the future, we may use more advanced note morphing techniques.

6. RESULTS

A page of results and discussion can be accessed at `http://www.cs.washington.edu/homes/iansimon/audio_analogies/`.

For our results, we used trumpet and guitar lesson recordings as the example data. This is useful test data, since the pieces are monophonic, each piece is of manageable length (less than one minute), and some of the pieces are used to demonstrate a particular playing style.

Our algorithm needed approximately one minute to compute each output waveform. SOLA is the computation bottleneck, as even though the Viterbi algorithm runs in $O(|A|^2|B|)$, it operates on the scale of notes, while SOLA operates on the scale of individual samples. We used a sampling rate of 16000 Hz for all examples. A score that contains only 50 notes can correspond to a waveform of over one million samples.

The result quality varies with several features of the example data and target score, as well as which version of our algorithm we used. This variation is summarized in Table 1.

7. APPLICATIONS AND DISCUSSION

An obvious application of our algorithm is the synthesis of music from MIDI scores. Current MIDI synthesizers use model-based approaches, and their output can easily be distinguished from music performed by a human. Our system produces music that, while not yet free of artifacts, sounds more like authentic human performances. A simple extension to our algorithm in which "incorrect" notes are also considered as candidates allows us to play target score B not exactly as written. This opens up the possibility for some more interesting applications.

The Audio Analogies framework, when applied to the goal of MIDI score realization, presents a nice balance between playing "Paul Desmond's saxophone", and "playing Paul Desmond's saxophone like Paul Desmond". This balance can be thought of as controlling the amount of *texture transfer* that takes place. In the image domain, texture transfer refers to the problem of texturing a given image with a sample texture. For music, a natural analogue is to play one piece using the style and phrasing of another. Audio Analogies allow us to control the extent to which musical texture is transferred. At one extreme, the musical score is interpreted rigidly, and its notes are played exactly. At the other, the musical score is completely ignored, and the analogy reduces to *texture synthesis*. In the image domain, texture synthesis refers to the problem of generating more texture like a given sample patch. In the realm of audio, texture synthesis can mean extending a piece of music indefinitely.

One can imagine an electronic piano keyboard with an "auto-stylization" dial. As a performer plays a piece of music, he/she can adjust this dial to control the α value of the sound coming from the keyboard. Perhaps the keyboard comes equipped with a variety of example styles that can be imitated. Of course, the Audio Analogies algorithm as we described it requires the entire score B in order to begin computing B', so the above scenario is not immediately achievable. The idea of being able to import musical styles from other performers or genres into a new piece of music, though, is enjoyable, and can often be done successfully using Audio Analogies.

Currently, we deal only with the case of monophonic music and focus predominantly on playing score B exactly. Some of our examples illustrate the effects of using a polyphonic example pair (A, A'), as well as our limited ability to modify score B in an attempt to capture more of the musical style of A.

8. FUTURE WORK

There are several directions we would like to take in the near future:

- *Automatically align A and A'.* This involves detecting note boundaries in A', and modifying the score A to reflect these boundaries [5]. Aligning by hand is too time-consuming for longer pieces of music, and automatic alignment could possibly yield better precision as well.

Input Data	Version 1 (Single Notes) Results	Version 2 (Note Pairs) Results
perfectly aligned example, most target pitches and durations present in example	very few noticeable artifacts	possible note blending artifacts
perfectly aligned example, many target pitches and durations missing from example	SOLA and resampling artifacts	possible note blending artifacts, as well as SOLA and resampling artifacts
imperfectly aligned example, all target intervals present in example, most target pitches and durations present in example	coarticulation artifacts	possible note blending artifacts, SOLA and resampling artifacts
imperfectly aligned example, all target intervals present in example, many target pitches and durations missing from example	coarticulation artifacts, SOLA and resampling artifacts	possible note blending artifacts, SOLA and resampling artifacts
imperfectly aligned example, some target intervals missing from example, most target pitches and durations present in example	coarticulation artifacts	possible note blending artifacts, coarticulation artifacts
imperfectly aligned example, some target intervals missing from example, many target pitches and durations missing from example	coarticulation artifacts, SOLA and resampling artifacts	possible note blending artifacts, coarticulation artifacts, SOLA and resampling artifacts

Table 1. *Go to* `http://www.cs.washington.edu/homes/iansimon/audio_analogies/` *to hear the results. The quality of the output of both versions of our algorithm depends upon certain characteristics of the input. Coarticulation artifacts can be heard when an example note contains undesired pieces of another note. SOLA and resampling artifacts result from excessive changes to the pitch or duration of a note. Note blending artifacts result from the blending between two notes of the same pitch and duration, but with differences not accounted for by our algorithm, such as timbre and vibrato.*

- *Use note sequences of any length as units.* When trying to synthesize style, a major restriction is our current method of playing one note in B' for one note in B. A stylistic performance of score B might add or remove notes. We could move closer to this goal by using techniques described (for images) by Jojic [4].

- *Extend our algorithm to polyphonic music.* Synthesizing polyphonic music necessitates changing our algorithm considerably. Multiple sequences of notes are evolving simultaneously, and we need a way of representing this. In addition, it would be useful to be able to extract a single note from a polyphonic waveform, though this is largely an unsolved problem.

9. REFERENCES

[1] J. Arcos, R. de Mantaras, and X. Serra. Saxex: a case-based reasoning system for generating expressive musical performances, 1997.

[2] Istvan Derenyi and Roger B. Dannenberg. Synthesizing trumpet performances. In *Proceedings of the International Computer Music Conference*, pages 490–496. International Computer Music Association, 1998.

[3] Aaron Hertzmann, Charles E. Jacobs, Nuria Oliver, Brian Curless, and David H. Salesin. Image analogies. In Eugene Fiume, editor, *SIGGRAPH 2001, Computer Graphics Proceedings*, pages 327–340. ACM Press / ACM SIGGRAPH, 2001.

[4] N. Jojic, B. Frey, and A. Kannan. Epitomic analysis of appearance and shape. In *Proceedings of the International Conference on Computer Vision (ICCV)*, 2003.

[5] Nicola Orio and Diemo Schwarz. Alignment of monophonic and polyphonic music to a score, 2001.

[6] Christopher Raphael. Automatic segmentation of acoustic musical signals using hidden markov models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 21(4):360–370, 1999.

[7] S. Roucos and A. Wilgus. High quality time-scale modification for speech. In *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing*, pages 493–496. IEEE, 1985.

[8] D. Schwarz. A system for data-driven concatenative sound synthesis, 2000.

[9] A. Zils and F. Pachet. Musical mosaicing. In *Proceedings of DAFX 01*, 2001.