Concatenative Synthesis Using Score-Aligned Transcriptions*

Roger B. Dannenberg

School of Computer Science, Carnegie Mellon University
dannenberg@cs.cmu.edu

Abstract

Concatenative synthesis assembles segments of pre-recorded audio to create new sound. In this variation, pre-recorded audio is labeled by aligning a polyphonic MIDI representation, essentially forming a symbolic transcription of the source material. Given a MIDI file to be synthesized, matching segments of MIDI describing the source are used to locate corresponding sound segments, which can be spliced to form the output. This technique is related to audio mosaics where similar spectral frames are substituted except that matching symbolic MIDI data allows for substitutions that are timbrally dissimilar yet harmonically and rhythmically identical.

1 Introduction

Concatenative synthesis allows synthesizers to reuse "natural" acoustic sounds from large databases in order to assemble output sound according to some specification. (Schwarz 2000, Schwarz 2003, Simon et al. 2005) While concatenative synthesis was originally developed for high-quality speech synthesis, the technique is also used for more artistic applications. The basic idea is that a *target* specification (a score, or features derived from audio) is synthesized by splicing sound *units* obtained from a database of *source* sounds. This approach is analogous to image mosaics and is sometimes called music mosaicing (Zils and Pachet 2001; Lazier and Cook, 2003).

One of the interesting and much discussed aspects of music mosaicing is the selection of sound units from the database. The quality of match between sound units and the target sound is an important factor in determining the similarity of the resulting sound.

Various criteria can be used to evaluate the similarity and appropriateness of sound units in (re)constructing the target. Typically, a variety of audio features are considered, such as pitch, loudness, and spectrum. Depending on the features, the resulting sound will capture some aspects of the target specification but perhaps not others. For example, if the spectrum is closely modeled, we expect to hear a similar timbral evolution. On the other hand, if the pitch is the important criteria for comparison then perhaps a melody will be retained and the timbre will change completely.

In general, the features that are used to compare units are an *abstraction* or a projection of a very high-dimensional signal space (e.g. 44100 samples per second) onto a low-dimensional feature space. To the extent that the low-dimensional feature space captures interesting perceptual information, music mosaicing will produce results such that abstract qualities in the original sound are preserved and recognized.

For music, a very salient property is harmony and pitch content. We recognize melody and chord progressions regardless of the instrumentation or timbre, dynamic levels, and other characteristics. It seems that harmony could be an interesting basis for audio mosaicing, but it is very difficult to determine harmony and pitch content from arbitrary music audio, especially in the case of polyphony. A program that could determine the pitch content of a sample of audio would need to solve the audio transcription problem.

An alternative to music transcription is polyphonic music alignment. In this process, a score, usually represented as a standard MIDI file, is time-aligned to audio. Once the correspondence between MIDI and audio is set up, one can easily read off the pitches sounding at a given location in the audio. Alternatively, given a section of MIDI where desired pitches are present, one can locate the audio realization of those pitches.

In the present study, synthesis is performed starting with a standard MIDI file. This is different from music mosaicing where the input is audio. Starting from MIDI is closer in spirit to concatenative synthesis, where a specific symbolic score (or a notated sequence of phonemes) is to be synthesized into music (or speech). However, this study is more concerned with harmony and polyphonic sounds. Due to the large space of possible note combinations (e.g. 88^{10} for 10-note polyphony over 88 pitches), it is unreasonable to expect the high-quality synthesis one might obtain from a monophonic instrument using a large database and concatenative techniques. Instead, the goal is to create interesting textures that retain the timbral quality of the source music in the database while following the harmony (and it turns out much of the rhythm) of the source MIDI file.

Figure 1 is a high-level description of the process. Music audio sources are aligned with corresponding MIDI files to obtain time maps so that chords in the MIDI files can be located in the audio source. Next, each MIDI file is segmented to obtain a sequence of chords or *concurrencies* (Pardo and Birmingham 2000). The time and duration of

*Published as: Roger B. Dannenberg, "Concatenative Synthesis Using Score-Aligned Transcriptions," in *Proceedings of the 2006 International Computer Music Conference,* New Orleans, LA, Nov. 2006. San Francisco, CA: The International Computer Music Assoc., 2006. pp. 352-355.

these chords comprise the units in the database. The target MIDI file to be synthesized is then segmented in the same way to extract concurrencies. A search process finds the unit with the best match in the database for each concurrency in the target. Each unit is mapped to the corresponding audio, and the audio units are concatenated to form the output.

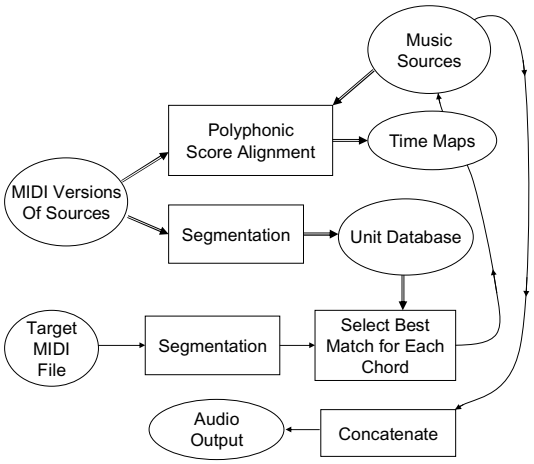

Figure 1. System diagram for concatenative synthesis using score-aligned transcriptions.

The next section describes in brief how polyphonic alignment is achieved. Section 3 describes how music is analyzed and segmented to form the database. Section 4 describes criteria that are used to determine the similarity of sound units in the database to segments of the source MIDI file. Section 5 describes some synthesis and implementation details. This is followed by some discussion and conclusions.

2 Polyphonic Alignment

A key ingredient in this approach to concatenative synthesis is the labeling of audio with symbolic pitches. An accurate labeling of pitches (as opposed to a spectral representation) will allow us to search the audio for particular pitches and chords. Labeling is achieved by aligning MIDI data to the audio file as described by Hu, Dannenberg, and Tzanetakis (2003).

Where does the MIDI come from? Most popular and classical music has been transcribed by hand into MIDI files that can be located on the Web. There is no reliable way to automate this search, so finding the file is a manual process. MIDI files are not always perfect transcriptions, and usually MIDI lacks the expressive timing of a recorded audio performance, so the MIDI data must be aligned to the audio before it can be used.

Polyphonic alignment is performed by converting the MIDI file and the audio file into a representation called the *chromagram*. (Wakefield 1999) A chromagram is a sequence of *chroma vectors*, 12-element vectors representing the total spectral energy corresponding to each of the 12 pitch classes (C, C#, D, …, B). The chroma vector is chosen because it captures harmonic and melodic information, which is shared by audio and MIDI, and it tends to be insensitive to amplitude and timbral differences, which tend not to match very well between audio and MIDI.

Audio data is divided into 250ms frames, each of which is analyzed with the FFT. The magnitudes of the FFT bins are then converted to a chroma vector.

For MIDI data, chroma vectors are estimated without converting to audio or performing FFTs. Instead, each chroma vector element is the sum of all the matching pitch classes sounding during that frame, weighted by the key velocity and duration (0.25, or less if the note begins or ends during the frame).

Audio and MIDI chroma vectors are each normalized so the 12 elements have a mean of 0 and a standard deviation of 1. The next step uses dynamic time warping to find the best time alignment, using Euclidean distance between chroma vectors.

Finally, the rough alignment, which is a path quantized to points along a 0.25s grid, is smoothed at each point by finding the best fit to the nearest 7 points, using linear regression. The resulting points define a sampled function that can be linearly interpolated to map between MIDI file time and audio file time.

The score alignment process is written in C++ and runs about 20 times real time on a 2.4 GHz Pentium 4 CPU. Source code is available from the author.

3 Music Analysis and Segmentation

Music analysis is performed on both the target and source MIDI data. After parsing the data from a MIDI file, the notes from all channels and tracks are merged, forming a single polyphonic part. All notes on channel 10 are removed to avoid drum data. Consider the data in Figure 2. Every time a note begins or ends, the set of currently sounding pitches changes. The score is segmented accordingly, and these pitch sets are called *concurrencies* (Pardo and Birmingham 2000).

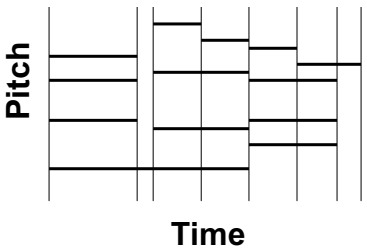

Figure 2. Scores are segmented at boundaries corresponding to note beginnings and endings to form pitch set sequences.

When the MIDI file has actual expressive performance timing as opposed to notes that are quantized to beat boundaries, there will tend to be very short concurrencies due to the "ragged" alignment of notes. A minimum

concurrency duration on the order of 0.ls is imposed to merge very short concurrencies. Scanning the data in time order, if a concurrency boundary occurs within 0.1s of the start of the concurrency, the boundary is removed.

Each unit in the database denotes a concurrency in the MIDI file as well as the corresponding segment in a source audio file. Each unit is represented by a 4-tuple consisting of (1) a set of pitches present in the concurrency (2) the onset time of the concurrency *in the corresponding audio file*, (3) the duration *in the audio file*, and (4) the *audio* file name. Notice that while the pitch data is extracted from the MIDI file, the times and file name refer to the corresponding audio file where sound will be obtained in the synthesis step.

Similar to sources in the database, the target MIDI file is represented by 4-tuples denoting its concurrencies. In this case, there is no corresponding audio, so the time and duration fields refer to those of the source MIDI file, and the file name is ignored.

4 Selecting Similar Sound Units

Given a target MIDI file, this system renders a sound with a similar sequence of concurrencies, thus reproducing the harmonic progressions of the target MIDI file using sounds from the database. Because the output units will match the timing of the target MIDI file, much of the rhythm of the target file is preserved as well. Furthermore, to the extent that units are found with matching top notes, melody will also be synthesized.

For each target file concurrency, a unit is selected from the database by evaluating similarity between the target concurrency and every unit in the database. The similarity is a weighted sum of the following (default weights are written in parentheses):

- *Pitch similarity* (10), the degree of match between sets of pitches. This is based on the F-score (see http://en.wikipedia.org/wiki/Information_retrieval), defined as follows: Let S be the set of pitches in the target concurrency and U be the set of pitches in the database (source) concurrency. Let $a = |U \cap S|$, $b = |U - S|$, and $c = |S - U|$. Let r (recall) $= a/(a+b)$ and p (precision) $= a/(a+c)$. Finally, the F-score $f = 2rp/(r+p)$, which ranges from 0 to 1.
- *Pitch class similarity* (1000): the degree of match between sets of pitch classes. This is computed just like pitch similarity (above), but sets of pitch *classes* replace sets of pitches.
- *Top note similarity* (100): If the highest notes are identical, return a score of 1. If the top notes have the same pitch class, return a score of 0.5. Otherwise the score is 0.
- *Duration suitability* (100): The unit should be long enough to play the entire concurrency. The score is $\min(0.1, \log(d_1/d_2))$, where d_1 is the duration of the database (source) unit and d_2 is the duration of the target concurrency. The *min* function expresses the idea that there is no advantage to having a source unit much longer than the target unit.
- *Recency rating* (1000): Avoid reusing database units too soon. Let n be the number of concurrencies selected since the last use of a given database unit. If the database unit is unused so far or if $n = 0$ (immediate reuse), return 1. Otherwise, return $1 - (1/n)$, yielding a score from 0 to 1.

A common consideration in concatenative synthesis is the quality of the transition, and often transitions are used directly from the database. Because the current database is small compared to the number of possible chord transitions, no attempt is made to use "natural" transitions. Another currently ignored but possibly important consideration is the presence and alignment of rhythmic accents and other articulatory details.

5 Synthesis

After selecting the unit with the highest similarity rating from the database, the start time, duration, and file name are used to access audio samples. In the current implementation, units are concatenated using a simple cross-fade. Normally, audio is extracted using the start time of the unit with a duration that is 0.1s longer than the duration of the source MIDI file concurrency. Each sound is given a 0.1s fade-in and a 0.1s fade-out, and sounds are overlapped by 0.1s. This creates a cross-fade between successive units and preserves the rhythm and timing of the target MIDI file. If the database unit is shorter than the source concurrency, the audio unit is repeated as many times as necessary to fill the time. (A more sophisticated form of time-stretching could be used here.)

The entire synthesis system is written in Nyquist (www.cs.cmu.edu/~music/nyquist) with the exception of the score-alignment, which is performed by a stand-alone program. The score-alignment program outputs text files describing notes and their aligned times and durations. These files are parsed by a Nyquist program to build a database of units. A similar process builds a list of concurrencies from the source MIDI file. Currently, the database is a simple list, and the most similar sound unit is found through a linear search. The extraction of sound from audio files, cross-fading, and assembly are all easily performed using Nyquist signal-processing primitives. The entire Nyquist implementation is about 460 lines of code (not counting the score alignment program).

5.1 Speeding Up the Search

To handle a very large database, linear search might be too slow. An index based on pitch class similarity can be used to speed up the search. This is simple if the search is constrained to consider only units that have a perfect match, or in other words, the weighting on pitch class similarity is so high that only exact matches between pitch class sets need be considered. Since there are 12 pitch classes, there

are $2^{12}-1 = 4095$ non-empty sets of pitch classes. When constructing the database, it can be partitioned according to pitch class sets. Naively, one might expect a speed-up of 4095, but not all pitch class sets are equally common.

To get some indication of how pitch class sets are distributed, I collected statistics from the database extracted from Beethoven's Symphonie 5, 1st Movement. There are 1900 concurrencies, but only 166 distinct pitch class sets (out of 4095). Thus, one might hope for a speedup of 166 because the database can be partitioned into 166 sets of units. This is still overly optimistic because the distribution of units across these 166 partitions is highly skewed. Assuming the source MIDI file has the same distribution of pitch class sets, the average expected number of audio units with a perfect pitch class similarity is approximately 42, giving a speedup of about 45 (i.e. 1900 / 42).

One could also consider imperfect pitch similarity matches, for example, by searching units that have only one additional pitch class or only one missing pitch class. This would of course reduce the speedup even further. Overall, these numbers should be taken as some indication of a promising approach, but the results will depend greatly upon the actual data. For example, a collection of symphonies in different keys should occupy a much greater proportion of the 4095 possible pitch class sets.

6 Results and Applications

When dealing with polyphonic audio, the number of combinations of pitches is so vast that one cannot hope to achieve anything like high quality synthesis, and this is not my goal. I have experimented using a Beethoven symphony to populate the database and various MIDI files as sources. In most cases, the output has a recognizable harmonic quality, but it is difficult to follow along and recognize the source music on a measure-by-measure or chord-by-chord basis. The target's rhythm is reconstructed at least to some extent because units are spliced on concurrency boundaries where target notes begin and end.

One variation I tried does a slow cross-fade back and forth between the original source, synthesized from the source MIDI file, and the concatenative synthesis output. The return to the source helps the listener to remain oriented and sets up expectations for the synthesis output, which is always a bit jarring due to differences in timbre and sometimes amusing in how the harmony is re-voiced.

Another approach is to synthesize the melody from the target MIDI file, using a conventional synthesizer, and to superimpose that with harmony that is created by concatenative synthesis of the same MIDI file. Here, the synthesized melody serves as a reminder of and orientation to the target piece while the somewhat chaotic and turbulent concatenative synthesis output supplies the harmony and some rhythm.

Some examples of this approach can be heard at http://www.cs.cmu.edu/~music/concat/concat.html.

7 Summary and Conclusions

Concatenative synthesis is a recent approach to sound synthesis that uses a database of stored and analyzed sounds to reconstruct music. The selection of sound units from the database is always driven by some notion of similarity between the database source sounds and the target specification. In this work, source sounds are labeled with MIDI transcriptions, using automated score alignment to simplify the collection of data. This allows selection to be based on symbolic pitches as opposed to spectral or other features. The resulting system can synthesize a harmonic sequence that corresponds to a target MIDI file. Experiments have synthesized music that contains the harmonic progression of the target MIDI file, but in general, the results are somewhat disorienting because of the juxtaposition of imperfectly matching chords and uncontrolled timbral content. Mixing in some synthesized sounds from the target MIDI file, for example by including the melody, can help the listener find the correspondence between the target and the concatenatively synthesized version.

A larger database is under construction, and it will be interesting to hear the effect of having more audio units available. A larger database should allow for better matching to the pitches specified in the MIDI file and even more interesting outputs. Future work might also attempt to enlarge the database by pitch-shifting sources.

References

Hu, N., R. B. Dannenberg, and G. Tzanetakis. 2003. "Polyphonic Audio Matching and Alignment for Music Retrieval." *2003 IEEE Workshop on Applications of Signal Processing to Audio and Acoustics (WASPAA)*. New York: IEEE, pp. 185-188.

Lazier, A., and P. Cook. 2003. "Mosievius: Feature Driven Interactive Audio Mosaicing." *Proceedings of the 5th International Conference on Digital Audio Effects (DAFx-03)*.

Pardo, B., and W. Birmingham. 2000. "Automated Partitioning of Tonal Music." *Proceedings of the 13th International FLAIRS Conference*. AAAI Press.

Schwarz, D. 2000. "A System for Data-Driven Concatenative Sound Synthesis." *Proceedings of the COST G-6 Conference on Digital Audio Effects (DAFX-00)*.

Schwarz, D. 2003. "New Developments in Data-Driven Concatenative Sound Synthesis." *Proceedings of the 2003 International Computer Music Conference*. International Computer Music Association, pp 443-446.

Simon, I., S. Basu, D. Salesin, and Maneesh Agrawala. 2005. "Audio Analogies: Creating New Music From An Existing Performance by Concatenative Synthesis." *Conference Proceedings International Computer Music Conference*. International Computer Music Association, pp. 65-72.

Wakefield, G. H. 1999. "Mathematical Representation of Joint Time-Chroma Distributions." *International Symposium on Optical Science, Engineering, and Instrumentation, SPIE'99*.

Zils, A., and F. Pachet. 2001. "Musical Mosaicing." *Proceedings of the COST G-G Confernece on Digital Audio Effects (DAFX)*.