

CONSTRAINED EXAMPLE-BASED AUDIO SYNTHESIS

Stephan Wenger and Marcus Magnor

Institut für Computergraphik
TU Braunschweig
Braunschweig, Germany

ABSTRACT

Music and sound play an important role for the emotional involvement of the spectator in visual media such as movies, video games or artistic shows. However, the generation of suitable background music and atmospheric sound matching the visual context is a tedious and typically completely manual task. For content creators, it would be desirable to be able to specify a sound example and a few key points at which certain parts are to be played, and to then automatically generate a soundtrack subject to these constraints. For interactive media such as video games, on the other hand, example-based background sound could automatically adapt to the game events. We propose a new example-based audio synthesis method for both applications which works for musical and texture-like sound examples alike. Our algorithm is evaluated on a variety of musical styles, from classical music to punk rock, and on several texture-like atmospheric sounds.

Index Terms— audio synthesis, example-based synthesis, sound textures

1. INTRODUCTION

A significant part of human cognition is based on the perception of sound. Sound conveys information about what is happening in our environment, even outside the field of vision. Sound, and especially music, also strongly affect the emotional state of the listener. Even early silent films were accompanied by – often live – music, and many motion pictures have become famous for their soundtracks. Since the generation of an appropriate soundtrack plays an important role whenever visual content is created, it receives an increasing amount of attention also in the computer graphics community [1–4]. This is especially true in the context of computer animation, where no on-scene sound recording is possible; but also interactive content like games as well as live performances and art installations demand an appropriate soundtrack.

One important function of the soundtrack is to emphasize the central message or mood of the content. Since these typically change over time, the soundtrack should be able to adapt to or be synchronized with important events in the scene. In

feature film productions, this is often achieved by manually cutting and blending sounds and music – or even by having a composer write a piece of music – to match the content. Obviously, this is a tedious and time-consuming task which may not be reasonable for smaller productions.

As a simple example, consider a home video recording of given duration that is to be accompanied by a piece of music of different length. If the user wants the beginning and end points of the video and audio to coincide, one of the signals would have to be scaled or cropped. This can be avoided if a *synthesis* or *retargeting* method like the one proposed in this paper is used to piece together a new soundtrack, matching the beginning and end points, from sections of the example piece of music or sound. This description easily generalizes to the case where multiple key events in the video are to be accompanied by specific parts of the example.

Another common task is the real-time generation of soundtracks that adapt to live events, for example in computer games where the mood of the background music may adapt to the twists of fate of the main character.

The aforementioned applications require the synthesis of a novel soundtrack from an audio example subject to different kinds of constraints. We propose an algorithm that approaches this problem in two separate steps. First, a user-specified audio example – e.g., in form of a simple MP3 file – is searched for self-similarities, that is, pairs of positions in the example where a jump from one position to the other will likely go unnoticed. Then, the list of possible jumps is used to find a succession of sections from the audio example that minimizes the transition errors between sections while complying with user-specified constraints, such as the length of the synthesized audio clip or the changing mood of the music. The approach is inspired by similar cut-and-stitch methods used in computer graphics, for example to generate textures of arbitrary extent in one dimension [5].

The structure of our paper is as follows: In Sect. 2, we discuss related approaches to audio analysis and synthesis. Sect. 3 covers the search for suitable cutting points, while Sect. 4 explains how an appropriate succession of example sections is found subject to constraints, and Sect. 5 deals with non-constrained synthesis. Results are provided in the accompanying video and audio files and discussed in Sect. 6. The

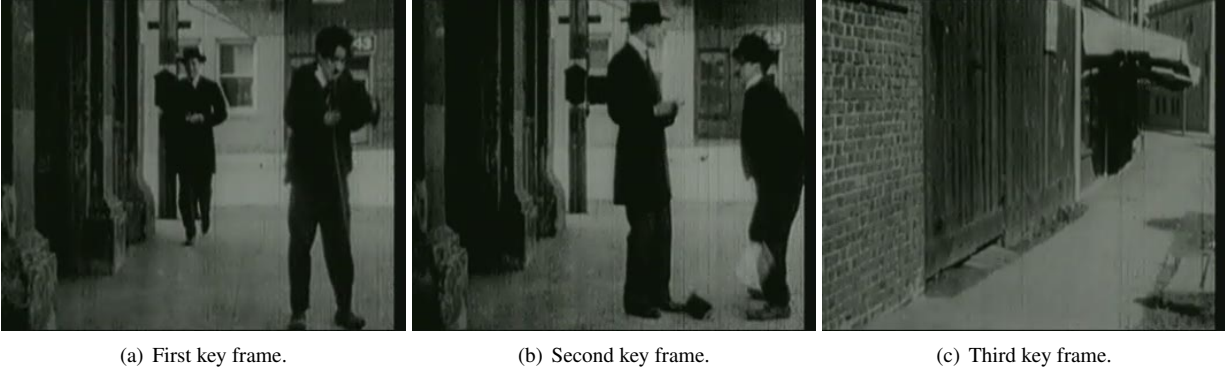


Fig. 1. Key frames in a video sequence from Charlie Chaplin’s “Police” (1916). The displayed frames are set to specific samples in the audio example. The first key frame (a) marks the beginning of the video sequence and is set to the first sample of the audio example, so that the synthesized sequence starts with the intro of the song. The second key frame (b) is located in the middle of the movie when the actor has just thrown the book to the floor. This key frame is set to the start of an intense instrumental sequence in the audio example. Finally, the last frame of the video (c) is set to the last sample of the audio example, so that both video and audio end simultaneously.

paper ends with concluding remarks in Sect. 7.

2. RELATED WORK

Example-based synthesis of *audio textures* by cutting and stitching has been studied for several years [6–8]. The methods rely on a segmentation of the example clip into suitable “sub-clips”. Based on the transition probability between one sub-clip and another, a texture of arbitrary length is then generated. However, because the transition probability does not capture the large-scale structure of the example, these approaches typically do not perform well on complex pieces of music.

Audio and user directed sound synthesis [9] is based on frame-wise similarity analysis and works for stochastic and periodic sound textures but has not been tested on musical examples. Sound-by-numbers [1] analyzes the example and constructs the synthesized sound in wavelet space, thereby capturing more large-scale structure of the example, which also makes it applicable to some musical genres like ambient music. A survey of similar sound texture modeling methods has been published by Strobl et al. [10].

Audio-based music structure analysis [11], in turn, attempts to analyze the large-scale structure of a piece of music based on the self-similarity of the piece. The resulting high-level description can theoretically be used to rearrange the parts of the example, but a more fine-grained analysis is required to create variations within one part, and the approach is not able to handle sounds without musical structure or with a kind of structure that is not anticipated in the underlying music theory.

Concatenative sound synthesis [12–15], on the other hand, allows to synthesize sounds subject to fine-grained constraints (pitch, loudness, timbre etc.) from a database of examples.

Most often used in composing electronic music, the approach can, in principle, also be used to synthesize sound textures from examples, but is likely to fail for musical examples.

In summary, while controlled synthesis of texture-like sounds is well studied, existing methods typically fail for structured musical examples. Our algorithm, in contrast, uses a multi-resolution scheme for the detection of self-similarities, which not only allows for synthesis of music, but also speeds up the algorithm so that long sound examples can be analyzed quickly. Because the multi-resolution scheme covers the example from the large-scale structure down to single samples, the approach handles music with complex large-scale structure as well as near-random sound textures, and allows for sample-perfect alignment of cuts, so that no blending is necessary. The direct search for cutting points also obviates the need for a separate segmentation step or for fixed-size search windows that have to be adjusted to the duration of features to be matched, so that the necessary amount of user interaction is reduced to specifying an audio example, e.g. as an MP3 file, and a list of constraints, e.g. the desired length of the synthesis result.

3. CUT POINT SEARCH

The most critical step in our algorithm is finding a good set of *cut points* at which a jump from one point in the example to another can occur without being noticed. The most obvious algorithm for finding such points consists in computing the self-similarity matrix of the example in some appropriate transform domain (e.g. by splitting the example into frames of a few thousand samples and comparing the frequency spectra or the more perceptually motivated mel-frequency cepstral coefficients of these frames) and locating regions of maximal self-similarity. However, this approach suffers from three im-

portant drawbacks.

Firstly, if only entire frames can be compared, all cut points will lie apart by an integer multiple of the frame size unless both the start and end point of the jump within the frame are subsequently optimized with another algorithm. This is an important limitation whenever the example exhibits temporal regularities, such as the beat in pieces of music, where all cut points should lie apart by an integer multiple of the beat length. While this could be achieved by using a beat-detection algorithm (e.g. [16]) in order to determine a reasonable frame size, this approach would fail for music with tempo variations and for non-musical examples.

Secondly, the maxima in the self-similarity matrix tend to cluster because frames that are temporally close to each other are likely to have similar frequency signatures, and because similarity between one pair of frames usually implies similarity of their successors. The subsequent path search would then quickly become overly complex, as every possible cut from a cluster of very similar cuts multiplies the number of possible paths.

Finally, the number of elements in the self-similarity matrix grows quadratically with the length of the input. While computation of the matrix may be feasible for short sound texture examples, it quickly becomes impractical for longer pieces of music.

We attempt to solve these three issues at once with a multi-resolution approach. The self-similarity matrix is first computed on very long frames (typically about 10 seconds, or about 500000 frames), such that the matrix stays reasonably small. From this low-resolution matrix, which captures the large-scale structure of the example, the best few entries are selected, and the algorithm is run again on these blocks with a smaller frame size and a lower number of entries to keep. If the initial frame size is large enough, this will guarantee a rather even distribution of cuts across the entire matrix. Also, when the frame size approaches single audio samples, sample-perfect alignment of the example sections before and after the jump emerges automatically so that no blending is necessary.

Let e be the example waveform represented as a vector of samples, $n = 5$ the number of resolution levels, $s = 16$ the scaling factor between levels of the resolution pyramid, and $k = 256$ the number of samples to keep on the first resolution level. Then, a tree of possible cuts is found by calling $\text{search}(e, e, s^n, k)$, where $\text{search}(e_1, e_2, l, k)$ is defined as

```

split  $e_1$  and  $e_2$  into frames of length  $l$ 
 $N_1$  = number of blocks in  $e_1$ 
 $N_2$  = number of blocks in  $e_2$ 
if this is not the innermost level
    apply Fourier transform to blocks
for all blocks  $b_1$  in  $e_1$ 
    for all blocks  $b_2$  in  $e_2$ 
         $c(b_1, b_2) := \frac{(b_1 - b_2)^2}{(b_1 + b_2)^2}$ 
        if  $b_1$  and  $b_2$  are from the same example location
             $c(b_1, b_2) := \infty$  (disallow very short jumps)
if this is not the innermost level
    for all block pairs  $(b_1, b_2)$  in the best  $k$  entries of  $c$ 
        child =
             $\text{search}(b_1, b_2, \max(l/s, 1), \max(\frac{k}{N_1 N_2}, 1))$ 
        add (child,  $c(b_1, b_2)$ ) to list of children
    return list of children
else
    return coordinates and values of best  $k$  entries of  $c$ .

```

Every path from the root of this tree to a leaf is a possible cut. The error induced by each cut can be defined as a weighted sum over the respective errors on all resolution levels. A uniform weighting usually works well. Optionally, all but the best few (e.g., 40) cuts can be discarded in the end in order to speed up the subsequent path search algorithm.

4. PATH SEARCH

As soon as a sufficiently large number of possible cuts has been determined, we may use these cuts to piece together sections of the example such that the synthesized target sound satisfies user-specified constraints. Most constraints – such as the desired overall length of the sample – can be formalized by prescribing the temporal position of a certain piece of the example – such as the first or last sample – in the synthesized output. The remaining task is now to find a path through the example, playing segments or jumping between the end points of a cut, that starts and ends at a given position in the example and produces a synthesis result of the specified duration. Multiple constraints can then be enforced by synthesizing the parts between each pair of successive constraints separately.

The search for an optimal succession of example sequences between two given end points can be formulated as the search for a shortest path in a graph. However, the size of this graph grows exponentially with the number of possible cuts, such that finding an exact solution quickly becomes impractical. Instead, we propose a heuristic best-first search that

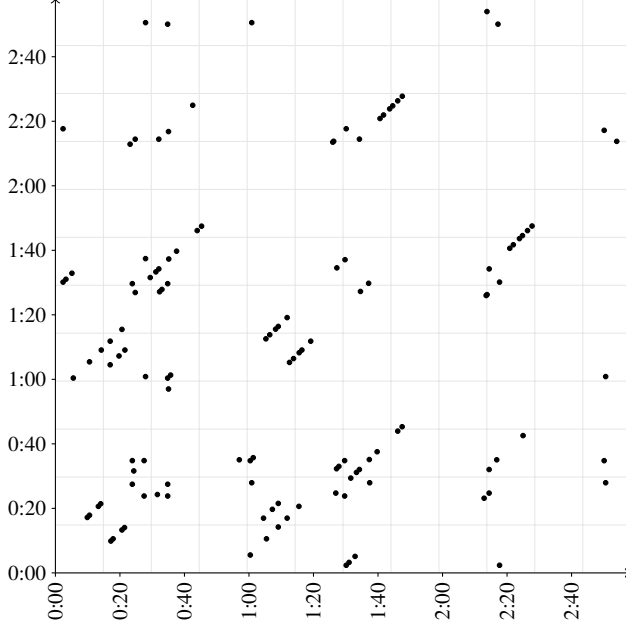


Fig. 2. Cut plot – akin to a self-similarity matrix – for the example song used for synchronizing the soundtrack for the Charlie Chaplin video. The dots indicate positions in the example where jumps from the time marked on the abscissa to the time marked on the ordinate are perceptually smooth. The grid lines indicate ten times the block size used in the lowest analysis resolution level.

finds the path with minimal error metric among all synthesized target sounds consisting of less than a specified amount of example segments.

The error metric to minimize has to enforce three possibly contradictory goals: The error e_{c_i} of the used cuts c_i shall be small, the sum of the durations l_{s_j} of the used example segments s_j has to be close to the desired duration l_{desired} , and unnecessary repetition of the same example segment has to be avoided. We achieve this by formulating a simple weighted sum of three error terms: the accumulated cut cost $\sum_i e_{c_i}$, the squared difference between the current and the desired duration $(\sum_j l_{s_j} - l_{\text{desired}})^2$, and the product of the number of occurrences of each possible cut c , $\prod_c \max(\sum_i \delta_{c,c_i}, 1)$, where δ_{c,c_i} is one when $c = c_i$ and zero otherwise. The error function for a given set of cuts c_i and the enclosed example segments s_j can thus be written as

$$\begin{aligned} \text{error}(c_i, s_j) &= \lambda_{\text{cut}} \sum_i e_{c_i} \\ &+ \lambda_{\text{duration}} (\sum_j l_{s_j} - l_{\text{desired}})^2 \\ &+ \lambda_{\text{repetition}} \prod_c \max(\sum_i \delta_{c,c_i}, 1) \quad , \end{aligned} \quad (1)$$

for user-specified weighting factors λ_{cut} , $\lambda_{\text{duration}}$ and $\lambda_{\text{repetition}}$. If necessary, this metric can easily be extended to incorporate further constraints, for example to enforce a change of mood in a piece of music.

The best-first search can be implemented efficiently by using a sorted heap of incomplete paths which initially contains only the path from the first sample of the example to the first possible cut. Successively, the respective best path is taken from the heap, and for each possible continuation of the path (playing until the next cut or jumping to another cut and playing from there), a new path is created by extending the old one, and is put onto the heap. As soon as a path is complete (that is, it reaches the end of the example), it is put onto a heap of complete paths. This is iterated until a certain number of complete paths has been found, and the best complete path is returned.

It is possible that some paths grow longer than the desired length of the target. Because the error metric of these paths can only become worse by appending new segments, they will most probably not be part of the final solution. Therefore, for increased efficiency, paths exceeding the desired target length are immediately discarded.

The final path contains a list of segments from the example. These segments can now simply be concatenated to produce the desired synthesized target. Because the position of the cuts is determined with sample-perfect precision, no blending is necessary. When creating a target of exactly the desired length is impossible without using a very bad cut (for example, if the desired length differs from the example length by a non-multiple of the beat length), pitch-invariant scaling of the target by a small factor can be applied in order to attain the exact length with minimal distortion.

5. INFINITE AND ON-THE-FLY SYNTHESIS

For some applications, such as video games or live art installations, it might be desirable to generate an endless audio stream from an audio example. This can easily be achieved by concatenating a Markov chain of example segments based on the list of cuts and their respective errors. For example, if at one point of the chain a number of cuts c_i with errors e_{c_i} is available, the transition probability along the cut might be given as

$$p(c_i) = \frac{e^{c_i}}{1 + \sum_i e^{c_i}} \quad ,$$

while the probability for continuing to play to the next cut would be

$$\frac{1}{1 + \sum_i e^{c_i}} \quad .$$

In order to avoid reaching the end of the example, a penalty for jumping or playing too close to the end may be included in the error metric, and cuts leading into the last part of the example may be deleted.

If the example is annotated with meta-information such as the perceived mood of different parts of a piece of music, the error metric can easily be adapted to enforce appropriate music to be played when, for example, dramatic game events occur.

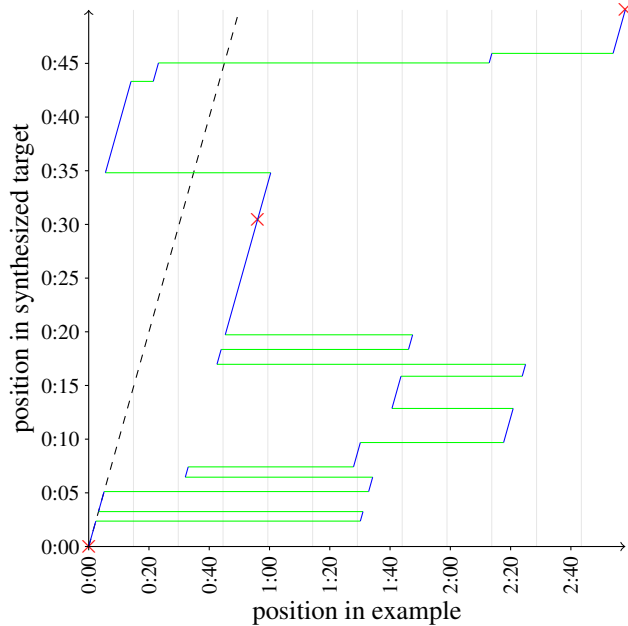


Fig. 3. Path plot for the synchronized soundtrack for the Charlie Chaplin video. The blue diagonal lines indicate which sample in the synthesized target (ordinate) is copied from which sample in the example (abscissa), while the green horizontal lines indicate jumps in the example during playback. The red crosses mark the key frame constraints that have been approximated by the path. The dashed line represents unmodified reproduction of the example for comparison. The grid lines indicate ten times the block size used in the lowest analysis resolution level.

6. RESULTS

In order to evaluate the applicability of our method in a practical context, we begin by generating a novel punk-rock soundtrack for a video sequence from a classic Charlie Chaplin movie (provided as supplementary material). Three key frames of the sequence are synchronized with corresponding samples from the audio example (Fig. 1). The beginning and end of the video sequence are set to the beginning and end of the song, respectively. A third key frame (Fig. 1(b)) marks the beginning of a fight between the two characters; it is set to the start of an intense instrumental sequence in the background music.

Fig. 2 shows the possible cuts that have been found in the punk rock sound example. Every dot in the diagram corresponds to a pair of positions in the sound example between which we are able to jump during playback with minimal perceptual error. The multi-scale algorithm has managed to distribute the cut positions quite evenly. However, diagonal clusters of cuts have not been entirely avoided. They represent cuts of the same length from similar positions in the

example. Since the effects of these cuts on the synthesized soundtrack is very similar, they might as well be collapsed into a single cut in order to speed up the subsequent path search. From looking at diagonal clusters, it also becomes obvious that even though many good cuts have been found, the set of different cut lengths is quite small. This is mostly due to the structure of the song: the length of a good cut is likely a multiple of the length of one beat, one measure, or even an entire verse or chorus. This effect becomes important during the path search step, because it implies that some constraints may not be satisfiable when the key points are too close together. For the “punk rock” example, our experiments indicate that the minimal distance between two key frames lies somewhere between 10 and 30 seconds, depending on the position of the constraints within the example. The attainable precision of the key point constraints depends on the structure of the example and the number of detected cut points, which may become prohibitively large when high accuracy is desired.

On a 3 GHz AMD Athlon 64, finding 256 cuts in the 2:58 long “punk rock” example took about twelve seconds.

In Fig. 3, the optimal path through the example subject to the three key frame constraints is displayed. On the horizontal axis, the position in the example sequence is shown, while the vertical axis represents the position in the synthesis result. The blue diagonal lines represent example sections that have been copied verbatim into the result, while the horizontal green lines show jumps within the example during playback. The red markers indicate the key frame constraint positions which require the synthesized samples at these key frames to be taken from specified places in the example. It becomes apparent that the path sometimes jumps back and forth between two parallel diagonals, e.g. two verses in the song. This is probably due to a very low penalty for cuts in the error function, Eq. (1). However, when listening to the song, these cuts are only noticeable because of obvious inconsistencies in the vocal part.

The time for finding valid paths varies strongly with the desired duration of the output, and with the number and position of the cuts in the vicinity of the desired start and end point in the example. For example, finding 32 complete paths of 30.5 s duration between positions 0:00 and 0:56 in the “punk rock” example took only 0.25 seconds, while for 32 paths of 19.5 s duration between 0:56 and the end of the example, it took about 68 seconds.

Synthesis results for different musical styles and for texture-like sounds are provided as supplementary material. For the latter, we also provide the results by Parker et al. [8] for comparison. In both cases, the length of the synthesized segment was prescribed as a constraint.

The fact that semantic content, like the text in the vocal part of the song or complex musical structures, is not detected by the cut point search algorithm is one of the biggest limitations of our method, while simple, reoccurring large-scale

structures, such as the verse and chorus of a pop song, are often matched surprisingly well, which can lead to interesting and often funny results when a jump occurs from the middle of one verse into another, for example in the “folk” example around 1:34 and 3:15. In general, however, semantic analysis (e.g. [11]) would be necessary in order to reliably produce convincing results for music with vocals. Other semantic breaks can be noticed, for example, in the “punk rock” example around 1:00, 3:14 and 3:55. More severe rhythmic inaccuracies occur in the “folk” example, which lacks a well-defined regular beat, at positions 2:53 and 3:28. Near the end of the video example, the tight constraints have forced a noticeably bad cut to be followed, which becomes obvious when the singer’s voice suddenly disappears.

7. CONCLUSION

We have presented an example-driven algorithm for constrained audio synthesis that works efficiently and with a minimum amount of user interaction for both musical and texture-like examples. Content-aware soundtracks for films and other pre-generated media can be created from a single audio example, such as an MP3 file, and a list of key point – audio sample mappings. For interactive media such as video games or art installations, live audio streams that adapt to the scene contents, which are currently typically created from hand-crafted audio loops or MIDI files, can be generated from arbitrary audio examples.

In contrast to existing approaches, the proposed method produces appealing results not only for texture-like sounds, but also for structured examples like music and has been tested on a variety of musical styles.

A possible extension of the proposed algorithm would be to incorporate semantic analysis, such as beat and pitch detection. In addition to possibly improved synthesis quality for difficult examples, this would increase the flexibility of the algorithm by allowing to automatically match the pitch and beat of different songs, so that medleys of otherwise incompatible songs could be generated automatically.

8. REFERENCES

- [1] M. Cardle, S. Brooks, Z. Bar-Joseph, and P. Robinson, “Sound-by-numbers: motion-driven sound synthesis,” in *Proc. 2003 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, 2003, pp. 349–356.
- [2] William Moss, Hengchin Yeh, Jeong-Mo Hong, Ming C. Lin, and Dinesh Manocha, “Sounding liquids: Automatic sound synthesis from fluid simulation,” *ACM Transactions on Graphics*, vol. 29, no. 3, pp. 1–13, 2010.
- [3] Changxi Zheng and Doug L. James, “Rigid-body fracture sound with precomputed soundbanks,” *ACM Transactions on Graphics*, vol. 29, no. 4, pp. 1–13, 2010.
- [4] Nikunj Raghuvanshi, John Snyder, Ravish Mehra, Ming Lin, and Naga Govindaraju, “Precomputed wave simulation for real-time sound propagation of dynamic sources in complex scenes,” *ACM Transactions on Graphics*, vol. 29, no. 4, pp. 1–11, 2010.
- [5] S. Lefebvre, S. Hornus, and A. Lasram, “By-example synthesis of architectural textures,” *ACM Transactions on Graphics*, vol. 29, no. 4, pp. 1–8, 2010.
- [6] L. Lu, S. Li, L. Wen-yin, H.J. Zhang, and Y. Mao, “Audio textures,” in *Proc. IEEE International Conference on Acoustics, Speech, and Signal Processing*, 2002, vol. 2.
- [7] L. Lu, L. Wenyin, and H.J. Zhang, “Audio textures: Theory and applications,” *IEEE Transactions on Speech and Audio Processing*, vol. 12, no. 2, pp. 156–167, 2004.
- [8] JR Parker and B. Behm, “Creating audio textures by example: tiling and stitching,” in *Proc. IEEE International Conference on Acoustics, Speech, and Signal Processing*, 2004, vol. 4.
- [9] M. Cardle, S. Brooks, and P. Robinson, “Audio and user directed sound synthesis,” in *Proc. International Computer Music Conference*, 2003.
- [10] G. Strobl, G. Eckel, D. Rocchesso, and S. le Grazie, “Sound texture modeling: A survey,” in *Proc. 2006 Sound and Music Computing International Conference*, 2006, pp. 61–5.
- [11] J. Paulus, M. Müller, and A. Klapuri, “Audio-based music structure analysis,” in *Proc. 11th International Conference on Music Information Retrieval*, 2010.
- [12] D. Schwarz, “A system for data-driven concatenative sound synthesis,” in *Proc. Digital Audio Effects*, 2000, pp. 97–102.
- [13] D. Schwarz, “The caterpillar system for data-driven concatenative sound synthesis,” in *Proc. Digital Audio Effects*, 2003, pp. 135–140.
- [14] D. Schwarz, “Current research in concatenative sound synthesis,” in *Proc. International Computer Music Conference*, 2005, pp. 9–12.
- [15] D. Schwarz, “Concatenative sound synthesis: The early years,” *Journal of New Music Research*, vol. 35, no. 1, pp. 3–22, 2006.
- [16] M. Goto and Y. Muraoka, “A beat tracking system for acoustic signals of music,” in *Proc. Second ACM International Conference on Multimedia*, 1994, p. 372.