Eurographics/SIGGRAPH Symposium on Computer Animation (2003)
D. Breen, M. Lin (Editors)

Sound-by-Numbers: Motion-Driven Sound Synthesis

M. Cardle[*], S. Brooks[*], Z. Bar-Joseph[†] and P. Robinson[*]

[*]Computer Laboratory, University of Cambridge
[†]MIT Lab for Computer Science

Abstract

We present the first algorithm for automatically generating soundtracks for input animation based on other animations' soundtrack. This technique can greatly simplify the production of soundtracks in computer animation and video by re-targeting existing soundtracks. A segment of source audio is used to train a statistical model which is then used to generate variants of the original audio to fit particular constraints. These constraints can either be specified explicitly by the user in the form of large-scale properties of the sound texture, or determined automatically and semi-automatically by matching similar motion events in a source animation to those in the target animation.

Keywords:
Audio, multimedia, soundtrack, sound synthesis.

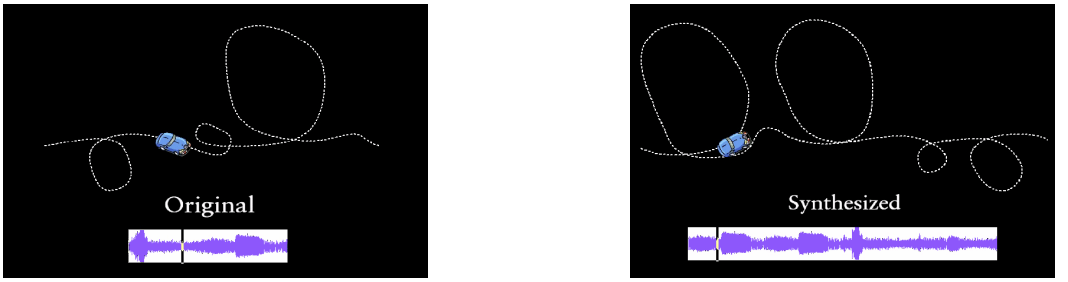

Figure 1 Fully-automated Sound Synthesis: Given a source animation and its associated soundtrack (*Left*), an unseen target animation of the same nature (*Right*) is analyzed to automatically synthesize a new soundtrack, with a high probability of having the same sounds for the same motion events.

1. Introduction

Human perception of scenes in the real world is assisted by sound as well as vision, so effective animations require the correct association of sound and motion. Currently, animators are faced with the daunting task of finding, recording or generating appropriate sound effects and ambiences, and then fastidiously arranging them to fit the animation, or changing the animation to fit the soundtrack.

We present a solution for simple and quick soundtrack creation that generates new, controlled variations on the original sound source, which still bears a strong resemblance to the original, using a controlled stochastic algorithm. Additionally, the motion information available in computer animation, such as motion curves, is used to constrain the sound synthesis process. Our system supports many types of soundtrack, ranging from discrete sound effects, to certain music types and sound ambiences used to emphasize moods or emotions. In order to support such a wide variety of sounds, we present an algorithm which extends the granular audio synthesis method developed by Bar-Joseph et al [2] by adding control to the synthesized sounds.

In the simplest case, users manually indicate large-scale properties of the new sound to fit an arbitrary animation or video. This is done by manually specifying

which types of sounds in the original audio are to appear where in the new soundtrack. A controllable statistical model is extracted from the original soundtrack and a new sound instance is generated that best fits the user constraints. The information in the animation's motion curves is used to facilitate the process. The user selects a sound segment that is to be associated with a motion event. Doing this for a single example enables all subsequent similar motion events to trigger the chosen sound(s), whilst seamlessly preserving the nature of the original soundtrack. For example, the animator might want to apply the sound of one car skidding to several cars being animated in a race without having to separate it from other background racing sounds.

Next, we extend this method, and present a completely automated algorithm for synthesizing sounds for input animations. The user need provide only a source animation and its associated soundtrack. Given a different target animation of the same nature, we find the closest matches in the source motion to the target motion, and assign the matches' associated sound events as constraints to the synthesis of the target animation's new soundtrack (Figure 1).

An advantage of our method is that it provides a very natural means of specifying soundtracks. Rather than creating a soundtrack from scratch, broad user specifications such as "more of this sound and less of that sound" are possible. Alternatively, a user can simply supply a sample animation (along with its soundtrack) and, given a new animation, say, in effect: "Make it sound like this". Finally, this significantly simplifies existing soundtrack recycling since no editing, wearisome looping, re-mixing or imprecise sound source separation is necessary. Sound-by-Numbers operates in a similar fashion to a children's paint-by-numbers kit. But instead of solid colors, or textures in Texture-by-Numbers [10], sound is automatically synthesized into the corresponding mapping.

Our method extends the Bar-Joseph et al.[2] algorithm (abbreviated as BJ below) of which a brief overview is given in Section 3. In section 4 and 5, we present our user-directed approach and explain three types of intuitive user-control. We show the results we obtain and further discuss the algorithm and its limitations in Section 6. We conclude by outlining some possible extensions of this work.

2. Previous Work

There are several existing methods for automating the task of soundtrack generation for animation.

Early work by Terzopoulos and Fleischer [24] used triggered pre-recorded sounds to model the sound of a tearing cloth. A more general approach introduced by Hahn and Hesham [8] ties motion parameters to parameterizable sound constructs, known as Timbre-trees. Control over the musical soundtrack generation process by animation was examined in Nakamura et al. [14], and a more automated approach, which uses motion to directly control MIDI and C-Sound based soundtracks, was carried out in Hahn et al. [9] and Mishra and Hahn [13]. These latter approaches assume an implicit model of sound, contrarily to our method that operates directly at the audio sample level of any existing soundtrack.

More physically-based approaches to motion-driven sound events enable real-time realistic synthesis of interaction sounds such as collision sounds and continuous contact sounds [23,26,16,25,15]. Our method differs from these previous approaches, in that sounds are not synthesized from scratch and no physical models of sound or motion are necessary. We simply use existing soundtracks and re-synthesize them in a controlled manner to synchronize them to new animations at a coarser-level than the previous physically-based approaches. In some respects, our system complements these previous approaches by allowing the definition of more loosely defined relationships between sound and motion. A physically-based approach can be used to generate exact collision sounds,while in parallel, our system can be used to build on existing soundtracks.

The inspiration for the present work and the basis for our soundtrack generation process is the work by Bar-Joseph et al. [2,6]. Their system uses the concept of granular synthesis [18] where complex sounds are created by combining thousands of brief acoustical events. Analysis and synthesis of sounds is carried out in a wavelet-based time-frequency representation. While the BJ algorithm works well on both stochastic and periodic sound textures, it does not provide any control over the new instances of the sound texture it generates, and it is not clear how it can be applied to automatically synthesize complete soundtracks for input animations. In this paper we present a revised version of the BJ algorithm, which allows us to control not only the location (across the animation sequence) where the new synthesized sound is to be located, but also the transition between two different sound textures. In addition, we combine our new synthesis algorithm with an animation alignment algorithm to automatically synthesize new sound segments for input animations. Another approach for sound texture generation is presented in Hoskinson and Pai [11]. Though this approach successfully synthesizes quality sound textures, it uses coarser 'natural-grains' and thus is less appropriate for our purposes where a finer-grained representation is preferable since it is more controllable.

An example of video-guided audio synthesis is presented in Schödl et al. [19], where sound is added to their video textures by cross-fading and playing the sound samples associated with each video frame in the original video. Limited audio continuity is supported since the

arrangements of audio samples are based on the video synthesis, and not on the properties of the soundtrack. This limits the method to highly self-similar and stochastic sounds the potential introduction of sonic artifacts due to the frequently utilized multi-way cross-fading algorithm. We have shown in our video that we can support much more complex and less self-similar sounds then Schödl et al.'s system (who only applied their method on two soundtracks). We therefore believe that the method presented in this paper is more general, and applicable to many more sound types.

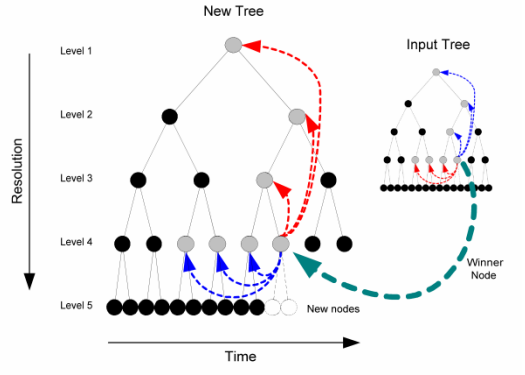

Figure 2 BJ Synthesis step during the construction of a portion of multi-resolution wavelet-tree: Level 5 nodes in the new tree are synthesized by stepping through each parent node at level 4. For each node in level 4, we find a winning candidate, in the input tree, that depends on its scale ancestors (upper levels, pointed at in blue) and temporal predecessors in the same level (those to its left on level 4, pointed at in red). The children of the winning candidate are then copied onto the corresponding positions at level 5.

3. Bar-Joseph Sound Synthesis

To generate new sound textures, the BJ algorithm treats the input sound as a sample of a stochastic process. This is accomplished by first building a tree representing a hierarchical wavelet transform of the input sound, and then learning and sampling the conditional probabilities of the paths in the original tree. The inverse wavelet transform of the resultant tree yields a new instance of the input sound.

The multi-resolution output tree is generated by choosing wavelet coefficients, or nodes, representing parts of the sample only when they are similar. Each new node of the output wavelet tree is generated level-by-level and node-by-node from the left to the right, starting from the root node. At each step, a wavelet coefficient is chosen from the source wavelet tree such that its node's ancestors and predecessors are most similar with respect to the current new node in the sound being synthesized (Figure 2). Wavelet coefficients from the same level are considered as potential candidates to replace it if they have similar temporal predecessors (i.e. the nodes to the left on the same level) and scale ancestors (i.e. the upper-level, coarser wavelet coefficient). Two nodes are considered similar when the absolute difference between their respective ancestors' and predecessors' wavelet coefficients is below a certain user-defined threshold δ. A small value of δ ensures similarity to the input and a large value allows more randomness.

A nodal match is found by first searching all the nodes at the current synthesized tree level for nodes with the maximum number of ancestors within the difference threshold δ. This initial candidate set C_{anc} is further reduced to candidate set C_{pred}, by retaining only the nodes from C_{anc} with the maximum number of up to k predecessors within the difference threshold δ (where k is typically set to 5). The winning node is randomly chosen by uniformly sampling from candidate set C_{pred}. Complete details of the algorithm can be found in Dubnov et al. [6]

4. Directed Sound Synthesis

The BJ algorithm works well with almost no artifacts on both stochastic and periodic sound textures. However, no control is possible over new instances of a sound texture since they are by definition random. We now introduce high-level user-control over the synthesis process. This is achieved by enabling the user to specify which types of sounds from the input sound should occur when, and for how long, in the output synthesized sound. These user-preferences translate into either hard or soft constraints during synthesis. In this section, we first look at how these synthesis constraints are defined, and then, by what means they are enforced in the modified BJ algorithm.

4.1. Constraint Specification

In order to synthesize points of interest in the soundtrack, the animator must identify the synthesis constraints. First, the user selects a source segment in the sample sound such as an explosion in a battle soundtrack (Figure 4). Secondly, the user specifies a target segment indicating when, and for how long, in the synthesized sound the explosion(s) can be heard. The constraints for the rest of the new soundtrack can be left unspecified, so that in our video example, a battle-like sound ambience will surround the constrained explosion.

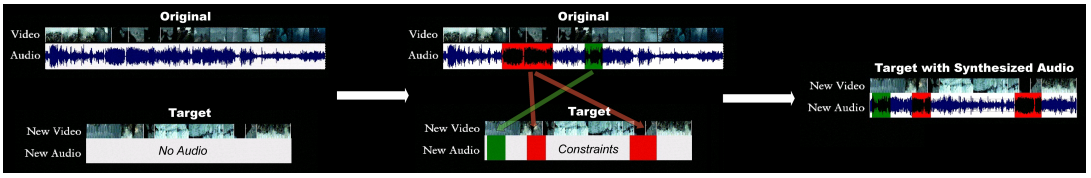

Figure 3 Soundtrack Synthesis for a Video sequence: The target video (*Left-Bottom*) is a rearranged soundless version of the source video (*Left-Top*). The explosion sounds in green, along machine gun sounds in red (*Middle-Top*), are defined as synthesis constraints in the target soundtrack (*Middle-Bottom*). These constraints are used to guide directed sound synthesis into generating the appropriate soundtrack for the target video (*Right*).

The source and target segments, each defined by a start and end time, are directly specified by the user on a familiar graphical *amplitude* x *time* sound representation. Since the target soundtrack has yet to be synthesized and therefore no amplitude information is available, target segments are selected on a blank amplitude timeline of the length of the intended sound. Note that the number, length and combinations of source and target segments are unrestricted, and that exclusion constraints can also be specified so as to prevent certain sounds from occurring at specific locations.

The user can associate a probability with each constraint, controlling its influence on the final sound. To this end, a weighting curve is assigned to each target segment, designating the probability of its associated source segment(s) occurring at every point in the target area. The weights vary from [-1, 1], where -1 and 1 are equivalent to hard-constraints guaranteeing, respectively, exclusion or inclusion. Soft-constraints are defined in the weight ranges (-1,0) and (0,1) specifying the degree with which exclusion or inclusion, respectively, is enforced. Furthermore, the reserved weight of 0 corresponds to unconstrained synthesis.

The special case, when inclusion targets with different sources overlap, is dealt with by selecting the target with the highest current weighting. For consistency, the user is prevented from defining overlapping hard-constraints.

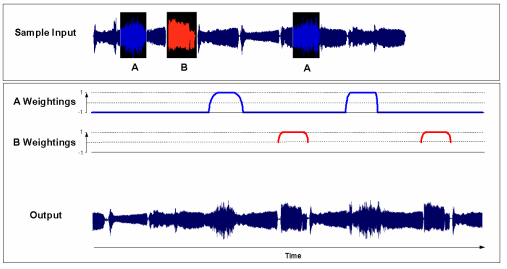

Figure 4 (*Top*) Source regions A and B. (*Middle*) Weighting curve for A and B. (*Bottom*) Directed synthesis output.

In order to use these constraints in our algorithm, we need to extract all leaf and subsequent parent nodes of the wavelet tree involved in synthesizing these source and target segments. To each source and target segment(s) combination we assign a unique constraint identifier. For example, the explosion constraints will have a different identifier to the gun-shot constraints. Using this we build two node-lists, S and T, which contain the tree level and position offset of all nodes in, respectively, the source and target wavelet-tree involved in the constraint specification. T additionally contains the constraint weight associated to each node. During the directed synthesis process, if the currently synthesized node is defined in T, its associated constraint identifier in T determines which nodes from S, and subsequently in the input wavelet-tree, should be used as potential candidates.

4.2. Hard and Soft Constrained Synthesis

Now that we know the source origins of every node at every level in the target tree, we can modify the BJ algorithm to take these constraints into account. In addition to enforcing similar ancestors and predecessors, successor restrictions are imposed. Successor nodes are defined as the neighboring nodes appearing forward in time at the same tree level. Similarly to predecessor nodes, these can be rather distant in terms of their tree graph topology.

Let d be the successor look-ahead distance defined as $d = 2^l \times k$, where l varies from 0 to n corresponding respectively to the root and leaf levels of the tree, and k is a user constant defining the anticipation strength (typically set to 5%). In this manner, d is kept consistent at different scales. We use d to split up the space of all nodes into the set of constrained and unconstrained nodes before carrying out matching on every node.

Hence, if the currently synthesized node belongs to T or has its d-th successor in T, or both, then W is the corresponding set of candidates inside and outside S satisfying the same constraint identifier conditions. Let all remaining nodes be contained in the set V. Nodal matching is then separately carried out on both W and V in parallel, resulting in two candidate sets C_{pred}^{W} and

C^{V}_{pred}, defined, respectively as the constrained candidate set and the unconstrained candidate set. They define the best matching candidates for both the constrained and unconstrained sets. The winning node is then randomly chosen by non-uniformly sampling from $C^{W}_{pred} \cup C^{V}_{pred}$. Nodes in C^{V}_{pred} are given the default weight of 0.1 whereas the ones in C^{W}_{pred} are all given the weight of *T*'s current weighting. Depending on *T*'s weight value, this has the effect of biasing the selection process in favor or against the nodes in C^{W}_{pred}. If *T*'s current weight is a hard-constrained 1, then the winner is picked by uniform sampling from C^{W}_{pred} only.

While the above algorithm works fine in most cases, Sometimes the quality of the matches found in C^{W}_{pred} might be inferior to those found in C^{V}_{pred} due to the reduced search space. We therefore want to prevent significantly inferior matches from C^{W}_{pred} being chosen in order to maximize audio quality. This is controlled by ensuring that the sum of the maximum number of found ancestors in C^{W}_{anc} and predecessors in C^{W}_{pred} is within a user- percentage threshold *r* of that of C^{V}_{anc} and C^{V}_{pred}. Let m_W and n_W be, respectively, the number of ancestors and predecessors for the best candidates currently in C^{W}_{anc} and C^{W}_{pred} within the randomness threshold δ. Let m_V and n_V be their equivalent in C^{V}_{anc} and C^{V}_{pred}, then if $(m_W + n_W) < (m_V + n_V) \times r$ then the candidates from W are discarded (*r* is usually set to 70%). The threshold *r* controls the degree with which soft-constraints are enforced at the cost of audio quality. We adopt a different strategy for hard-constraints as explained below.

In the naïve BJ algorithm, initial candidates include all nodes at the current level. Doing this over the whole tree results in a quadratic number of checks. Hence, a greatly reduced search space is obtained by limiting the search to the children of the candidate set of nodes of the parent. However, on the borderline between unconstrained and hard-constrained areas, the reduced candidate might result in C^{W}_{pred} being empty, since no node is within the imposed threshold limits. Consequently, in our algorithm, if no candidates are found in C^{W}_{pred} whilst in hard-constrained inclusion mode, a full search is conducted in *S*. If matches within the threshold tolerance still cannot be found, the best approximate match in *S* is utilized instead.

5. Automated User Control

In this section, we use the algorithm described in Section 4 to present three different user-interaction methods to specify the synthesis constraints. These include manual, semi-automatic and fully automatic constraint definition.

5.1. Manual Control

The user starts by specifying one or more source regions in the sample sound. In the example depicted in Figure 4, two distinct source regions are defined corresponding to areas A and B (top). Note that A is defined by two segments. The user then draws the target probability curve for both sources A and B directly on the timeline of the new sound. A's weightings are zero except for two sections where short and smooth soft-constraints lead to a 1-valued hard-constraint plateau. This results in region A smoothly appearing twice, and nowhere else. On the other hand, B's curve also defines two occurrences but is undefined elsewhere, imposing no restrictions. Thus sounds from B might be heard elsewhere.

5.2. Semi-Automatic Control

In this mode of interaction, the motion data in the target animation is used. The user associates sound segments with motion events so that recurring similar motion events trigger the same sounds. We detect all these recurring motion events, if any, by finding all motion segments similar to the query motion, the number of which is controlled by an adjustable similarity distance threshold θ.

We support matching over 1D time-varying motion curves such as 1D position or angle variations. Matches that are non-overlapping and up to *L* times longer, and *H* times shorter, than the query motion are retained (usually *L* is set to 2 and *H* to 0.5). Similarity is calculated across the entire target motion for sliding windows ranging from *L* to *H* in size. Similarity is determined by applying the Iterative Deepening Dynamic Time Warping (IDDTW) distance measure [5], a fast variant of Bruderlin and Williams's [3] original Dynamic Time Warping (DTW) on motion. By squeezing and stretching motions before calculating their similarity, DTW produces better measure of similarity between two motions because it is not as sensitive to small distortions in the time axis as the Euclidean distance. We recently have added support for more matching primitives such as 2D and 3D motions, as well as over the complete skeleton in motion capture. Further details can be found in Cardle et al. [4].

By default, each motion that matches the user-selected motion is given the same weight in the synthesis. Alternatively, the synthesis weightings can be made proportional to the strength of the corresponding matches. The effect is that strong motion matches will

have high probability of having the same audio properties as the query, and inversely for weak matches. Let s_x be the similarity measure of a current match x, w_q the query's audio weight (set by the user) and c, a user percentage, then x's audio strength, w_x, is defined as:

$$w_x = w_q\left(\left(\frac{s_x-\theta}{1-\theta}\right)c+1-c\right)$$

By modifying the value of c, the animator can modulate the effect of the matching strength on the resulting audio strength.

Our interface is further automated by performing audio matching to find similar audio segments to the query sound segment in the rest of the sample soundtrack. These audio matches, along with the query audio, are combined to form the same source audio segment for the motion matches. By performing sound-spotting audio matching [21], perceptually similar non-overlapping audio segments to the query are found in the rest of the soundtrack. An interface slider enables the animator to control the number of returned audio matches. This is especially valuable for selecting frequently recurring sounds over extended soundtracks.

5.3. Fully-Automatic Control

In contrast to the approaches above, this method requires practically no user-intervention beyond providing the following inputs: a sample animation with its soundtrack and a different animation, preferably of the same nature. After the user specifies the 'steering' motion track, a new soundtrack is automatically synthesized with high probability of having the same sounds for the same motion events as those in the sample animation.

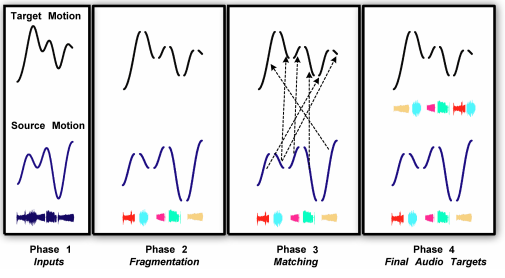

Figure 5 Phases involved in fully-automatic control: (*Phase 1*) The source motion, its soundtrack and the target motion are entered. (*Phase 2*) Both motions are broken up at sign changes in first derivative. (*Phase 3*) Matches between the source and target motion fragments are found. (*Phase4*) Each fragment in the target motion now has a source audio segment assigned.

We therefore need to determine which portions of the source motion best match with those in the new, target motion. This is achieved by using the motion matching algorithm recently presented in Pullen and Bregler [17]. The algorithm is depicted in Figure 5. The motion curve is broken into segments where the sign of the first derivative changes. For better results, a low-pass filter is applied to remove noise beforehand. All of the fragments of the (smoothed and segmented) target motion are considered one-by-one, and for each we select whichever fragment of source motion is most similar. To achieve this comparison, the source motion fragments are stretched or compressed in time to be the same length as the target motion fragment. This yields the K closest matches for each target fragment. An optimal path is found through these possible choices to create a single stream of fragments. The calculated path maximizes the instances of consecutive fragments as well maximizing the selection of the top K closest matches [17]. We then assign the audio of the matching source fragment to that of the target fragment.

At the end of this process, every target fragment has been assigned a single source audio segment taken from its matching source fragment. From this, an index of source/target constraint combinations is constructed by merging targets with overlapping sources. In practice, soft constraints, where preferred nodes have weights just over 0, give the best results. This is because hard-constraints can produce artifacts if used extensively. Furthermore, additional weighting is given to perceptually significant sounds so as to increase the likelihood that they will be enforced. Weights are therefore made proportional to the average RMS volume over the entire audio segment. Louder sounds usually attract more attention and therefore should have higher priority.

The resulting target soundtrack is usually of a higher quality if there are sections in time where fragments that were consecutive in the source data are used consecutively to create the path. This is accommodated by Pullen and Bregler's [17] method as the algorithm considers the neighbors of each fragment, and searches for paths that maximize the use of consecutive fragments.

By breaking up the motion at first derivative sign changes, we enforce better audio continuity over portions of motion that are constant. On the other hand, segmentation based on second derivative changes, or inflexion points, gives better audio continuity at changes in the motion. Consequently, our system simply generates two soundtracks, one for each segmentation

strategy, and the animator picks whichever best fits his or her expectations.

6. Results and Discussion

We now present several applications of Sound-by-Numbers. The examples are in the accompanying video as printed figures could not convey our results meaningfully.

The first example illustrates the use of manual control to derive a new sound track from an exsiting one when the corresponding video sequence is edited. Only a few seconds were necessary to synthesize 30 seconds of output at 32 KHz on a 1.8Ghz processor. The synthesis time increases with the length of the synthesized target sound and its sampling frequency.

In our next example semi-automatic control is used to produce the soundtrack of a flying bird animation. All similar flight patterns to the user selected ones are assigned to the same target sounds, whilst the rest of the soundtrack conveys the jungle background sounds. Notice that no complex audio editing was required here, just a few mouse clicks. Not surprisingly, better results are obtained if the source and targets regions are similar in length, otherwise unexpected results occur. For example, a laughing sequence sounds unnatural if it is prolonged for too long using hard-constraints. We also found that the directed synthesis very occasionally switches to approximate matching (maybe for one node out of thousands for the example here) so sound quality is rarely adversely affected. This can be eliminated by activating the automatic padding of a short soft-constraint gradation before and after hard-constraint segment boundaries. Repetitions in the synthesized sounds can be discouraged by simply lowering the weights of candidates that have just been picked. Finally, if the results are not what the user expected, at most a small number of quick iterations are required to produce a soundtrack that better accommodates the user's intentions.

Our final example takes advantage of our fully automated approach to generate new racing car sounds to accompany the edited version of the original racing car animation.

7. Limitations

Good candidates input sources are sounds that allow themselves to be manually re-arranged without incurring perceptual problems. This is especially true of traffic sounds, crowd recordings, jazz and ambient music, audience laughing and collections of short sounds where maintaining the long-term order of occurrence is not essential. Sounds with clear progressions, such as a very slowly increasing siren sound, cannot be meaningfully rearranged by hand, and therefore, cannot also be done by our algorithm. Similarly, our method is not appropriate for speech processing including human singing. This does not invalidate our work since the supported sound types still cover a wide variety of video and animation segments.

8. Conclusion and Future Work

In this paper we have introduced a fully automatic algorithm for generating soundtracks for an input animation based on other animations soundtracks. In order to achieve this, we have described a new sound synthesis algorithm capable of taking into account users' preferences, whilst producing high-quality output. Multiple interaction modes provide a variety of user intervention levels ranging from precise to more general control of the synthesized soundtrack. Ultimately, the fully-automated method provides users with a quick and intuitive way to produce soundtracks for computer animations.

Although it is feasible to use conventional audio software, such as ProTools®, to manually generate very short soundtracks, it rapidly becomes impractical for the production of extended soundtracks. Re-arranging the whole soundtrack so as to produce constantly varying versions of the original would quickly become cumbersome in ProTools®. Also, in our system, constraints can be later changed on-the-fly leading to a full re-synthesis of the soundtrack.

There are still many opportunities for future work such as the integration of our system into video textures [19] and motion synthesis algorithms [17,1,12], as well as real-time operation.

Currently, overlapping target regions are dealt with by choosing the highest probable target. Sound morphing techniques [20] could be applied to combine source sounds in overlapping regions according to their weightings. Additionally, identifying quasi-silent portions [22] and giving them lower priority during constrained synthesis would improve constraints satisfaction and therefore control.

9. Acknowledgements

We thank Loic Barthe, Neil Dodgson and Carsten Moenning for providing valuable feedback. This work is supported by the UK Engineering and Physical Sciences Research Council and the Cambridge Commonwealth Trust.

References

1. Arikan, O., and Forsyth, D.A. 2002. Interactive Motion Generation From Examples. In *Proceedings of ACM SIGGRAPH 2002*, San Antonio, Texas, August 22-27.
2. Bar-Joseph, Z., Lischinski, D., Werman, M., Dubnov, S., AND EL-YANIV, R. 1999. Granular Synthesis of Sound Textures using Statistical Learning. *International Computer Music Conference*, 178-181.
3. Bruderlin, A., and Williams, L. 1995. Motion signal processing. In *Proceedings of ACM SIGGRAPH 1995*, 97-104.
4. Cardle, M., Vlachos, M., Brooks, S., Keogh, E. and Gunopulos, D., 2003. Fast Motion Capture Matching with Replicated Motion Editing. *Proceedings of ACM SIGGRAPH 2003 Technical Sketches*.
5. Chu, S., Keogh, E., Hart, D. and Pazzani, M. 2002. Iterative Deepening Dynamic Time Warping. In *Second SIAM International Conference on Data Mining*.
6. Dubnov, S., Bar-Joseph, Z., El-Yaniv, R., Lischinski, D., and Werman, M. 2002. Synthesizing sound textures through wavelet tree learning. In *IEEE Computer Graphics and Applications*, 22(4), 38-48.
7. Foote, J. 2000. ARTHUR: Retrieving Orchestral Music by Long-Term Structure. In *Proceedings of the International Symposium on Music Information Retrieval*, Plymouth, Massachusetts.
8. Hahn, J., Hesham, F., Gritz, L., and Lee, J.W. 1995. Integrating Sounds in Virtual Environments, *Presence Journal*.
9. Hahn, J., Geigel, J., Lee, J.W., Gritz, L., Takala, T., and Mishra, S. 1995. An Integrated Approach to Sound and Motion, *Journal of Visualization and Computer Animation*, Volume 6, Issue No. 2, 109-123.
10. Hertzman, A., Javobs, C., Oliver, N., Curless, B., and Salesin, D. H. 2001. Image Analogies. In *Proceedings of SIGGRAPH 2001*, Computer Graphics Proceedings, Annual Conference Series, 327-340.
11. Hoskinson, R., and Pai, D., 2001. Manipulation and Resynthesis with Natural Grains. In *Proceedings of the International Computer Music Conference 2001*.
12. Li, Y., Wang, T., and Shum, H-Y. 2002. Motion Textures: A Two-Level Statistical Model for Character Motion Synthesis. In *Proceedings of ACM SIGGRAPH 2002*, San Antonio, Texas, August 22-27.
13. Mishra, S., and Hahn, J. 1995. Mapping Motion to Sound and Music in Computer Animation and VE, Invited Paper, *Pacific graphics*, Seoul, Korea, August 21-August 24.
14. Nakamura, J., Kaku, T., Hyun, K., Noma, T., and Yoshida, S. 1994, Automatic Background Music Generation based on Actors' Mood and Motions, *Journal of Visualization and Computer Animation*, Vol. 5, No. 4, 247-264.
15. O'Brien, J. F., Shen, C., and Gatchalian, C. M., 2002. Synthesizing Sounds from Rigid-Body Simulations. In *Proceedings of ACM SIGGRAPH 2002 Symposium on Computer Animation*.
16. O'Brien, J. F., Cook, P. R., and Essl G., 2001. Synthesizing Sounds from Physically Based Motion. In *Proceedings of SIGGRAPH 2001*, Computer Graphics Proceedings, Annual Conference Series, August 11-17.
17. Pullen, K., and Bregler, C. 2002. Motion Capture Assisted Animation: Texturing and Synthesis, In *Proceedings of ACM SIGGRAPH 2002*, San Antonio, Texas, August 22-27.
18. Roads, C. 1988. Introduction to granular synthesis, *Computer Music Journal*, 12(2):11–13.
19. Schödl, A., Szeliski, R., Salesin, and D., Essa, I. 2000. Video textures. In *Proceedings of SIGGRAPH 2000*, pages 489-498, July.
20. Serra , X., AND Bonada, J., Herrera, P., and Loureiro, R. 1997. Integrating Complementary Spectral Models in the Design of a Musical Synthesizer, *Proceedings of the International Computer Music Conference*.
21. Spevak, C., and Polfreman, R., 2001. Sound spotting - A frame-based approach, *Proc. of the Second Annual International Symposium on Music Information Retrieval: ISMIR 2001*.
22. Tadamura, K., and Nakamae, E. 1998. Synchronizing Computer Graphics Animation and Audio, *IEEE Multimedia*, October-December, Vol. 5, No. 4.
23. Takala, T., and Hahn, J. 1992. Sound rendering. In *Proceedings of SIGGRAPH 92, Computer Graphics Proceedings*, Annual Conference Series, 211-220.
24. Terzopoulos, D., and Fleischer, K. 1988. Deformable models. *The Visual Computer*, 4, 6, 306-331.
25. van den Doel, K., Kry, P. G., and Pai, D. K. 2001. *Foley automatic: Physically-based sound effects for interactive simulation and animation*. In *Proceedings of SIGGRAPH 2001*, Computer Graphics Proceedings, Annual Conference Series, 537–544.
26. van den Doel, K., AND Pai, D. K. 1996. Synthesis of shape dependent sounds with physical modeling. In *Proceedings of the International Conference on Auditory Display*.