Technical Report

UCAM-CL-TR-XXX
ISSN XXX-XXX

Number XXX

UNIVERSITY OF CAMBRIDGE
Computer Laboratory

Automated Sound Editing

Marc Cardle
King's College

May 2004

15 JJ Thomson Avenue
Cambridge CB3 0FD
United Kingdom
phone +44 1223 763500
http://www.cl.cam.ac.uk/

Technical reports published by the University of Cambridge
Computer Laboratory are freely available via the Internet:

http://www.cl.cam.ac.uk/TechReports/

ISSN XXXX-XXXX

ABSTRACT

The goal of this work is to simplify and automate the production and re-use of sound. In sound production, reusing pre-recorded sounds to create new soundtracks is difficult because these sounds are static and might not fit the target context. One option is to find better fitting instances in a sound library. The other is to manually rearrange the sounds to the new context. This is time-consuming, even for a skilled sound editor. In this work, automated techniques are introduced to remodel pre-recorded sounds whilst maintaining their original character. The user is presented with high-level controls over the remodelling process such as "*more of this sound here and less of that sound there*". Additionally, in the case of computer animation, our system allows a user to simply supply a sample animation (along with its soundtrack) and, given a new animation, say, in effect: "*Make it sound like that*". This is achieved by using data-driven sound re-synthesis methods to generate meaningful rearrangements of sounds.

TABLE OF CONTENTS

Chapter 1

INTRODUCTION

1.1 Background

There has been a digital revolution in films and video games during the last 20 years. This revolution is the shift from analogue to digital methods of recording and manipulating visuals and sound. This section examines industrial use of digital sound in production.

1.1.1 Sound

Sound is fundamental to our experience of the real world, and so the role of sound in relation to on-screen images is vital. Contrary to popular assumptions, both acoustic and visual backgrounds contain a large amount of useful and important information. But due to the fact that they are often perceived unconsciously, their importance is not always recognized. However, in some cases, the sound that accompanies visual images can make or break almost any film:

> *Films are 50 percent visual and 50 percent sound. Sometimes sound even overplays the visual.*
>
> David Lynch (Filmmaker) [Home Theater Buyer's Guide 1998]

Surprisingly, if one includes the cost of music, the sound budget for a typical Hollywood film is traditionally only 5% of the total cost of the film [Carlsson 2002]. In video games, sound and music are almost typically done last and work with whatever system resources are left over from graphics and game mechanics [Maitland 2001]. However, this is changing. Technological improvements in sound hardware and better general awareness mean that sound requirements are now becoming part of the initial design of games [Boyd 2002].

Traditional approaches to soundtrack production have concentrated on pre- or post-processing methods [Zaza 1991] with reference to animation and video. In the pre-processing technique, a soundtrack is composed early in development, before the animation or video has been finalized. Using timing information from the soundtrack, motion events and video are then synchronised to sounds.

In post-processing methods, sounds are synchronised to existing visual events. The motions of objects in the animation or the sequence of frames in video, are determined before any sounds are generated. In film and television, the sound synchronisation is often then done manually in a process known as Foleying, named after Jack Foley of Universal Studios who created a way to add sound effects during post-production

using a recording studio (Figure 1). It is conventionally done in a room where the floor is covered with the different types of surfaces such as gravel, wood, asphalt and leaves. The picture is then projected and the Foley artists try to copy the actions on screen as accurately as possible, while their sounds are recorded. Clothing rustle, rain, car hoods, thunder, doors opening, keys rattling and the movement of props such as dishes are likely to be recorded here. Even kisses are Foleyed.

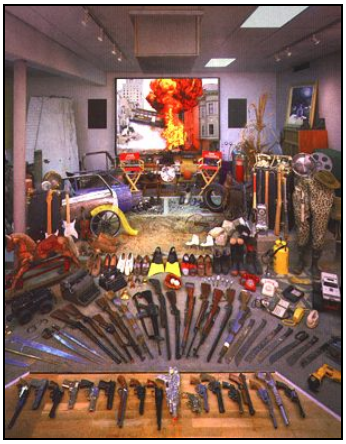

Figure 1: A Foley studio.

Foley does not encompass sounds like explosions, automobile engines, dog barks, or other mechanical entities. This is simply for practical reasons, since driving a car around in the studio or blowing up a building is usually not possible. These sounds are the domain of digital sound effects which draw on pre-recorded libraries. Using sound editing software, these are then layered and mixed into the final sound track.

A deficiency in this approach is that editing the film forces a regeneration of the soundtrack, which can be time-consuming and expensive. Although Foleying and sound effects processing has recently been automated, they remain difficult and laborious tasks. Moreover, few audio signal representation methods are capable of synthesizing everyday sounds for effects in film, television, video games and virtual environments. Our research fits in the post-processing category, with the goal of simplifying the sound production process.

In video games and virtual worlds, the images are constantly changing. Therefore, a common technique is to playback pre-recorded sounds when predefined events occur. This quickly becomes monotonous for frequent events, where the same sounds are repeated for each occurrence. There has been surprisingly little work using the information already present in the animated graphics to drive the generation of sound. It

would be interesting to use that motion data to control the sound generation step to automate the sound design process further.

1.2 Motivation and Aims

Producing content for film, TV, animation, virtual environments or video games is a labour intensive process. This research project arises from the current trend in audio processing and computer graphics communities to simplify the production of audio and visual content. This work endeavours to automate the production of soundtracks and animation whilst maximizing reuse of existing assets to reduce both cost and labour, yet maintain quality. There are two goals to this research:

- **Automate the editing of soundtracks**

  Every sound effect and ambient noise we hear in video or interactive environments, such as video games and virtual worlds, is chosen by a sound designer who carefully sorts through hundreds of pre-recorded sound samples, existing soundtracks or specially recorded sounds. Recorded sounds have fixed timing, so they might not fit the target context. Another option is to edit the sound manually to fit. Such an operation requires retargeting the sound to the new context while keeping its original character. This is time-consuming, even for a skilled sound editor. Our goal is to simplify this process.

- **Automate the editing of sound *to* animation**

  Sounds are generally associated with motion events in the real world. When sounds and motions do not have the proper correspondence, the resulting confusion can lessen the effects of both. Hence, producing effective animations and films requires synchronisation of the sound and motion, which remains an essential, yet difficult, task.

In our other work [Cardle et al. 2002a; Cardle et al. 2002b], we dealt with the problem of using sound to drive alterations to animations to better convey their synchronicity. This was done by locally modifying animations using perceptual cues extracted from the soundtrack. Here, we address the inverse problem: that of modifying sound to fit an animation. The goal is to improve synchronicity in an automated manner whilst maximizing reuse of existing sounds. The general idea is to use the information present in the animation so that recurring events will trigger the same sounds.

It is hard to describe audio and animation verbally or with static graphics, and many of our results should be heard or seen in order to be appreciated. Therefore, we have made several audio and video examples available on the accompanying DVD-ROM.

Chapter 2

BACKGROUND

This chapter provides background information on sound. A survey of the literature regarding sound production for computer animation and related fields is presented. Since the scope of this work spans across a set of comprehensive research fields, the discussion is limited to papers directly related to the present goals.

2.1 Previous Work in Sound Editing

There are several existing methods for automating the task of soundtrack generation for computer animation, video, video-games and virtual reality (VR). The most relevant ones are reviewed and their inadequacies to satisfy our goals are revealed.

2.1.1 Related Approaches

2.1.1.1 Non-linear Sound Editing Tools

In current film and television productions, non-linear sound editing software is used to build up multi-track recordings containing dialogue, music, Foley effects and sound effects. They offer a wide variety of recording, editing, visualisation and filtering operations. For example, *ADR Studio Montage* is a professional plug-in for speech and Foley for the sound editing software *Pro Tools* (Figure 2). It was used extensively in big-budget movies, including *Star Wars Episodes I* and *II*, *The Matrix, The Lord of the Rings* and many television shows.

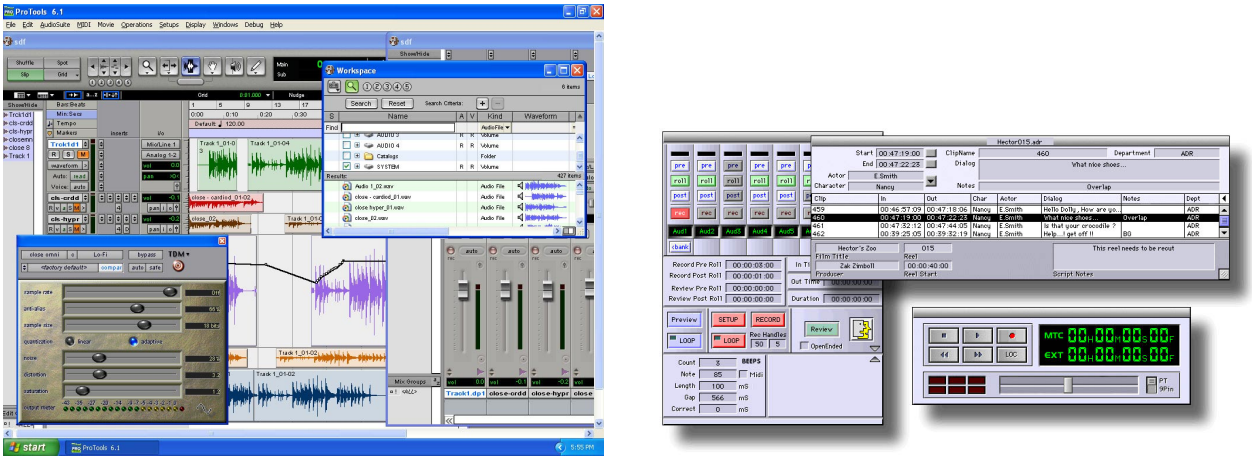

Figure 2: (***Left***) *Pro Tools'* comprehensive multi-track editing and filtering user-interface. (***Right***) *ADR Studio Montage* plug-in for *Pro Tools*.

A regular source of sound effects is a stock library, where digitally sampled waveforms, commonly referred to as sampled sounds, are stored on individual CDs or in complete sets with a common theme. The studio's

sound editor then picks a selection of sounds and, using conventional sound editing software, combines them to fit the target video or animation. If the sound effects do not fit exactly, they must either be re-arranged or recorded. Though this might be straightforward for very short segments, a more automatic approach would be more appropriate for longer segments. Re-arranging a whole soundtrack so as to produce constantly varying versions of the original quickly becomes cumbersome in *Pro Tools*. Also, any changes to the visuals force re-editing of the soundtrack. Note that the conventional approach *is* feasible, as many film production houses use it, though admittedly at a high cost of time and labour. In VR and video-games, this is not feasible due to the constraints of real-time and the necessity of long streams of audio.

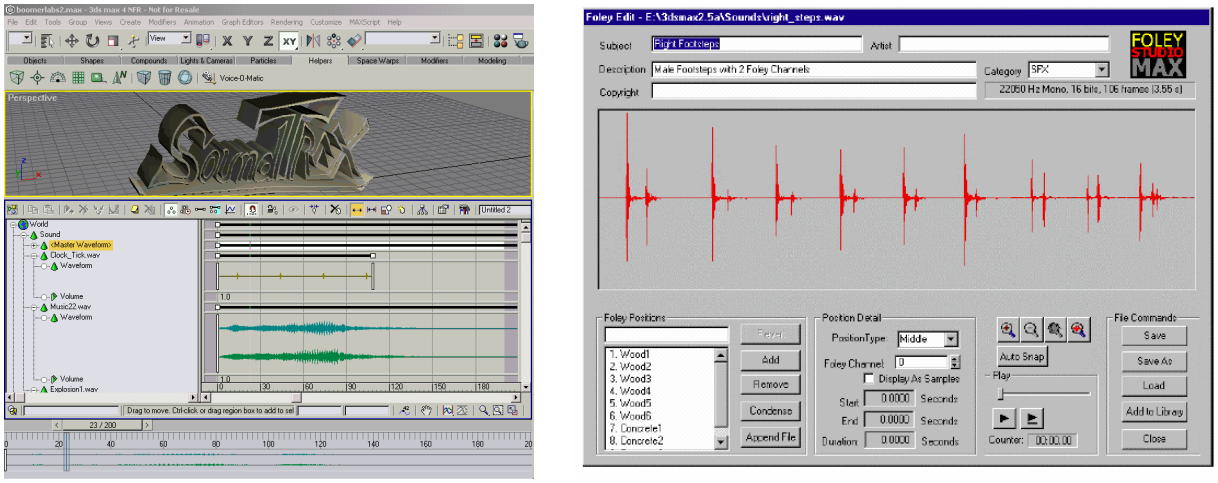

Figure 3: **3D modeller's sound editors.** (***Left***) SoundTrax's plug-in interface is highly integrated into that of *3D Studio Max*. (***Right***) External sample editing dialog of the *Foley Studio* plug-in in *3D Studio Max*.

2.1.1.2 Sound Editors for 3D Modelling and Animation Packages

Standard 3D modelling and animation packages like *3D Studio Max* and *Maya* typically offer limited support for sound. Tools, such as the *SoundTrax* and the *Foley Studio* plug-ins (Figure 3), exist to help 3D animators automate the soundtrack creation process. They do this by making use of data available in the 3D environment and objects populating it. Pre-recorded sounds are attached to objects or particle systems and triggered such as when object collisions are detected (Figure 4). Position, movement and environment data is used to add filtering effects such as 3D positioning, Doppler-shift, distance fades, directional sound sources and echo. Acoustic materials applied to objects can affect the sonic effect of occlusions and collisions. When the animation is changed, the sound events are automatically re-timed and re-filtered. Triggered and looped sounds can be randomized to provide more natural, less repetitive sounds. Unfortunately, the animator has to hand pick the group of static candidate samples and no audio continuity is possible.

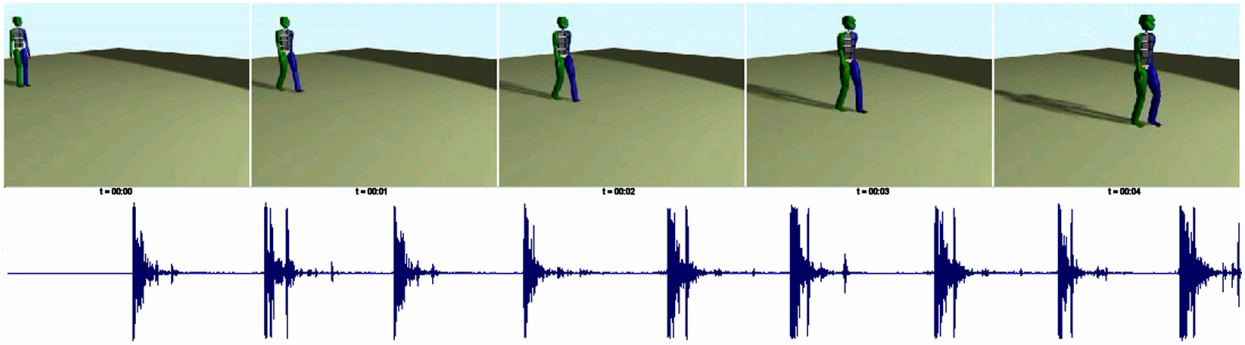

Figure 4: **Foley Studio Output**. A triggered pre-recorded stepping sound is attached to each leg of the avatar. The impact force between the foot and the floor affects the volume of the triggered sound. The animated walk sequence, shown above, generates the synchronized walking sound plotted below.

These plug-ins, as well as current research efforts in sound for VR [Funkhouser et al., 2002], have focused primarily on techniques for localizing sounds. While this is an important problem, it is certainly not the whole picture. They do not address the problem of automatically generating continuously varying variations on a pre-recorded sample, which still bear a strong resemblance to the original, whilst synchronising to animation data.

2.1.1.3 Video-game and Virtual Reality Sound

Like film and television, game audio contains music, speech and sound effects. But games and VR also require interactivity and real-time effects applied to the audio. Most current sound generation systems for games limit their representation of sound sources to sampled sounds. Sampled sounds are computationally inexpensive to play, and in many cases can produce good results with little effort. Synthetic sounds (see below in Section 2.1.1.4), on the other hand, are still computationally expensive to generate and difficult to specify. Game companies therefore often rely on commercial stock CD sound effects libraries for the majority of their raw sound material [Peck 2001]. These sounds can be affected by the structures in the environment such as reverberation or occlusion, or spatially localized, and they can respond to physics (such as Doppler shift). Video games can use the same sample *ad infinitum* during game play. Simple repetition is not effective for long. In the same manner as in 3D modellers, permutations and the randomisation of the sampled sounds are hand-defined so that players do not feel like they are hearing the same repeated sounds.

Figure 5: **Video-game sound**. (***Left***) *Blood Wake* video-game screenshot where randomized chain gun sounds are used. (***Right***) The Orc army sounds from the *Lord of the Rings: The Two Towers* video-game also use randomisation to add variety.

For example, Microsoft's *Blood Wake* game contains player and enemy chain-gun sounds (see Figure 5(left)). The sound designer therefore created two groups of shot sounds with eight variations within each group. The audio programmer then wrote a system that would call these sounds in a quasi-random order. It was random, but was weighted to be less likely to call the same sound twice in a row. A similar approach was used to generate the *Orc* army sounds in the Electronic Arts' recent release of *Lord of the Rings: The Two Towers* video-game [Boyd 2002] (see Figure 5(right)) and in Microsoft's *Xbox Halo* game for the laser-gun sounds [O'Donnell 2002] (Figure 6). When what is desired is simply a controlled variation on the original sample that still bears a strong resemblance to the original, the above audio techniques have critical problems. They require extensive manual intervention along with custom programming.

Figure 6: (***Left***) Custom sound configuration software developed by Microsoft's development team for their *Halo* game. This interface helps set the randomisation and permutations of each utilized sound. (***Right***) *Halo* game-play screenshot.

2.1.1.4 Parameterizable sound constructs

Instead of using triggered sounds, Hahn and Hesham [1995] tie motion parameters to parameterizable sound constructs, known as Timbre-trees (see Figure 7). At the core of the technique is the idea of a Timbre-tree that represents a sound as a tree composed of functional units. These include standard mathematical (+,-,*,/) and signal processing functions (e.g. filtering and convolution) as well as several special-purpose functions useful for sound synthesis, such as a number of elementary waveforms (saw-tooth, triangle, square, sine), sub-trees and several types of noise (white and Perlin noise [Perlin 1985]). The output from the root of the tree is the computed value of the sound for that time sample point. By evaluating a tree with a time dependent set of parameters, the characteristics of the sound can be changed on-the-fly. The parameters associated with the tree are mapped to parameters of the motion such as velocity, direction or angular velocity.

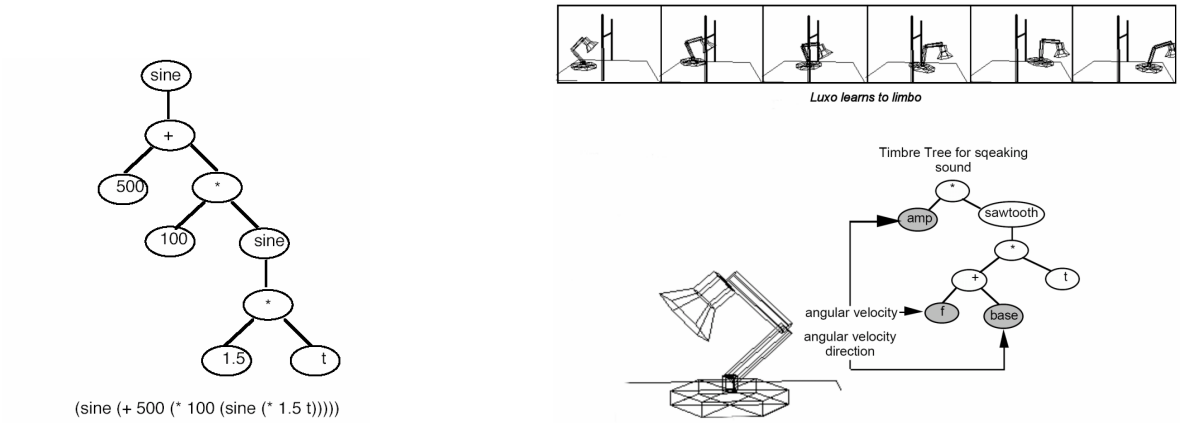

Figure 7: **Timbre-trees**. (***Left***) Timbre-tree for a police siren sound. As the current time parameter *t* varies, so does the siren sound. (***Right***) The motion of Pixar's famous Luxo lamp produces a raspy sound due to its squeaky hinges that varies in pitch and amplitude with angular velocity. Therefore, a sawtooth wave is modulated in frequency and amplitude by the angular velocity. Depending on if the angular motion is clockwise or anti-clockwise, different sounds are heard since the joints have different sounds when opening and closing.

The problem lies in the creation of such parameterized sound structures, and in mapping from the parameter space of the motion domain to that of the sound space. The animator must derive the Timbre-tree of a sound from a general idea of how the sound was produced and then somehow tie its parameter set to the motion. For this reason Timbre-trees are developed by deriving a heuristic from a rough notion of how the sound is actually produced by a real physical system. These heuristics are then used to help find an appropriate parameterisation for the sounds so that they can be mapped to the motion events. Not surprisingly, creating Timbre-trees for arbitrary sounds is a difficult process that requires repeated trial-and-error cycles from an experienced and creative sound designer. Even with evolutionary algorithms to explore

the vast space of possible sounds [Hahn et al. 1996], Timbre-trees are difficult to use especially as some sound events do not have an obvious direct physical correspondence. Furthermore, there is no support for re-use of pre-recorded soundtracks with Timbre-trees and describing complex natural sounds such as traffic sounds or a restaurant ambience remains an open problem.

2.1.1.5 Physically-based approaches

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 [Takala and Hahn, 1992; van den Doel and Pai, 1996; O'Brien et al., 2001; van den Doel, 2001; O'Brien et al., 2002]. These methods are *physically-based* since the synthesis algorithms are designed by modelling the physical mechanisms that underlie sound production. If an object is struck, the energy of impact causes deformations to propagate through the body, causing its outer surfaces to vibrate and emit sound waves. By analyzing the surface motions of animated objects, the acoustic pressure waves in the surrounding medium are derived. For example, the sound produced by a struck object can vary with the impact velocity and location (Figure 8). By analogy, the physically-based modelling approach has also been adopted in computer graphics, for modelling radiosity and light propagation. Physically-based models for audio and graphics can be easily synchronized, at least in principle. This allows a high degree of perceptual coherence of acoustic and visual events.

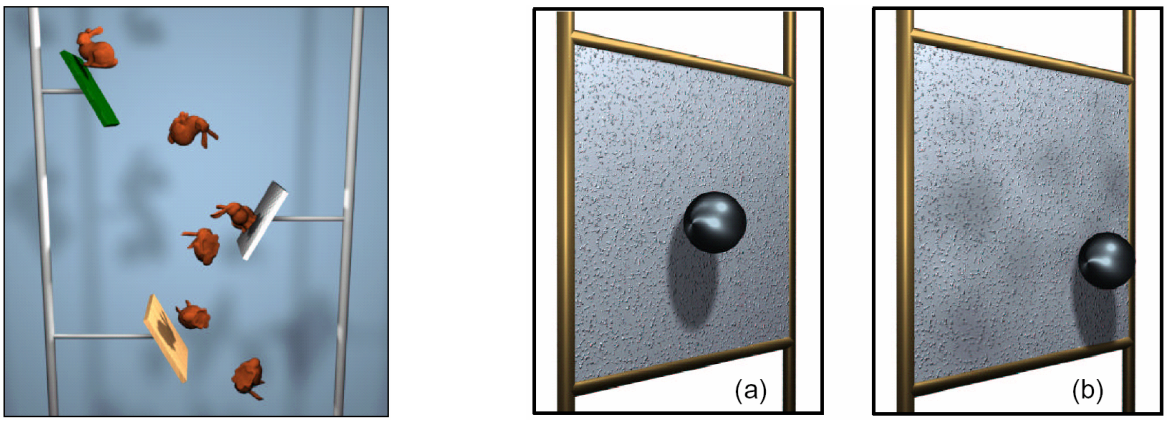

Figure 8: **Physically-based sound synthesis**. (***Left***) The motion of several bunnies falling through a chute produces synchronised impact sounds for both the bunnies and the shelves [O'Brien et al. 2002]. (***Right***) A square plate being struck (a) on the centre and (b) off centre generates corresponding realistic sounds [O'Brien et al. 2001].

The problem with physically-based approaches is that sounds are synthesized from scratch, so they do not exploit the multitude of existing pre-recorded sound libraries. Also, they require physical models of sound or motion to identify the underlying generative process for a given sound. This is feasible for straightforward collision sounds but becomes non-trivial for more complex sounds where the relationship between the animation and the soundtrack is less clearly defined.

2.1.1.6 Temporal Matching

Another way of re-using soundtracks in animation and video was examined by Tadamura and Nakamae [1998]. Instead of using sound synthesis, they use temporal matching between different media to synchronize each medium's start and end time with that of the others. The length of audio and MIDI soundtracks are changed interactively by the user throughout the sequence. A screenshot of their prototype interface is shown in Figure 9. In the case of audio (such as narrations), they remove silent and nearly silent (or quasi-silent) portions as necessary to fit the user constraints. For MIDI background music, adaptive tempo alterations are used. The advantage is that soundtracks can be re-used on different animations and videos without noticeable deterioration. The disadvantage is that the extent to which a soundtrack can be re-synchronised is limited. This method is well suited to relatively small timing alterations and does not support re-ordering; although standard commercial audio packages can be used for this.

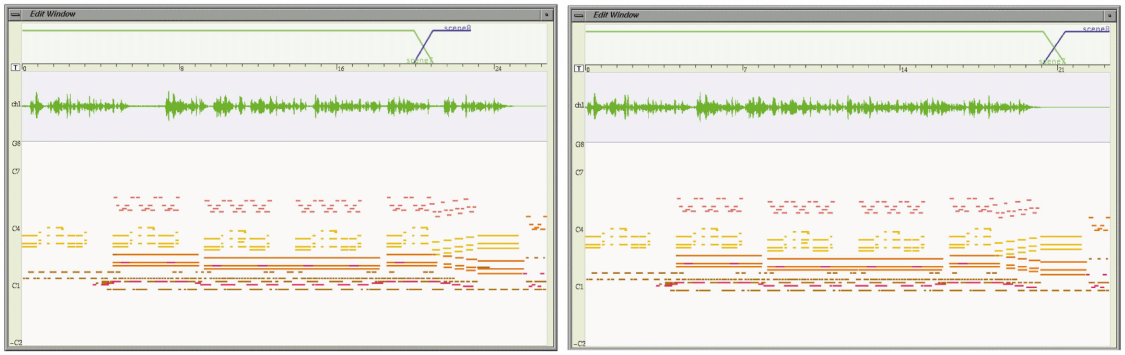

Figure 9: **Interface for Tadamura and Nakamae's [1998] system**. (*Left*) Before the adjustment process and (*Right*) after the adjustment process. The end of the MIDI (*bottom*) and the end of audio narration (*top*) are adjusted to fit the user constraint.

2.1.1.7 Texture Synthesis-by-example

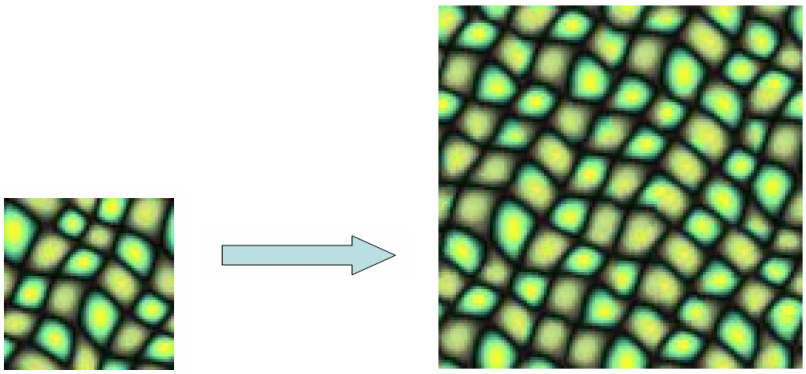

Figure 10: **Texture Synthesis**. Only a small input image (*left)* is necessary to synthesize a larger version (*right)*.

The goals of 2D texture synthesis-by-example are similar to what we are trying to accomplish over sound as outlined in our initial goals. The emphasis is on re-use of existing textures to form new textures to fit another context. Therefore, the sound synthesis methods presented in this report are conceptually and algorithmically similar to those used in the active field of 2D texture synthesis-by-example. A texture synthesis method starts from an example image and attempts to produce an arbitrarily sized texture with a visual appearance similar to the example. Recent texture synthesis algorithms share a common theme of local neighbourhood-based statistical approaches. They are based on the assumption that textures can be formalized as a stationary stochastic source modelled by conditional probabilistic distributions. The source is considered stationary so that the statistics it exhibits in a region are invariant to the region's location. A texture sample is a sample from this stochastic source. Simple non-parametric sampling methods, such as in Efros and Leung [1999] and Ashikhmin [2001], can be used to estimate the distribution describing this stochastic source. The conditional distribution of a pixel, given all its spatial neighbours synthesized so far, is estimated by querying the sample image and finding all similar neighbourhoods. Generating new random samples is therefore referred to as *sampling* the model. In the classic Efros and Leung [1999] method, this is done by generating the output image pixel-by-pixel in scan-line order, choosing at each step a pixel from the example image whose neighbourhood is most similar with respect to the Euclidean norm to the currently available neighbourhood in the texture being synthesized. The algorithm must therefore compare the current neighbourhood to that of all pixels in the example image. Figure 11 depicts a step in the process to synthesize the output image in Figure 10.

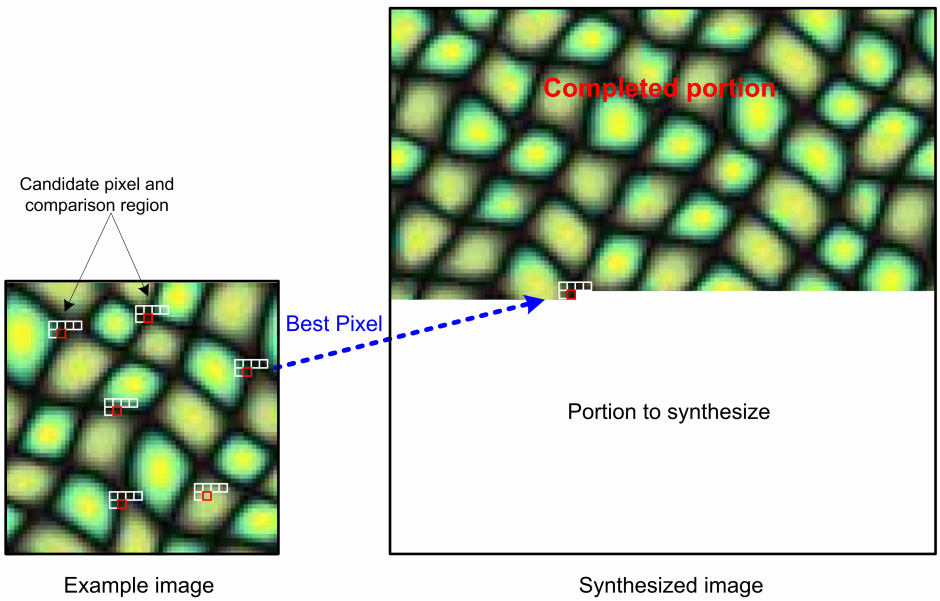

Figure 11: **The Efros and Leung [1999] texture synthesis process**. New pixel values (in red) in the synthesized image are generated in scan-line order by selecting the candidate pixel with the most similar L-shaped neighbourhood to that of the pixel being synthesized.

Following the Efros and Leung [1999] method, control over the image texture synthesis algorithm was introduced in [Ashikhmin 2001, Hertzmann et al. 2001]. By allowing the user to manually specify an input and output mask, the synthesis algorithm knows where the texture has to come from and where the texture needs to be synthesized to. This is a form of directed texture synthesis, called *Texture-by-Numbers*, which allows the user to re-arrange an image to conform to a paint-by-numbers sketch. The basic idea behind these approaches is that the synthesis algorithm now requires that each pixel satisfy the desired output map as well as satisfying the texture synthesis requirements. This is done by including the neighbourhoods inside the user maps in the final calculation of the pixel neighbourhood similarity measure described above. In Figure 12, the blue area delineates the unwooded areas in the input image. The synthesis algorithm then uses the output map to determine where these unwooded areas must be generated in the output image. Having the same type of control over sounds would greatly simplify the production of a soundtrack.

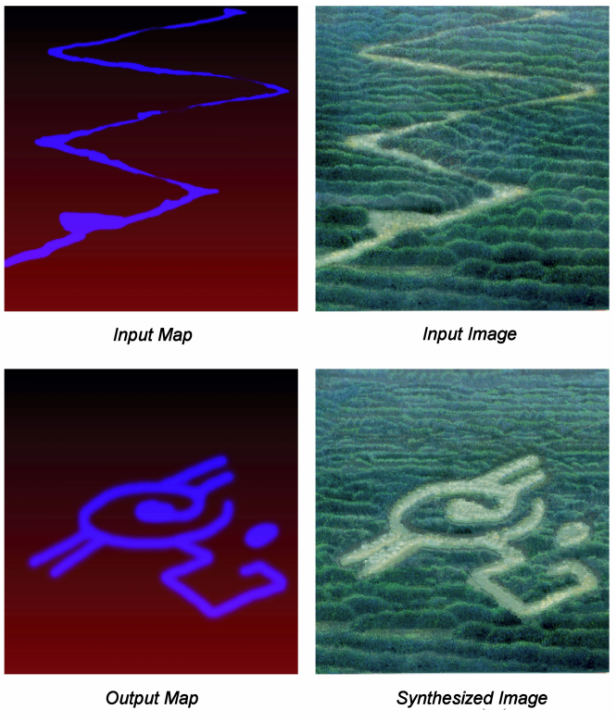

Figure 12. **Texture-by-Numbers** [Hertzmann et al. 2001]. The input image is spatially re-ordered following the user maps. Note that the choice of the colours in the user map is arbitrary as long as they are consistent.

Another relevant texture synthesis technique is *texture transfer* where an image is *texturized* with some arbitrary texture [Ashikhmin 2001, Efros and Freeman 2001, Hertzmann et al. 2001]. A controlling image and an input texture are provided by the user. The controlling image is then reconstructed using elements of the input texture. In Texture-by-Numbers, the controlling image corresponds to the output map which determines pixel region memberships. Here, the controlling image determines a spatial map where a certain quantity must be satisfied such as image intensity, blurred image intensity or local image orientation angles. Texture transfer synthesizes images drawn from the statistical distribution of neighbourhoods in the input texture while trying to match the quantities in the controlling image as closely as possible. Figure 13 shows an example where luminance of the face image is used to constrain the synthesis of the rice texture. The face appears to be rendered in rice. This is because bright areas of the face and bright patches of rice are defined to have low correspondence error and therefore are matched up during synthesis. The ability to apply similar methods to sounds would enable a soundtrack to be rebuilt from any given *sound texture* [Saint-Arnaud and Popat 1997], where the controlling soundtrack would determine the temporal map where a certain quantity such as volume or pitch must be satisfied. A large class of natural and artificial sounds such as rain, waterfall, traffic noises, people chatting, machine noises, etc., can be regarded as sound textures.

Figure 13: **Texture Transfer**. The face image (*bottom-left*) is rebuilt using elements of the rice texture (*top-left*) resulting in a face made of rice (*right*).

2.1.1.8 Sound Synthesis-by-example

Having observed that the goals of constrained texture synthesis are the same as ours on sound, we look at existing work that applies conceptually similar algorithms to texture synthesis on sound. This is referred to as sound synthesis-by-example and can be thought of as a temporal extension of 2D image texture synthesis (Figure 14). The difference here is that instead of focussing on spatial coherence as in texture synthesis, focus is on temporal coherence in sound synthesis. The ordering of events in sound is fundamental since we perceive a sound as a sequence from the first sound to the last. On the other hand, an image is experienced as a whole since we (rarely) look at it in a scan-line order. The *x* and *y* dimensions of an image can be treated in the same manner with image synthesis whereas in sound, the temporal and spatial dimensions should be analysed differently.

The first sound synthesis-by-example method was introduced by Bar-Joseph et al. [1999; Dubnov et al. 2002]. They use the concept of *granular synthesis* where complex sounds are created by combining thousands of brief acoustical events [Roads 1988]. Analysis and synthesis of sounds is carried out using a wavelet-based time-frequency representation. Their method generates a new sound by synthesizing a new multi-resolution wavelet-tree level-by-level from the root down. New wavelet coefficients of the synthesized tree are ranked by comparing their neighbourhood to that of the coefficient's ancestors (up one tree level) and temporal predecessors (previous coefficient on same level) in the original sound-texture's tree. The winning coefficient is then uniformly picked from the set of candidate coefficients that are within a user threshold. This can be seen as a multi-resolution extension of the image synthesis method proposed by Efros and Leung [1999] where the synthesis not only takes place in scan-line fashion but also on a level-by-level basis. Another difference is that local neighbourhoods are augmented to include the corresponding neighbourhoods of ancestors from previously synthesized upper levels. More details are given in Section 3.3.

While the Bar-Joseph et al. (BJ) algorithm works on both stochastic and periodic sound textures, it does not provide control over the new instances of the sound texture it generates. It is perfectly suitable for cases when the BJ algorithm is simply used to generate longer variations of a short sample sound. No control is possible over how the sample sound is re-arranged to form the new sound. It simply has to generate a new instance of a sound that appears to be from the same source as a given sample sound. A useful extension to this work would be to control the nature of the temporal re-arrangement in the new sound while it still appears to come from the same source. This would be essential for synchronisation purposes where we want certain sounds in the sample to appear at specific locations in time.

Figure 14: **Sound synthesis-by-example**. A new variation (*right*) on the original sound source (*left*) is generated while still bearing a strong resemblance to the original.

Other approaches to sound texture generation are presented by Hoskinson and Pai [2001] and Lu et al. [2002]. They operate in a similar fashion to Bar-Joseph et al. [1999], by first analyzing and segmenting the input into variable-sized chunks, or grains, that are recombined into a continuous stream, where each chunk is statistically dependent on its predecessor (details in Section 3.4). Again, no control is possible over new instances of a sound texture except that they appear to be from the same source as a given sample sound. Adding user control to the synthesis process would extend the basic concept of a sound texture to further increase its applicability.

2.1.1.9 Musical Synthesis

A great deal of further work on sound modelling outside the field of computer graphics has been undertaken. Digital sound and music have considered modelling the sounds generated by musical instruments accurately, or even creating totally new sounds [Roads 1996; Cook 2002]. Classic techniques like additive synthesis, subtractive synthesis, frequency modulation, formant synthesis, linear predictive coding or physical modelling methods are aimed at either musical or speech synthesis, their suitability for the synthesis of ambient sounds has not been thoroughly investigated. Since the nature of musical tones is fundamentally different from those of real-world sounds, and particularly of ambient and natural sounds, those synthesis methods which are mainly targeted towards the generation of harmonic spectra are not

typically applicable. More fundamentally, the goal of this research is not to create completely new sounds or to modify the spectral content of existing sounds, but to re-order them in time to fit high-level user constraints whilst preserving the original spectral content.

2.1.2 Conclusion

Like the present work, many of the above systems have investigated the partial automation of sound production. However, as stated in our initial goals, we aim to facilitate the editing of soundtracks by using the extensive libraries of pre-recorded samples and animation data. Non-linear editing tools effectively draw on such sound libraries but require considerable manual intervention. Triggered sounds, such as those used in video-games, can be repetitive and ultimately suffer from the latter problem, but have the advantage that they can be driven by animation. Paramaterizable sound constructs, such as Timbre-Tree and physically-based approaches, can produce perfectly synchronized and realistic soundtracks for animation with little user intervention. However, they support a limited sound class and do not allow for re-use of existing sounds. Since our aims for sound production are more similar to recent innovations in texture synthesis, we discussed a subset of these methods that possess algorithmic and conceptual similarities to the present work. Texture synthesis inspired recent work on sound synthesis-by-example which is the basis of our own approach.

Chapter 3

SOUND EDITING

Sound is traditionally divided into three elements: dialogue, music and effects. In this chapter, we primarily focus on sound effects; that is any auditory information that is not speech or music. Most current video, video-games, virtual reality (VR) and animation productions use pre-digitized sound effects rather than synthesized sounds. Pre-digitized sounds are static and are difficult to change in response to user preferences or actions. Furthermore, obtaining a pre-digitized, application specific sound sequence is difficult and often requires sophisticated sound editing hardware and software [Miner, 1994]. Creating an acoustically rich soundtrack requires thousands of sounds and variations of those sounds. Obtaining these very large digitized sound libraries is expensive and impractical. The alternative to using pre-digitized sound is to use sound synthesis. The approach described here is a step towards providing a flexible sound synthesis tool for video, animation, video-games and VR. The goal is to 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. The algorithm automatically generates soundtracks for input animations based on other animations' soundtracks. Additionally, audio-driven synthesis (also referred to as *sound transfer*) is supported by matching certain audio properties of the generated sound texture to that of another soundtrack.

Three different controllable sound synthesis models were considered:

- A wavelet-based approach
- A natural grain-based approach
- A self-similarity-based approach

In all cases, the source audio is analyzed and segmented into smaller chunks, such as grains or wavelet coefficients, which are then recombined to generate statistically similar variations of the original audio.

Control is obtained by specifying where preferred grains from the source audio should be favoured during the synthesis, or by defining the preferred audio properties (e.g. pitch and volume) at each instant in the new soundtrack.

3.1 Aims

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 aim to provide more natural means of specifying soundtracks specifically for computer animation with applicability to video, video-games and VR where the reuse of existing soundtracks is maximized. Rather than creating a soundtrack from scratch, broad user specifications such as "*more of this sound and less of that sound"* or "*rebuild this soundtrack with these new sounds*" should be possible. Alternatively, in the case of computer animation, a user should simply supply a sample animation (along with its soundtrack) and, given a new animation, say, in effect: "*Make it sound like that*". Another option would be to allow a user to simply supply a driving soundtrack, a new target sound, and say in effect: "*Make it sound like this whilst syncing with that*". For example, a laughing audience's recording could be automatically replaced by a synchronized *booing* soundtrack. All these methods would significantly simplify existing soundtrack recycling since no editing, looping, re-mixing or sound source separation would be necessary.

The target soundtrack type is not dialogue or music since maintaining the long-term order of occurrence is essential in these cases. This includes both synchronous sounds (matched to an on-screen source) and asynchronous sounds (no on-screen source visible), but the primary focus is on sound effects, Foley effects and ambient sounds. Ambience is the background recording of a particular place that identifies it aurally. While mostly in the background, these sounds are vital for the creation of mood and realism. Examples include cave, factory and swamp ambiences.

3.2 Overview

We present three approaches for simple and quick soundtrack creation that generates new, controlled variations on the original sound source, which still bear a strong resemblance to the original, using controlled stochastic algorithms. Additionally, motion information available in computer animation, such as motion curves, is used to constrain the sound synthesis process. Our system supports many types of soundtracks, ranging from discrete sound effects to certain music types and sound ambiences used to emphasize moods or emotions.

In the simplest case, the user manually indicates 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 at particular locations 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. This is referred to as *Sound-by-Numbers* since it operates in a similar fashion to Paint-by-Numbers kits for children. But instead of solid colours, or textures in Texture-by-Numbers [Hertzman et al. 2001], sound is automatically synthesized into the corresponding mapping.

We extend this method to automatically synthesize 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.

Finally, in two of our models, we make it possible to use audio to constrain the synthesis process. The goal here is to produce a new sound texture that exhibits some similar property (such as volume or pitch) to that of a separate guiding soundtrack. The user only has to specify the audio matching feature to use. An advantage of this method is that it provides a novel, yet very natural means of specifying soundtracks.

3.3 Wavelet-based Sound Synthesis

This sections deals with our first controllable synthesis model. It utilizes wavelet-tree learning as the basis for its analysis and synthesis of sound. Three interaction methods for defining a new sound are also presented, each with increasing automation. We present an algorithm which extends the granular audio synthesis method developed by Bar-Joseph et al. [1999] (abbreviated as BJ below) by adding control to the synthesized sounds, a description of which is given in Section 3.3.1. In section 3.3.2 and 3.3.3, we present our user-guided approach and detail three types of intuitive user-control.

3.3.1 Sound Synthesis by Wavelet Tree Sampling

The sound synthesis method proposed by Bar-Joseph et al. [1999] is a variant of granular synthesis which operates on a time-frequency representation of the audio to generate new sound textures. The outputs are synthesized sounds of arbitrary length that appear to have been produced from the same underlying stochastic process as the finite input sound. In order to construct a new sound, a wavelet hierarchy is constructed from the input sound. The goal here is to produce a new sound hierarchy from the original which has structural changes at the macro level yet maintains the intermediate and micro-level structures which characterize the input sound.

The synthesis process is performed in two stages. First, the input sound is separated into a wavelet coefficient tree. This produces interdependent multi-resolution grains ranging from large scale properties of the audio to near sample level details. A new sound texture is then generated by recombining these grains using non-parametric sampling. The generated wavelet coefficient tree is made statistically similar to that of the input's sounds. The inverse wavelet transform of the new tree produces the new soundtrack. The following section explains the various steps necessary to synthesize a new sound texture, as depicted in Figure 15. The wavelet analysis of audio signals is introduced including how the wavelet coefficient tree is obtained. Section 3.3.1.3 details the phases of the BJ algorithm and its shortcomings when more controlled output is desired.

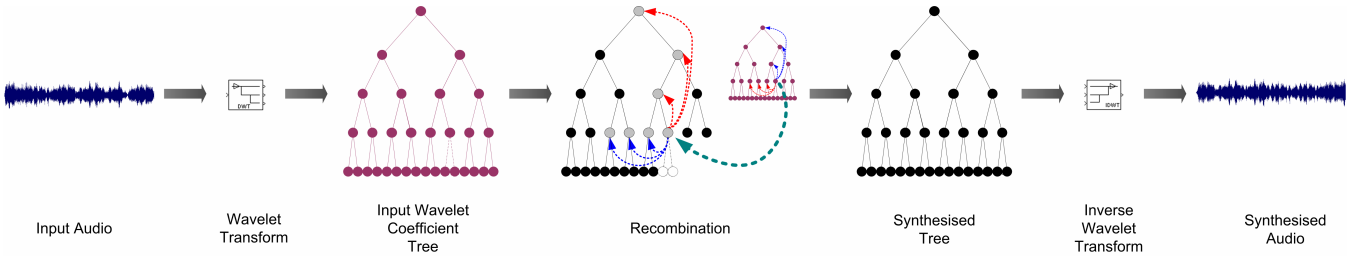

Figure 15: **Overview of unconstrained sound texture generation steps.** The wavelet-tree representation of the input audio is first obtained. By recombining the wavelet coefficients, a statistically similar variant of the original audio's wavelet tree is generated. The inverse wavelet-transform is then performed on it to produce a new sound texture.

3.3.1.1 Wavelet Transform

The Wavelet Transform, or multi-resolution analysis (MRA), is a relatively recent and computationally efficient technique for extracting information about signals such as audio. Wavelets have the ability to analyse different parts of a signal at different scales by breaking up a signal into shifted and scaled versions of the original (or *mother*) wavelet. A wavelet is a waveform that has an average value of zero over its finite interval domain. The continuous wavelet transform (CWT) is defined as the sum over all time of the signal multiplied by scaled, shifted versions of the wavelet function Ψ:

$$C(a,b) = \int_R s(t) \frac{1}{\sqrt{a}} \psi\left(\frac{t-b}{a}\right) dt$$

where:

a is a scale
b is the position
s is the input signal
Ψ is the wavelet function
C are the wavelet coefficients

From an intuitive point of view, the wavelet decomposition consists of calculating a *resemblance index* between the input signal and the wavelet located at position b and of scale a. If the index is large, the resemblance is strong, otherwise it is slight. The indexes $C(a,b)$ are called coefficients. The scale, which simply means stretching (or compressing) the wavelet, is approximately related to frequency by the following relationship:

$$F_a = \frac{F_c}{a.P}$$

where:

a is a scale
P is the sampling period
F_c is the centre frequency of a wavelet in Hz
F_a is the approximated frequency corresponding to the scale *a*, in Hz

The CWT results in a set of wavelet coefficients *C*, which are a function of scale and position. By multiplying each coefficient by the appropriately scaled and shifted wavelet, we get the constituent wavelets of the original signal.

Signals with fast oscillations or even discontinuities in localized regions may be well-approximated by a linear combination of relatively few wavelets. Wavelets were developed as an alternative to the short time Fourier Transform (STFT) to overcome problems related to its frequency and time resolution due to its fixed analysis window size. STFT provides uniform time resolution for all frequencies. On the other hand, the Wavelet Transform (WT) provides high time resolution and low frequency resolution for high frequencies, and high frequency resolution and low time resolution for low frequencies (see Figure 16).

This approach makes sense especially when the signal at hand, such as audio, has high frequency components for short duration and low frequency components for long duration. In that respect, it is closer to the human ear which exhibits similar time-frequency resolution characteristics [Wang and Shamma, 1994].

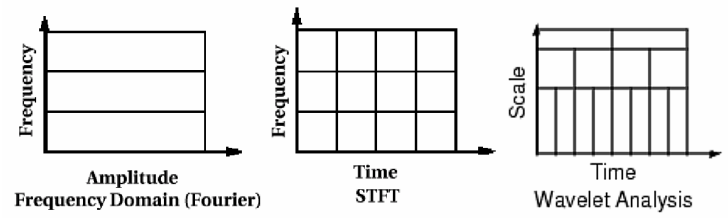

Figure 16: Wavelet analysis allows the use of long time intervals where we want more precise low-frequency information, and shorter regions where we want high-frequency information.

3.3.1.2 Discrete Wavelet Transform

CWT is computationally expensive since wavelet coefficients have to be calculated at every possible scale and position. The Discrete Wavelet Transform (DWT) analysis requires only the values of the transform at power of two (or *dyadic*) scales and positions to obtain the same accuracy (Figure 17).

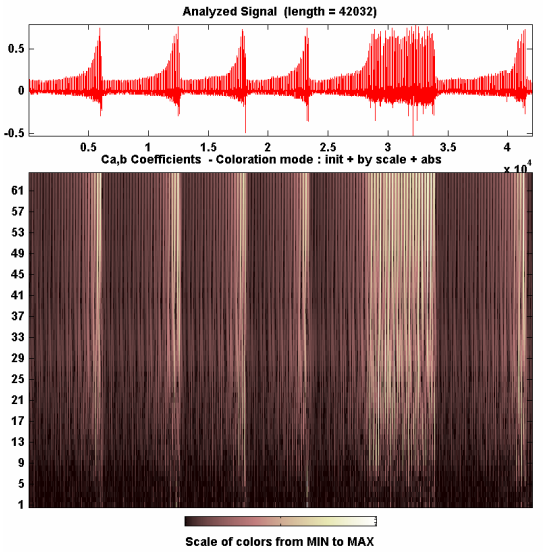

Figure 17: **Discrete Wavelet Transform**. Wavelet coefficients calculated for an audio sample.

The input signal is analyzed in different frequency bands with different resolution by decomposing the signal into a coarse approximation and detailed information. The coarse approximation is then further decomposed using the same wavelet decomposition step. This is achieved by successive high-pass and low-pass filtering of the time domain signal and is defined by the following equations:

$$y_{high}[k] = \sum_n \left(x[n] \cdot l[2k-n]\right)$$

$$y_{low}[k] = \sum_n \left(x[n] \cdot h[2k-n]\right)$$

where:

x is the input signal
h is a high-pass filter
l is a low-pass filter
y_{high} is the high-frequency detail of x
y_{low} is the low-frequency approximation of x

Note that at each step y_{low} and y_{high} are down-sampled by 2 so that we get as many data points before and after the filtering operation. The nature of the complementary decomposition filters h and l is determined by the shape of the utilized wavelet. This decomposition process consists of $\log_2 N$ stages at most, given a signal x of length N. The process is depicted in Figure 18. This results in the wavelet-decomposition or Mallat tree (Figure 19). For our purposes, the coefficients corresponding to every decomposition level (in blue in Figure 18) in the Mallat tree are re-organized into a fully-balanced binary tree (Figure 19). If there is a complete decomposition of depth m, the lower layer of the binary tree has 2^m nodes, each $N/2^m$ samples

long, where *N* is the number of samples in the original audio. This binary tree forms the input to the sound synthesis algorithm.

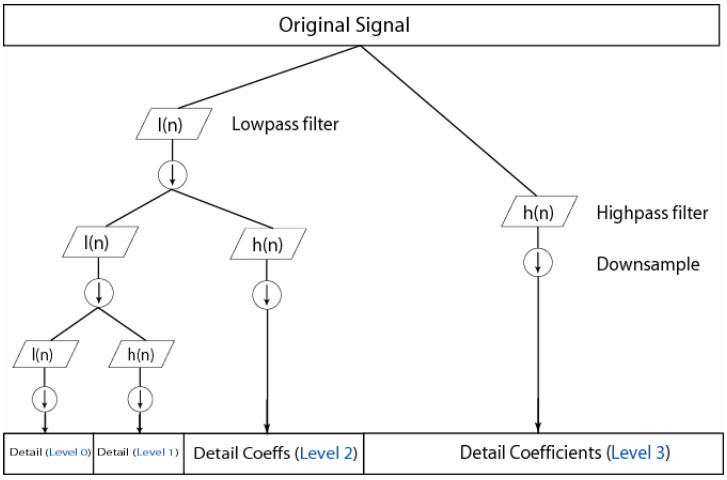

Figure 18: **The Discrete Wavelet Transform decomposition process**. The levels in blue correspond to the wavelet-coefficient tree levels used in the sound synthesis. The coarse representation of the original signal on level 1 requires fewer coefficients than the detail coefficients of level 2 as depicted by the corresponding size of the lower rectangles.

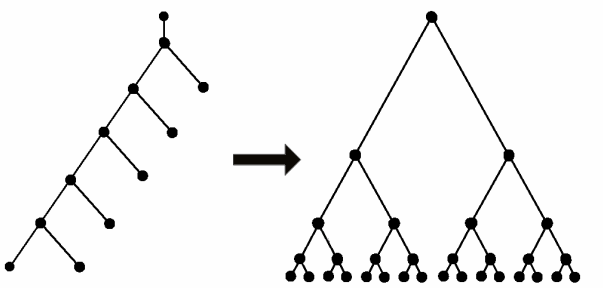

Figure 19: The Wavelet Transform tree (*left*) is reorganized into a fully-balanced binary Wavelet Transform Mallat tree (*right*). The top level of the binary tree (*left*) corresponds to the average value of the whole tree. The precision of the wavelet coefficients increases further down the tree.

3.3.1.3 The Algorithm

To generate new sound textures, the BJ algorithm treats the input sound as a sample of a stochastic process. The algorithm first builds a binary tree representing a hierarchical wavelet transform of the input sound, and then learns and samples 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 using a longest matching suffix distance metric. 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 20). 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 fidelity to the input sound and a large value allows more randomness.

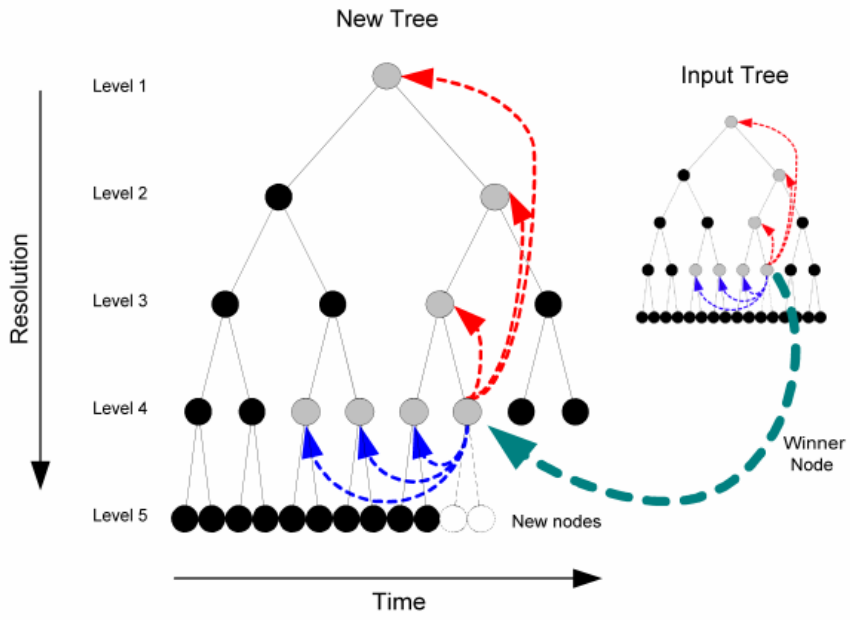

Figure 20: **BJ Synthesis step during the construction of a portion of a multi-resolution wavelet-tree**: Level 5 nodes in the new tree are synthesized by stepping through each parent node at level 4. For each node in level 4, we find a winning candidate, in the input tree, that depends on its scale ancestors (upper levels, pointed at in Red) and temporal predecessors in the same level (those to its left on level 4, pointed at in Blue). The children of the winning candidate are then copied onto the corresponding positions at level 5.

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 4). The winning node is randomly chosen by uniformly sampling from candidate set C_{pred}. More specifically, the following steps in

Figure 21 are performed during the synthesis phase.

- Build wavelet-tree t of input sound
- Randomly select root of new tree t'
- For every node of every level in t', starting from the top:
  - C_{anc} = All nodes in t at same level found with maximum number of ancestors within difference-threshold δ
  - C_{pred} = All nodes in C_{anc} found with maximum number of predecessors within difference-threshold δ
  - Winner is randomly picked from C_{pred} by *uniform* sampling
  - Copy winner node's children wavelet coefficients into t'
- Apply inverse wavelet transform of t' to synthesize the new sound.

Figure 21: The tree synthesis algorithm overview.

3.3.2 Directed Sound Synthesis

The BJ algorithm works on both stochastic and periodic sound textures. However, no control is possible over new instances of a sound texture. 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. An overview of this constrained synthesis is given in Figure 22. Note that only once the user has selected the source segments and defined the target sound length, can the instances of the target sounds be defined. The synthesis uses these selections to constrain the wavelet-coefficient recombination. 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.

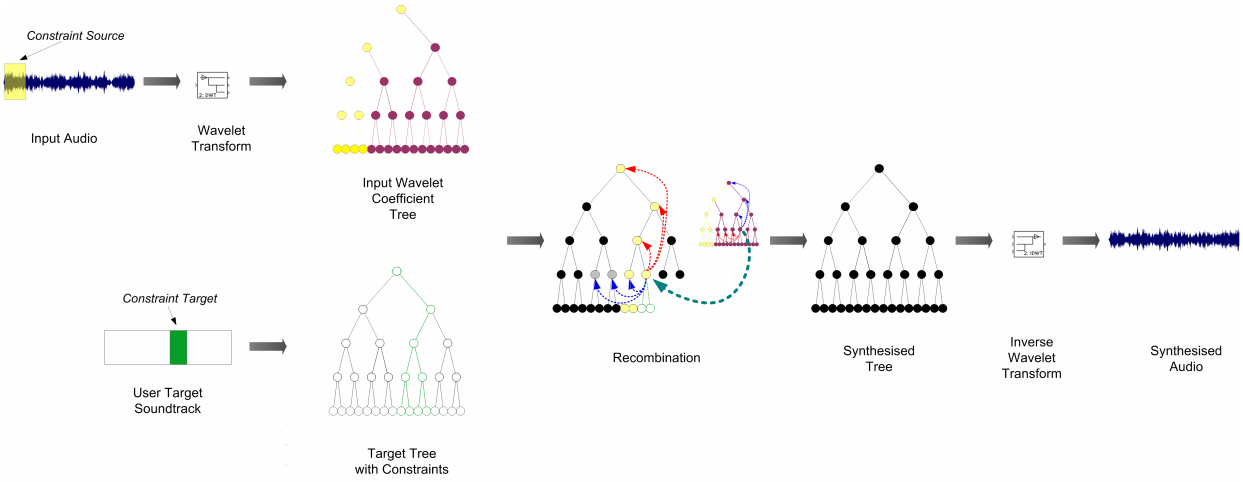

Figure 22: **Constrained sound synthesis overview.** First, the user selects the source sounds in the input audio (*in yellow, top left*), and when and for how long these sounds are to be synthesized in the target sound (*in green, bottom left*). Secondly, the wavelet-tree representation of the input audio is obtained, as well as an empty tree for the target sound. The nodes corresponding to the source and target regions are appropriately tagged. By recombining the wavelet coefficients whilst satisfying the user-constraints, a statistically similar variant of the original audio's wavelet tree is generated. The inverse wavelet-transform is then performed on it to produce the target sound texture.

3.3.2.1 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 23). Secondly, the animator specifies a target segment indicating when, and for how long, in the synthesized sound the explosions can be heard. The constraints for the rest of the new soundtrack can be left unspecified, so that in our *System Overview* video example (on the DVD-ROM), a battle-like sound ambience will surround the constrained explosion.

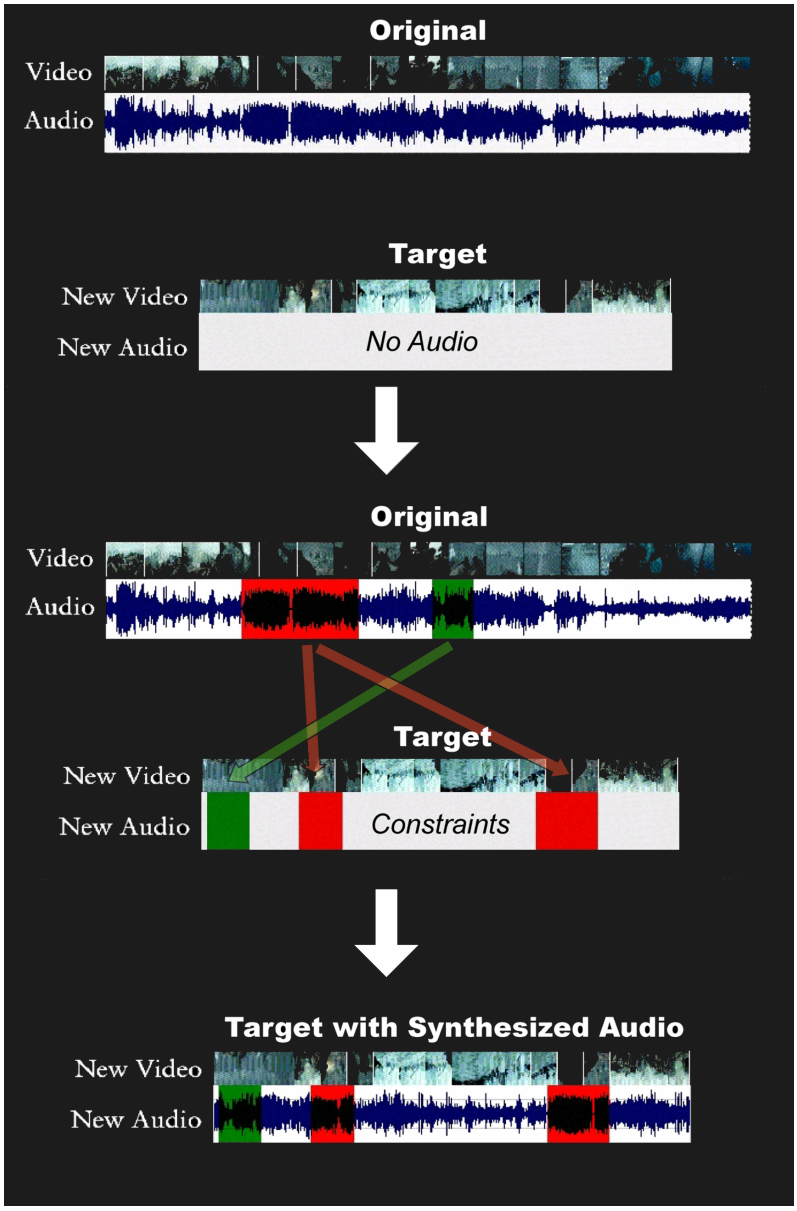

Figure 23: **Soundtrack Synthesis for a Video sequence**: The target video (*Top-lower*) is a rearranged soundless version of the source video (*Top-upper*). The explosion sounds in green, along with machine gun sounds in red (*Middle-upper*), are defined as synthesis constraints in the target soundtrack (*Middle-lower*). These constraints are used to guide directed sound synthesis into generating the appropriate soundtrack for the target video (*Bottom*).

The source and target segments, each defined by a start and end time, are directly specified by the user on a familiar graphical *amplitude*×*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 and overlap 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 segments occurring at every point in the target area. The weights vary in [-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 (Figure 24). In the special case where all weights are set to 0, unconstrained synthesis is identical to the original BJ algorithm.

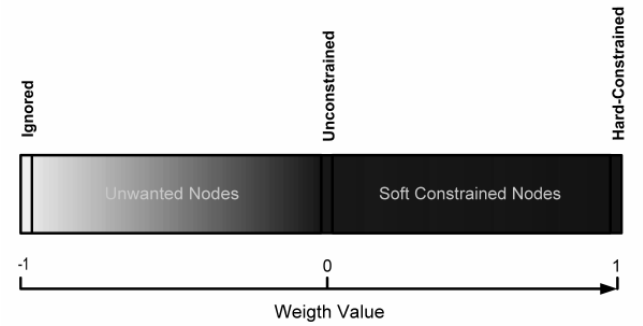

Figure 24**: Nodal weighting scale**. Going from left to right: nodes with weight -1 have no probability of occurring; improbable nodes range from (-1, 0); unconstrained nodes are set by default to 0; more probable nodes are in the range (0,1); finally node with weight equal to 1 impose hard-constrained synthesis.

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 the source and target segments for each constraint. Figure 25 depicts the node determination process for a given source segment. Each source and target segment combination defines a unique constraint $c \in \{1,2,...,n\}$ such as the explosions constraint and the gun-shots constraint. There are no limits as to the total number of constraints *n*. For each constraint c, we define two nodelists S_c and T_c, which contain the tree level and position offset of all nodes in, respectively, the source and target wavelet-tree involved in the constraint specification of c. T_c additionally contains the constraint weight associated with each node for constraint c. During the directed synthesis process, if the currently synthesized node is defined in T_c then this determines which nodes from S_c, and subsequently in the input wavelet-tree, should be used as potential candidates. We define $S = \{S_1, S_2, ..., S_n\}$ as the overall set of source nodes and $T = \{T_1, T_2, ..., T_n\}$ as the overall set of target nodes over all constraints *n*.

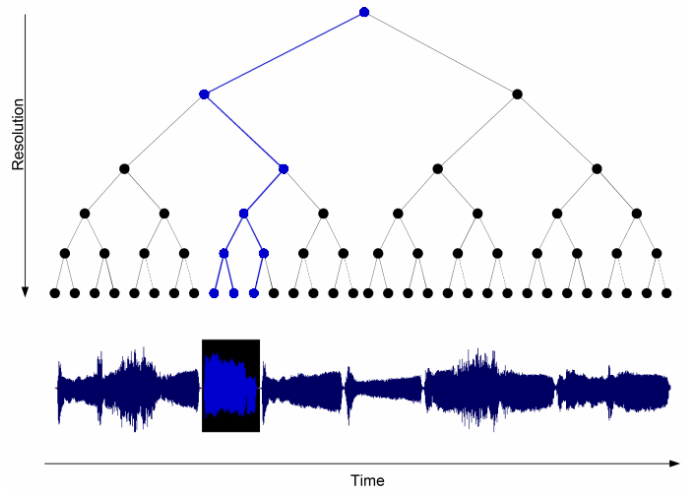

Figure 25: (*Bottom*) The user selects the black time segment on the time-amplitude representation of source sound. (*Top*) This enables the extraction of its corresponding wavelet coefficients (Blue nodes) in the balanced binary wavelet-tree required to generate that same segment.

3.3.2.2 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 neighbouring nodes appearing forward in time at the same tree level. During synthesis, potential candidate nodes are partitioned into the set of constrained w and unconstrained nodes $\overline{w}$ depending on the nature of their successor node.

In order to avoid perceptual discontinuities in the synthesized sound, the algorithm looks at distant successors instead of just the immediate successors. This allows for better anticipation of future sound constraints. Nodes that appear before or after a source sound in the input soundtrack will more likely be selected, respectively, before and after a target sound in the synthesized soundtrack. The more we go down the tree, the further away is the successor node in nodal terms. Let d be the successor look-ahead distance defined as $d=2^l.k$, 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 tree levels. 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.

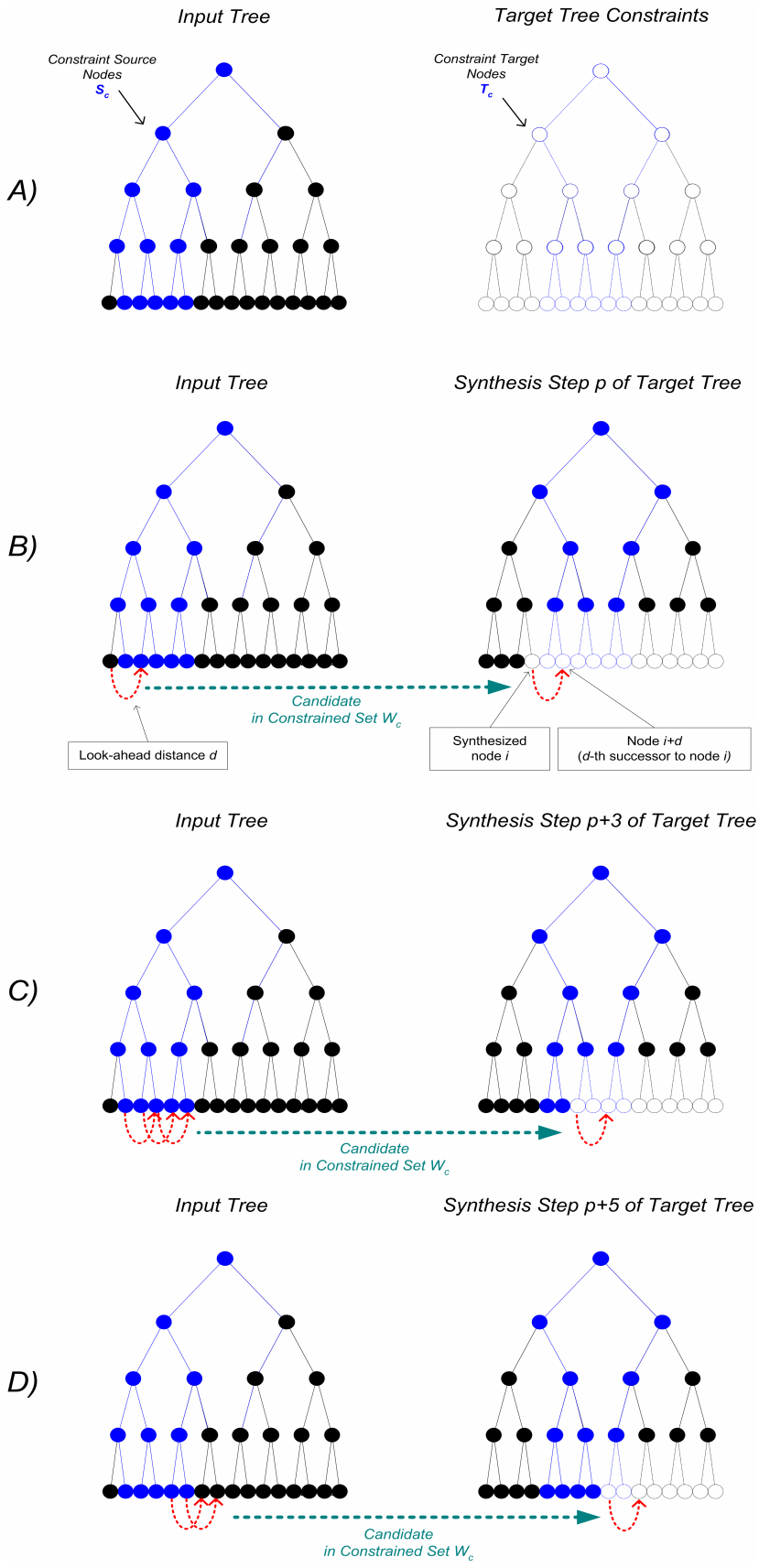

Figure 26: **Candidate selection for constrained set W_c.** Depending on the current synthesis position, the synthesis constraints change along with the set of candidate nodes W_c that satisfy these constraints. ***(A)*** The source tree on the left is generated from the input soundtrack. The blue nodes determine the set of source nodes S_c. The empty target tree on the right determines the nature of the new sound. The blue dashed nodes define T_c and indicate where nodes from S_c are desired. ***(B)*** At synthesis step *p*, node *i* is being synthesised in the target tree. Its anticipated successor node *i+d* is the node *d* positions to the right of node *i*, where *d* is the look-ahead distance. Here, node *i* is outside S_c and its successor node *i+d* is inside S_c. Therefore, the constrained candidate set W_c is formed of any nodes in the input tree outside S_c with a successor in S_c. ***(C)*** At synthesis position at step *p+3*, the constrained candidate set W_c is formed of any nodes in the input tree inside S_c with a successor in S_c. ***(D)*** At synthesis position at step *p+5*, the constrained candidate set W_c is formed of any nodes in the input tree inside S_c with a successor outside S_c.

If for constraint c the currently synthesized node belongs to T_c or has its *d*-th successor in T_c, or both, then w_c is the corresponding set of candidates inside and outside S_c satisfying the same constraint conditions (see Figure 26). Let all remaining nodes be contained in the set $\overline{w}_c$. Nodal matching is then separately carried out on both w_c and $\overline{w}_c$ in parallel, resulting in two candidate sets $C_{pred}^{w_c}$ and $C_{pred}^{\overline{w}_c}$, 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 for constraint c.

The winning node is then randomly chosen by non-uniformly sampling from $C_{pred} = C_{pred}^{w_c} \cup C_{pred}^{\overline{w}_c}$. Nodes in $C_{pred}^{\overline{w}_c}$ are given the default weight of 0 whereas the ones in $C_{pred}^{w_c}$ are all given the weight of T_c's current weighting[1]. Depending on T_c's weight value, this has the effect of biasing the selection process in favour or against the nodes in $C_{pred}^{w_c}$. If T_c's current weight is a hard-constrained 1 inclusion, then the winner is picked by uniform sampling from $C_{pred} = C_{pred}^{w_c}$ only. If T_c's current weight is a hard-constrained -1 exclusion, then candidates in $C_{pred}^{w_c}$ are ignored and the winner is picked by uniform sampling from $C_{pred} = C_{pred}^{\overline{w}_c}$ only.

In the case where multiple overlapping constraints are defined for the currently synthesized node, then there will be several sets of constrained $C_{pred}^{w_i}$ and unconstrained $C_{pred}^{\overline{w}_i}$ candidates; one for each constraint *i*. These sets are combined to form a single set of constrained and unconstrained candidates from which the algorithm can pick. We therefore get $C_{pred} = \left(\bigcup_{i=\{1,2,..,n\}} C_{pred}^{w_i} \right) \cup \left(\bigcap_{i=\{1,2,..,n\}} C_{pred}^{\overline{w}_i} \right)$ where n is the number of overlapping constraints. Nodes in $\bigcap_{i=1,2,...n} C_{pred}^{\overline{w}_i}$ are given the default weight of 0 and the ones in each $C_{pred}^{w_i}$ are given the weight in T_i for their constraint *i*. If a candidate z is included in multiple constrained sets from j separate constraints (i.e. that $z \in \bigcap_{i=\{1,2,..,j\}} C_{pred}^{w_i}$) then its weight is averaged out over all corresponding constraints weights in $T_{i=\{1,2,..,j\}}$.

[1] Note that the weights are actually converted from the range [-1,1] to [0,1] before non-uniform sampling. We used the range [-1,1] since it is more intuitive to the end-user.

While the above algorithm works well in most cases, sometimes the quality of the matches found in $C_{pred}^{w_c}$ might be inferior to those found in $C_{pred}^{\overline{w}_c}$ due to the reduced search space. We therefore want to prevent significantly inferior matches from $C_{pred}^{w_c}$ being chosen in order to maximize audio quality. This is controlled by ensuring that the sum of the maximum number of found ancestors in $C_{anc}^{w_c}$ and predecessors in $C_{pred}^{w_c}$ is within a user-percentage threshold *r* of that of $C_{anc}^{\overline{w}_c}$ and $C_{pred}^{\overline{w}_c}$. Let m_{w_c} and n_{w_c} be, respectively, the number of ancestors and predecessors for the best candidates currently in $C_{anc}^{w_c}$ and $C_{pred}^{w_c}$ within the randomness threshold *δ*. Let $m_{\overline{w}_c}$ and $n_{\overline{w}_c}$ be their equivalent in $C_{anc}^{\overline{w}_c}$ and $C_{pred}^{\overline{w}_c}$, then if $\left(m_{w_c} + n_{w_c}\right) < r\left(m_{\overline{w}_c} + n_{\overline{w}_c}\right)$, the candidates from w_c 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 in the source tree. 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 set might result in $C_{pred}^{w_c}$ being empty, since no node is within the imposed threshold limits. Consequently, in our algorithm, if no candidates are found in $C_{pred}^{w_c}$ whilst in hard-constrained inclusion mode, a full search is conducted in S_c. If matches within the threshold tolerance still cannot be found, the best approximate match in S_c is utilized instead.

3.3.3 User Control of Synthesis

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

3.3.3.1 Manual Control

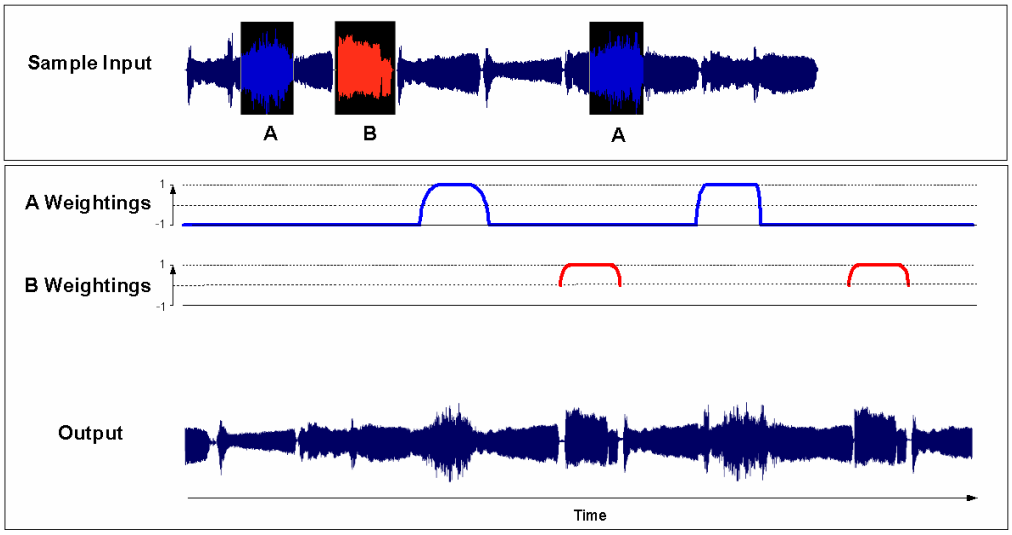

Figure 27. **Manual Constraint Specification**. (*Top*) Source regions A and B. (*Middle*) Weighting curves for A and B. (*Bottom*) Directed synthesis output.

The user starts by specifying one or more source regions in the sample sound. In the example depicted in Figure 27, two distinct source regions are defined corresponding to areas A and B (top). Note that A is defined by two segments which define the same source A. The user then draws the target probability curve for both sources A and B directly on the timeline of the new sound. A's weighting is -1 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 where unconstrained synthesis is utilized.

3.3.3.2 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. It is based on query-by-example [Keogh and Pazzani 1999] where the animator selects a reference sound or motion segment and asks the system to retrieve perceptually similar occurrences.

This results in a faster specification of the synthesis constraints since only one constraint is specified. This is especially true over extended animations and soundtracks. The selection of a single motion segment and its

target sound triggers the definition of multiple constraints. All motions similar to the query motion in the rest of the animation are assigned the same target sound. Sounds similar to the target sound are also detected and assigned to the same sound constraint.

Let us take the example of a car animation. The animator wants each turning motion of the car to correspond to tyre-screeching sounds from a given car sounds soundtrack. Firstly, only one car turning motion is selected by the user and all other turning motions are found. Secondly, the animator finds a single sound and lets the system find all other occurrences of that same sound. These sounds are then assigned as constraints to all turning motions. This gives more variety in the synthesized soundtrack since there is a larger set of slightly different tyre-screeching to pick from for each turning motion.

We now look at the different motion and audio matching strategies, as well as how audio synthesis weights for each motion match are determined.

3.3.3.2.1 Motion Matching

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. Users can roughly control the number of non-overlapping matches by setting the minimum threshold similarity θ. A value of zero of threshold θ corresponds to the lowest detected similarity measured, and 1 to the highest. Alternatively, the user can directly specify the maximum number of returned matches n. The system then returns up to n top matches ranked by their similarity to the query motion.

The techniques used to match over multi-dimensional motion curves, ranging from one dimensional positions or angle variations to high-dimensional motion capture sequences, are detailed in Chapter 4. The process is illustrated in Figure 28 and Figure 29 for the case of a one dimensional positional motion curve.

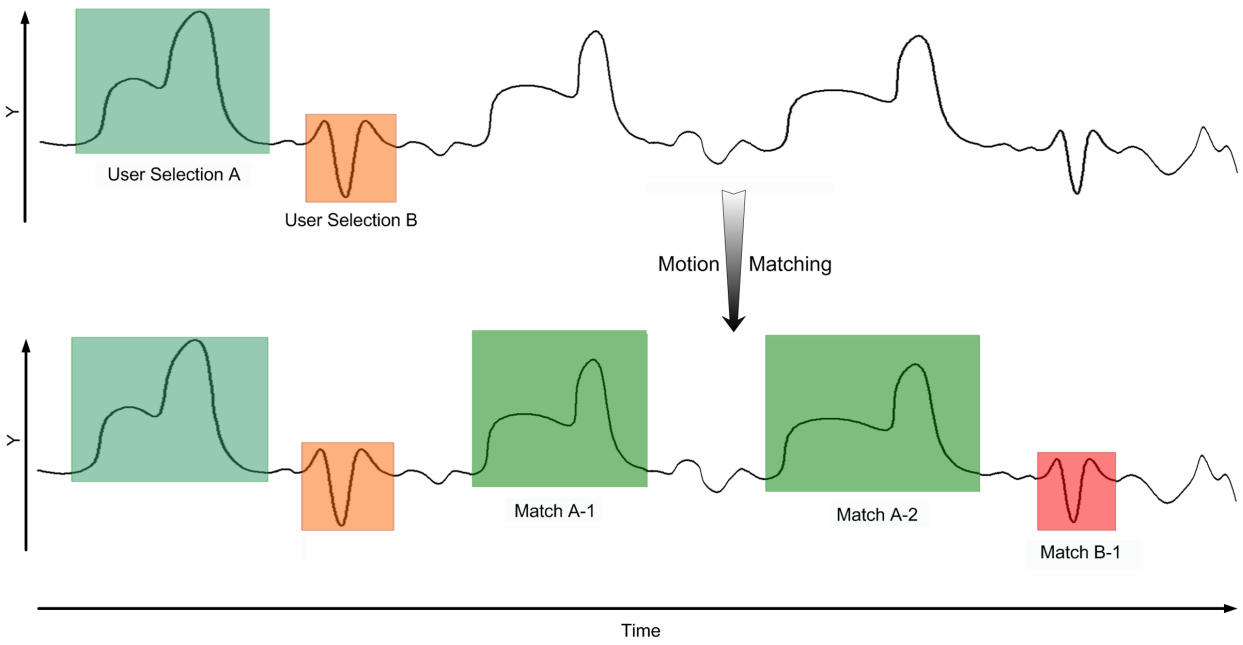

Figure 28. **Finding matching motions.** (*Top*) Motion segments A and B are first selected from the animation by the user. These define the query motions for which all similar occurrences must be detected. Motion matching is therefore performed to find all qualifying matches given the current matching thresholds. (*Bottom*) Two similar occurrences of query motion A are located at match A-1 and A-2; and only one occurrence, B-1, for query motion B.

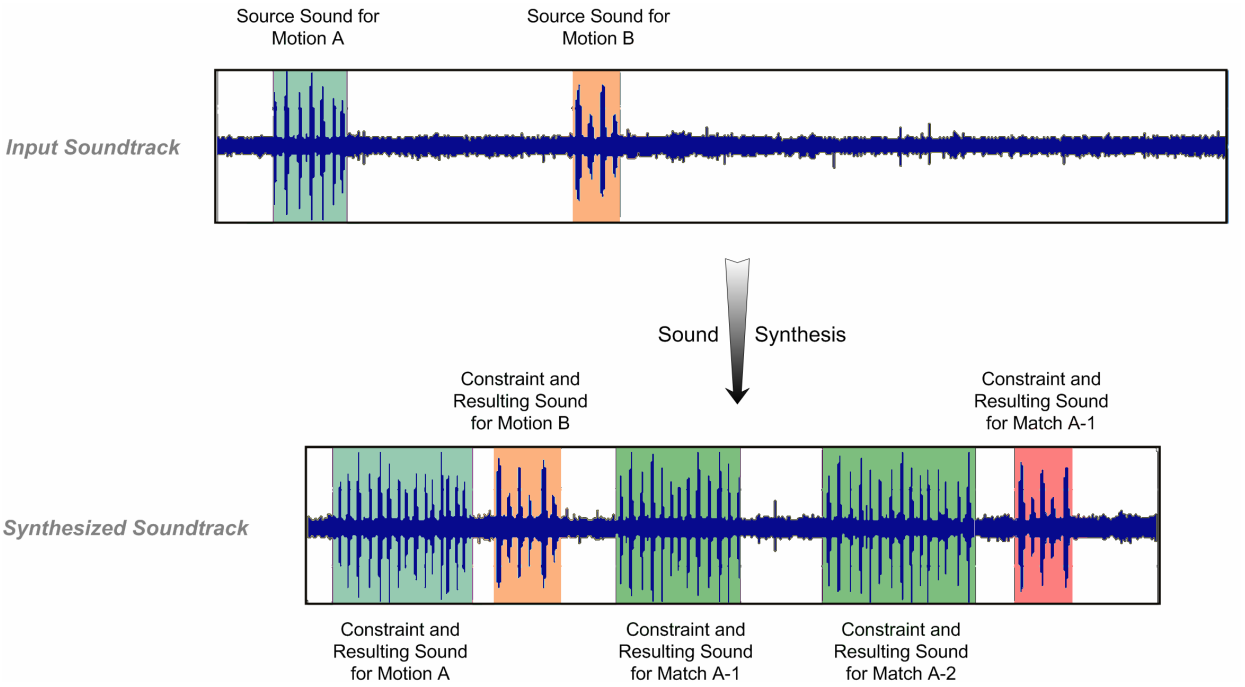

Figure 29. **Automated sound synthesis constraint specification.** Following on from Figure 28, the user-selected motions A and B are both assigned to separate source sounds from a separate soundtrack (*top*). These source sounds automatically become synthesis constraints for both the query motion and their respective matches (*bottom*). The resulting soundtrack, obtained by constrained synthesis, will accordingly trigger similar sounds for recurring query motions.

The matches only define the locations of the synthesis constraints on the target soundtrack. A weight must then be assigned to each constraint. 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 a high probability of having the same audio properties as the query, and conversely for weak matches. Given a match x, its audio strength w_x is therefore defined as:

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

where:

- x is the current match
- w_x is the audio weight of x, ranging from -1 to 1
- w_q is the query's initial audio weight (set by the user), ranging from -1 to 1
- S_x is the normalised similarity measure of x, ranging from 0 (dissimilar) to 1 (similar)
- θ is the minimum threshold similarity for acceptable matches, ranging in (0,1)
- c controls the global impact of the match strength S_x on w_x

w_x will approach the value of w_q when S_x is close to 1. In other words, the match's final weight will approach the query weight the more similar it is to the query motion. w_x is not directly proportional to the absolute value of S_x but to its relative value compared to θ. The reason for this becomes apparent when the similarity threshold θ needs to be set close to 1 to select the desired matches. The relative similarity difference between the matches determines each match's final weight. Otherwise, there would be very little difference between each match's final weight if the relative similarity difference between the matches was not taken into account. The term c is included to give the animator a global control over the effect of the matching strength on the resulting audio weights. If c is set to 0 then $w_x = w_q$ which is equivalent to our earlier, default weight assignment method. As c is increased, the similarity between the query and each match determines how close w_x is to w_q.

3.3.3.2.2 Sound Matching

Our system 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. This is especially valuable and

timesaving for selecting frequently recurring sounds over extended soundtracks. The animator only has to find a single occurrence of the source sound without having to sift through the whole soundtrack.

Matches are determined using the set of audio matching techniques introduced by Spevak and Polfreman [2001]. Amongst their alternative retrieval algorithms, we use the computationally efficient and most successful, plain trajectory matching algorithm. This offers robust and perceptually accurate matches since it is based on Mel Frequency Cepstral Coefficients (MFCC) [Hunt et al. 1996]. MFCC are a perceptually motivated compact representation of the spectrum used in speech processing and recognition. They are based on the discrete cosine transform of the log amplitude and smoothed discrete Fourier spectrum. An MFCC vector is extracted for every 20ms frame of the soundtrack. These vectors have thirteen cepstral coefficients that characterize the broad shape of the spectrum at each point in the soundtrack. Further details on how MFCC are extracted are given in Section 3.5.1.1.1. Matches are determined by finding non-overlapping audio segments with a series of similar MFCC vectors to that of the query audio. In other words, the trajectory in feature space of the query audio consists of a series of MFCC vectors. This trajectory is compared to every possible test trajectory of equal length over the whole soundtrack. Distance at every frame is measured using a one-to-one Euclidean distance measure of each frame's respective MFCC. Peak picking detects the set of potential matches and a minimum distance rule enforces a minimal inter-onset interval between adjacent or overlapping matches. Animators can roughly control the number of matches by setting the minimum threshold similarity percentage, or by directly setting the maximum number of returned top-*n* matches. The matching steps are depicted in Figure 30.

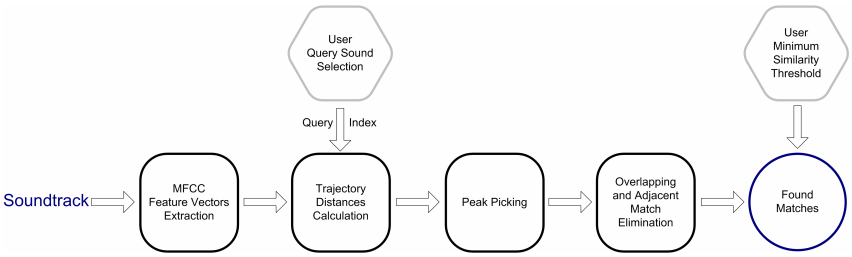

Figure 30: MFCC-based audio matching process.

Figure 31 illustrates the matching process in the case of trajectory matching on a given sample sound. First, the animator selects the query segment from the soundtrack. The distance to the query is then measured over the trajectory in feature space. In this example, only the top-four best matches' minima from the resulting distance curve were kept.

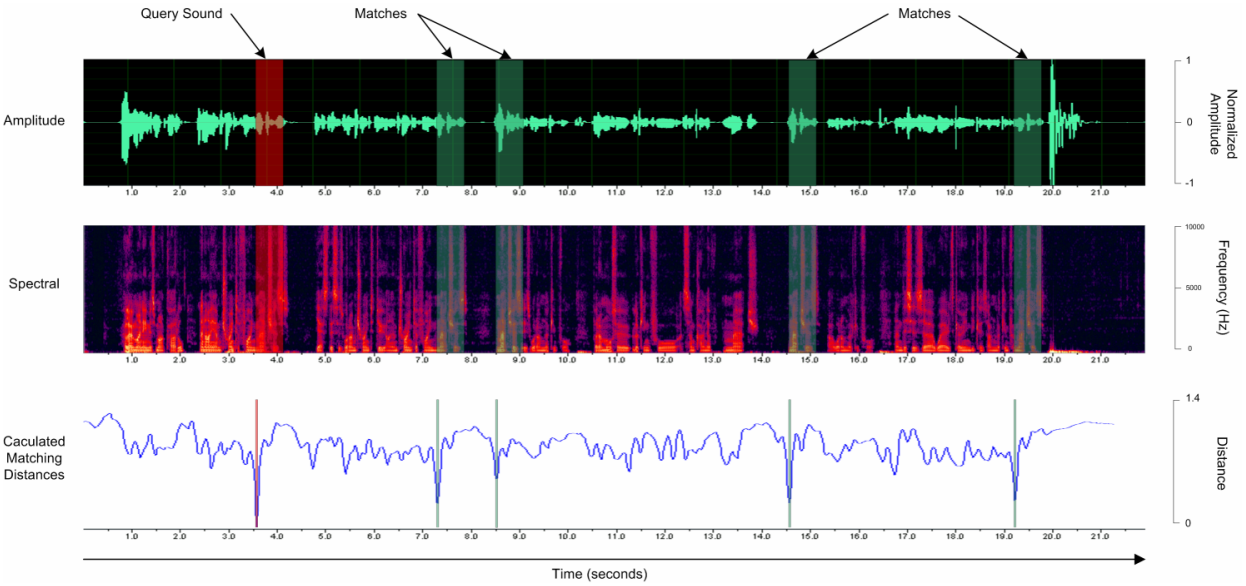

Figure 31: **Audio Trajectory Matching**. The user first selects the query sound (*in red*) from the amplitude waveform (*top*) or spectral representations (*bottom*) of the input soundtrack. The similarity to the query is then calculated over the trajectory in MFCC feature space. In this example, only the matches from the resulting similarity curve were kept (*in green*). These correspond to the best four minimum values of the distance curve.

3.3.3.3 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 a high probability of having the same sounds for the same motion events as those in the sample animation. The method is depicted in Figure 32.

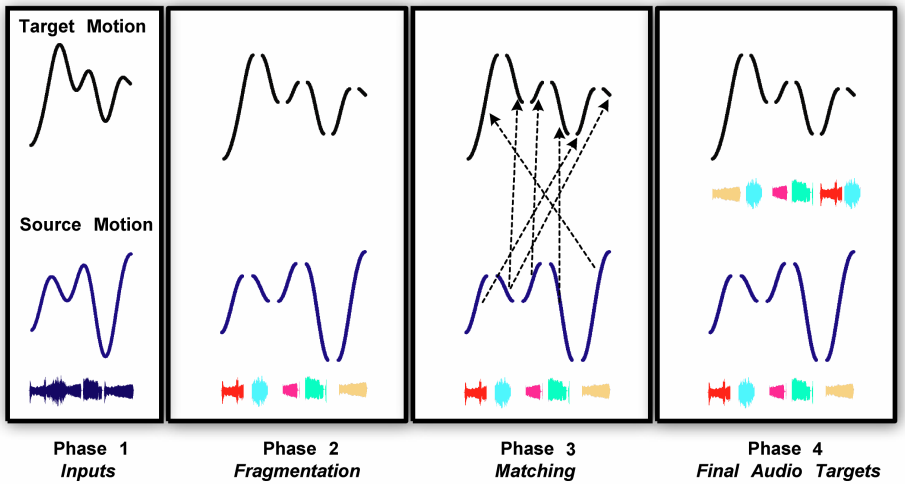

Figure 32 **Phases involved in fully-automatic control**: (*Phase 1*) The source motion, its soundtrack and the target motion are entered. (*Phase 2*) Both motions are broken up at sign changes in their first derivative. (*Phase 3*) Matches between the source and target motion fragments are found. (*Phase 4*) Each fragment in the target motion now has a source audio segment assigned to it. These audio fragments are used as synthesis constraints to generate the soundtrack for the target animation.

We therefore need to determine which portions of the source motion best match with those in the new, target motion. This is achieved by repurposing the motion matching algorithm recently presented by Pullen and Bregler [2002], originally used for motion enhancement. 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. We then calculate the Euclidean distance between each target fragment to every other source fragment. Note that we ignore fragments under one fourth as short or over four times as long as the target fragment being matched. Only the K closest matches (we use K=5) for each target fragment are kept (see Figure 33 and Figure 34). If the motion is multi-dimensional (such as motion capture) the splits occur at changes in derivative of the motion's barycentre, and the total fragment distance is summed over all dimensions.

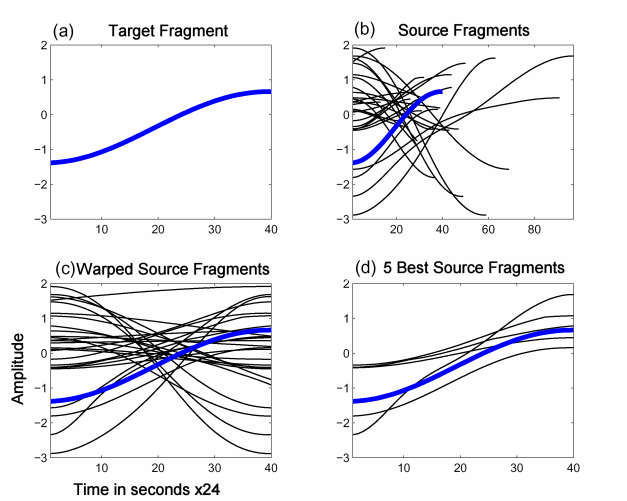

Figure 33: **Motion Fragment Matching Process**. (*a*) In Blue, the target fragment to be matched. (*b*) All valid source fragments to which the target fragment must be compared to. (c) The same source fragments are time-warped to be of the same length as the considered target fragment. (d) Only the top 5 closest matches are used in the optimal path calculation.

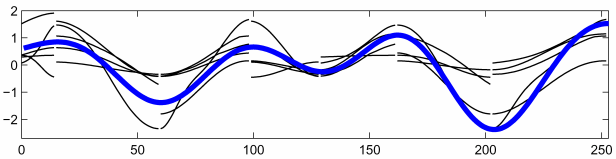

Figure 34: **K-closest matches**. For each fragment of the target motion (in blue), the 5 closest source fragments are displayed.

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 as maximizing the selection of the top *K* closest matches. For each target fragment, we evaluate the cost of transitioning from one *K*-closest source fragment to another. A penalty score of 1 is given to non-consecutive source fragments, and 0 for consecutive ones. The algorithm finds all possible combinations of fragments that go through the points of zero cost. Figure 35 illustrates the case where only the top 3 nearest matches are kept. The top row shows the target fragments, along with their top 3 nearest source fragments in the bottom row. The blue lines indicate that these source fragments are consecutive in the original source animation, indicating a path of zero cost. Any of the following best paths can be chosen: 4-5-6-2, 4-5-4-5 and 1-2-4-5 since they have two instances of zero cost (i.e. consecutive fragments). Note that in path 4-5-6-2, fragment 2 is selected since it is the closest match.

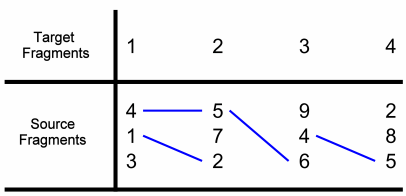

Figure 35: **Optimal path determination**. In this example case, we are looking for the optimal source fragment sequence for the given 4 target fragments. The top row shows the target fragments, along with their top 3 nearest source fragments in the bottom row. The blue lines indicate that these source fragments are consecutive in the original source animation, indicating a path of zero cost.

Given the optimal fragment path, 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, better results are obtained when preferred nodes have weights just over 0. This is because hard-constraints can produce artefacts 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 higher quality if there are sections in time where fragments that were consecutive in the source data are used consecutively to create the path. As we have seen, this is accommodated by Pullen and Bregler's [2002] method as the algorithm considers the neighbours of each fragment, and searches for paths that maximize the use of consecutive fragments.

3.3.4 Conclusion

In this section, we have introduced multiple interaction modes to 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. 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.

3.4 Natural Grain-Based Sound Synthesis

The following section details a controlled sound synthesis method based on Hoskinson and Pai's [2001] natural grain-based synthesis. The motivation was to determine if a different synthesis paradigm would yield better quality results over a wider variety of sound types. Natural grain-based synthesis operates at a coarser level by first analyzing and segmenting the input into variable-sized chunks that are recombined into a continuous stream, where each chunk is statistically dependent on its predecessor.

Although the synthesis mechanism is radically different from the wavelet-approach, the interaction is similar. Synthesis constraints are defined in the same manner for the manual, semi-automatic and fully automated interaction modes. In addition, a new mode of operation is introduced that uses audio to constrain the synthesis process.

3.4.1 Natural Grain-Based Unconstrained Sound Synthesis

The general approach taken here is to find places in the original soundtrack where a transition can be made to some other place in the soundtrack without introducing noticeable discontinuities. A new sound is synthesized by stringing together segments from the original soundtrack. This synthesis method can be broken up into two stages: the analysis phase and the synthesis, or recombination, phase.

3.4.1.1 Analysis Phase

In the analysis stage, the sound is segmented to determine *natural* transition points. The sound between these transition points is considered atomic and not broken up any further. The soundtrack is broken up into short windows for which the energies of each of the first six levels of wavelet coefficients are calculated. The scheme is derived from Alani and Deriche [1999], originally used to segment speech into phonemes. The idea is to find local changes in the audio by measuring the correlation between corresponding wavelet levels across adjacent frames.

The soundtrack is first divided into small windows of 1024 samples, with 256 samples of overlap. For an input at 44Khz, each frame corresponds to 23.2 ms. A frame is characterized by its top six levels of the

wavelet transform coefficients. The energy of each level between two frames is given by the sum of the absolute values for each level of coefficient differences. These energies are used to measure the correlation between adjacent frames throughout the soundtrack. A Euclidean distance over 4 frames is used to measure the strength of transition between frames *a* and *b*, as given below:

$$D(f_a, f_b) = \sum_{i=a-1}^{a} \sum_{j=b}^{b+1} \sum_{k=1}^{6} \left(\frac{X_{i,k}}{\sum_{p=1}^{6} X_{i,p}} - \frac{X_{j,k}}{\sum_{q=1}^{6} X_{j,q}} \right)^2$$

where *i* and *j* are the frame numbers, *k*, *p* and *q* are the wavelet levels and $X_{i,k}$ and $X_{j,k}$ are the energies of wavelet coefficient levels on level *k* of the input sound. The denominators correspond to normalisation terms that divide each level's energy by the sum of energies in that frame. This focuses the correlation on the differences in strength of bandwidths between frames. The process is depicted in Figure 36 for frames *a*=2 and *b*=3.

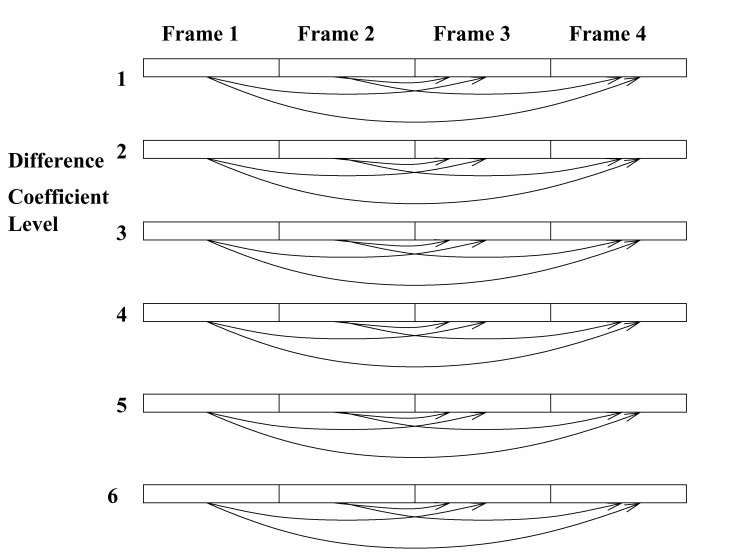

Figure 36: **Correlation measure between frames *a*=2 and *b*=3**. The Euclidean distance between each wavelet coefficient level is calculated and added up to obtain an overall distance measure between frames. The arrows represent difference calculations between matching levels. Four frames are necessary to calculate the distance between frames 2 and 3.

Once all coefficient levels are compared, the differences between them are summed to yield the total difference measure between two frames. This results in *n*-3 distance measures for a soundtrack with *n* frames (since the first and last two are ignored) representing the degree of change between a frame and its immediate neighbours.

Segmentation points are then identified by finding the local minima in the sequence of frame distance measures. These transition points correspond to locations in the audio where it changes the least suddenly. They determine the *natural grains* forming the input, the granularity of which is controlled by the user. A threshold is given to this end, so that the differences that are below it are taken as new grain boundaries. For small thresholds, splits at points of little change are thus favoured, whilst those at points of relatively large amounts of change are discouraged. The threshold defaults to 25% of the total possible number of grains ordered by the size of their local minimum, as in the case depicted in Figure 37. All transitions with an inter-onset interval under 4 frames are ignored.

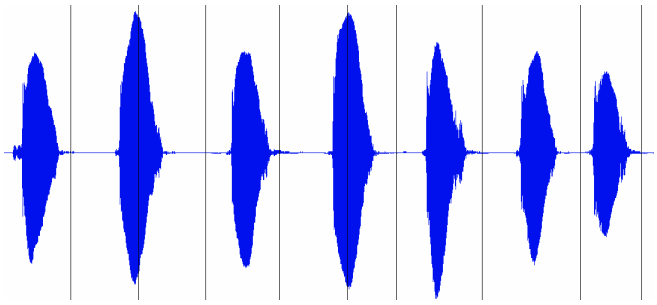

Figure 37: **Example segmentation of a waveform.** The vertical black lines indicate points of segmentation.

3.4.1.2 Recombination Phase

Before synthesis can start, we need to establish which grains flow most naturally from any given grain. Only after this can randomized variants of the input soundtrack be generated that retain as many of its original characteristics as possible. To this end, the method featured above to calculate inter-frame energy distances is used once again. The distance metric allows us to construct probabilities of transition between every grain by comparing the distance between the end of the grain to the beginning of all other grains.

The sound texture is essentially a Markov process, with each state corresponding to a single grain, and the probabilities P_{ij} corresponding to the likelihood of transitions from one grain i to another j. Let $p_{ij} = 1/\mathrm{D}(i,j)$ where $\mathrm{D}(i,j)$ is defined as the differences in normalized energy metric of the last two windows of i and the first two windows of j. All the probabilities for a given row of P are normalized so that $\Sigma_j P_{ij} = 1$, then:

$$P_{ij} = \frac{p_{ij} + C}{\sum_{j=0}^{n} p_{ij} + nC}$$

where n is the number of grains and C is a damping constant. The magnitude of certain probabilities might be significantly greater than others and cause repetitions in the synthesized sound. This is alleviated with the damping constant C which helps to even out the grain weightings. Higher values of C will give smaller probabilities more chances of being picked, whilst de-emphasizing (though never totally) the preference towards highly probable grains. The user can alter the noise constant C to affect the randomness in the generated audio.

The synthesis step does a Monte-Carlo sampling from the empirical distribution P (Figure 38) to decide which grain should be played after a given grain. In this manner, the smoother the transition between the current grain and the next, the higher the probability that this grain is chosen as its successor.

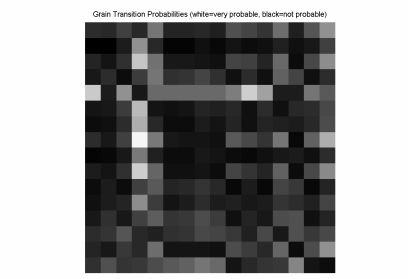

Figure 38: Example of a generated transition probability matrix. Lighter squares indicate higher transition probability, and inversely, darker squares indicate low transition probability.

A series of grain indices are produced during the synthesis steps. Before the corresponding audio sample chunks for each grain are recombined to form the final output, cross-fading over two frames (about 5ms) is carried out between successive grains. Since energies are normalized before comparison, boundary amplitude changes are not given much weight in the synthesis process. Cross-fading gives a better general cohesiveness to the output and eliminates artefacts due to misalignments in signal amplitudes.

3.4.2 Directed Sound Synthesis

The above algorithm can synthesize both stochastic and periodic sound textures. However, it suffers from the same problem as our initial wavelet-based algorithm, in that no control is possible over new sound texture instances. Once again, high-level user-control over synthesis is introduced. This is achieved by weighting the contributions of different regions in the source signal, in much the same way as for the wavelet-based synthesis in Section 3.3.2. Again hard or soft constraints are used to enforce the user-

preferences. In this section, we first look at how these synthesis constraints are defined, and then by what means they are enforced in our constrained algorithm.

3.4.2.1 Constraint Specification

The user can specify which types of sounds from the input soundtrack should occur when, and for how long, in the output synthesized soundtrack. As with the wavelet-based approach in Section 3.3.2.1 , a series of source and target segments is specified by the user. Again, the number, length, overlap and combinations of source and target segments are unrestricted. Exclusion constraints and unconstrained areas are also permitted. The user associates a probability with each constraint so as to control its influence on the final sound. The weighting scheme also follows the ranges [-1, 1] with hard-constraints defined for weights -1 and 1. The weights of overlapping targets are added up so that grains that satisfy multiple constraints are even more or less likely to occur.

Therefore, each separate constraint $c \in \{1,2,...,n\}$ defines one or more source segments s_c from the input sound and one or more target segments T_c in the new audio. s_c and T_c are defined by a series of locations in milliseconds. The piecewise weighting curve ω_c defines the likelihood of audio from the source s_c appearing at every instant of T_c. Once the input audio has been segmented, the grains included in time segments s_c form S_c, the set of source grains for constraint c. Figure 39 illustrates the process. Note that any mismatches between grain boundaries and segment boundaries are resolved by choosing the nearest grain boundary to that of the segment's.

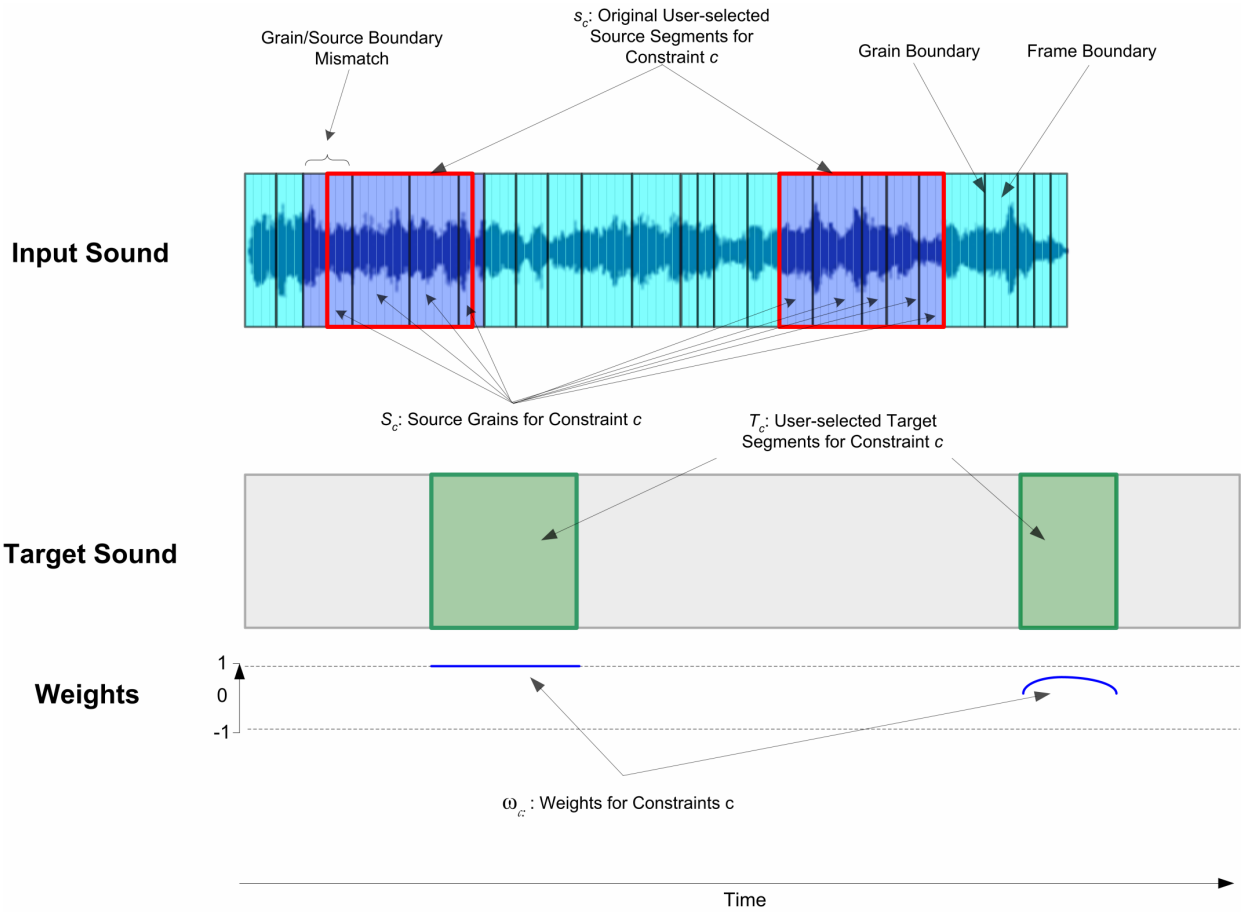

Figure 39: **Parameters defining a constraint *c***. (*Top*) The two red squares delineate the source segments s_c as defined by the user. All grains included in the Red areas are assigned to S_c, the set of source grains for constraint c. Any mismatches between grain boundaries and segment boundaries are resolved by choosing the nearest grain boundary to that of the segment's. (*Middle*) The target segments are shown in Green. (*Bottom*) A weight is assigned to each target segment above.

3.4.2.2 Hard and Soft Constrained Synthesis

Directed synthesis is achieved by dynamically biasing the grain probabilities P_{ij} in the Markov table during synthesis so as to maximize constraint satisfaction. Since grains originating from the current targeted source are preferred for inclusion constraints, we therefore proportionally increase their associated likelihood of being selected. Conversely, for exclusion constraints we proportionally decrease their likelihood of being selected. This is done by scaling P_{ij} proportionally to the weights in each constraint.

Let i be the last generated grain and t the current temporal synthesis position in the target sound, then we must rescale all the probabilities P_{ij} to the next grain j using the weights in ω_c for each constraint c. This is only necessary if T_c is defined at instant t.

Calculating the weight for each grain

Before P_{ij} is rescaled, the final weight ${}^{j}W''(t)$ over all constraints assigned to grain j must be determined. First, the weight of grain j over a single constraint c is obtained. Since grains vary in size; we place each grain j at the current synthesis position t to determine the anticipated overlap between the weighting curve ω_c and grain j. The overlap is simply $(t, t+L_j)$ where L_j is the length of grain j in milliseconds. Therefore, the weight ${}^{j}W_c(t)$ of grain j over a single constraint c at position t is defined as:

$$ {}^{j}W_c(t) = \frac{\omega_c(t, t+L_j)}{L_j} $$

where:

j is the current grain
L_j is the length in milliseconds of grain j
c is the current constraint
$\omega_c(.)$ is the weighting curve for constraint c
t is the current synthesis position in milliseconds

${}^{j}W_c(t)$ is normalised over the grain length to prevent longer grains from obtaining higher overall weights and disfavouring shorter grains in the synthesis. The resulting weight is only assigned to grains originating from the source grains S_c, therefore:

$$ {}^{j}W_c'(t) = \begin{cases} {}^{j}W_c(t) & if \quad j \in S_c \\ 0 & if \quad j \notin S_c \end{cases} $$

where:

S_c is the set of source grains for constraint c

The weight ${}^{j}W_c'(t)$ for grain j takes into account its membership to the constraint's source. Only the grains that belong to the constraint's source are affected by the constraint. The weight ${}^{j}W_c'(t)$ is calculated over all constraints *n*. Therefore, the final weight value ${}^{j}W''(t)$ for grain j is summed over all constraints:

$$ {}^{j}W''(t) = \min\left(\max\left(\sum_{c=1}^{n} {}^{j}W_c'(t), -1 \right), 1 \right) $$

where *n* is the total number of constraints with their targets T_c defined at instant t.

Since it is possible that a grain belongs to multiple overlapping sources from separate constraints, its weights are added up. We want the contribution of each constraint to be accumulated. For example, if constraint A has a weight of 0.3 for grain j at instant t, and constraint B has one of 0.5, then grain j (which belongs to both A's and B's sources) will have a cumulated weight of 0.3+0.5=0.8. The final weight is clipped so as not to lie outside the [-1,1] range.

The weight ${}^{j}W''(t)$ is calculated for every grain at each new synthesis position t. The effect here is that multiple constraints with overlapping targets can affect the choice of the next grain. For example, grains from both an explosion sound constraint and a gunshot sound constraint can compete for the same target region, while simultaneously disfavouring grains from a laughing sound constraint.

Rescaling the probabilities

Once ${}^{j}W''(t)$ is obtained for each grain, we can rescale the probabilities of all grains in the following manner:

$$P'_{ij} = \left({}^{j}W''(t) + 1 \right)^{\eta} \cdot P_{ij}$$

where:

P_{ij} is the transition probability from grain i to grain j

η controls the overall weight influence

P'_{ij} is normalised so that $\sum_j P'_{ij} = 1$. The overall weight influence η is user-defined. Bigger values of η enforce the constraints more strongly at the cost of randomness and audio continuity.

Hard-Constraints

The resulting probabilities do not take into account hard inclusion and exclusion constraints. Consequently, hard-constraints are enforced over the resulting transition probabilities before they can be used.

In the case where a candidate grain j has one or more occurrences of a hard exclusion '-1' weight over the interval $\omega_c(t, t + L_j)$ for any constraint c, its associated P_{ij} is set to 0. This ensures that grains with hard exclusion constraints will not be selected.

In the other case where a candidate grain j has one or more occurrences of a hard inclusion '1' weight over the interval $\omega_c(t, t + L_j)$ for a constraint c, the probabilities of all other grains not belonging to the source S_c of constraint c are set to 0. This ensures that only the hard included grains will be selected.

3.4.2.3 Anticipation

Smooth transitions near boundaries between unconstrained and hard-constrained areas can be difficult to achieve. The hard-constrained areas drastically reduce the grain candidate set, forcing the selection of grains with low probabilities when jumping from unconstrained to hard-constrained areas and vice-versa. Anticipating hard-constraints in the synthesis avoids such situations. Anticipation works by propagating grain selection preferences back in time from an approaching hard-constraint '1' area in T_c to the current synthesis position t. The goal is to favour grains that have a high probability of having a successor grain included in S_c. Therefore, a look-ahead mechanism is triggered when the current synthesis position t is less than a distance d from the start of a hard-constrained area T_c where:

$$d = \left\lceil \frac{s_{sound} \cdot \beta}{s_{grain}} \right\rceil \cdot s_{grain} \quad \text{and} \quad s_{grain} = \frac{s_{sound}}{n_{grain}}$$

where:

s_{sound} is the total length of the input sound
n_{grain} is the number of grains detected during segmentation
s_{grain} is the average grain size
β controls the anticipation distance (usually set to 5%)

Anticipation works by backtracking from the start of T_c up until the current synthesis position t with a step-size of s_{grain} and using the grain preferences to bias the grain choice at point t. The effect is that, gradually, grains with a high probability of having distant successors in S_c will be favoured.

Before synthesizing the last unconstrained grain i before the start of T_c, we first find the set of top *n* grains $\mathbf{a}_1$ with the highest likelihood that their next grain j belongs to S_c. Grains in $\mathbf{a}_1$ are favoured by increasing their respective probabilities. These preferences can be propagated back to the current synthesis t even if it is not directly before the start of T_c. We simply backtrack in time by s_{grain} and find the top *n* grains $\mathbf{a}_2$ with the highest likelihood that their next grain belongs to $\mathbf{a}_1$. This continues *r* times until we reach t where $\mathbf{a}_r$ is used to bias the choice of the next grain (see Figure 40 where *r* is set to 3). Before renormalisation, the probabilities of the top *n* grains $\mathbf{a}_r$ are scaled in the following manner:

$$P'_{ij} = P_{ij} + \alpha \cdot P_{ij}\left(\frac{r \cdot s_{grain}}{d}\right)$$

where the user-threshold α controls the impact of anticipation on the probabilities (usually α is set to 30%). The closer t is to the start of T_c, the more the anticipation process will influence the transition probabilities for the next selected grain.

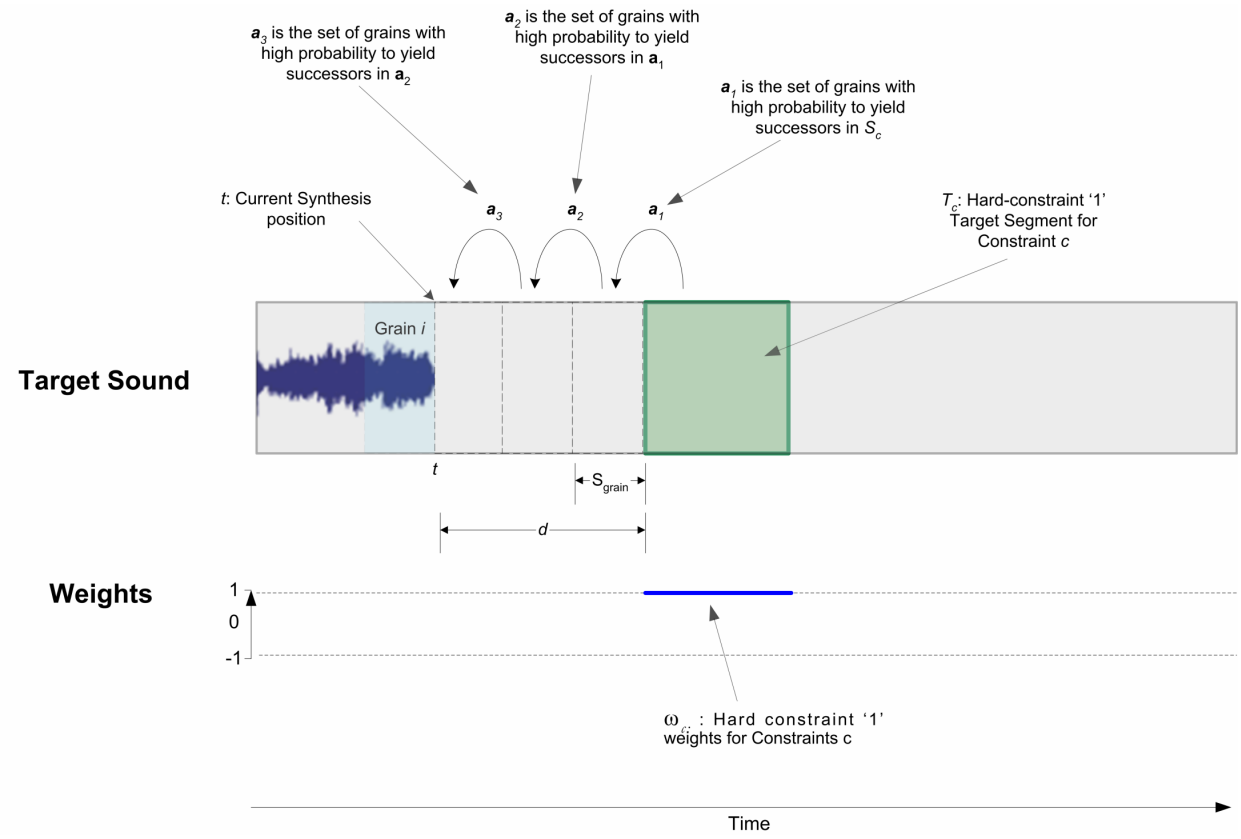

Figure 40: **Anticipation Process**. The transition probabilities P_{ij} from the current grain *i* to the next grain *j* are rescaled to favour smooth transitions from unconstrained areas to hard-constrained '1' areas. The anticipation probability rescaling process is activated when a hard-constrained area is detected a distance *d* from the current synthesis position *t*. The grains with highest probability of having successors satisfying the constraint are propagated back to the current synthesis position. Here, all the grains contained in the set $\mathbf{a}_3$ will have their weights rescaled.

3.4.3 User Control

The grain-based approach benefits from all the interaction methods available to its wavelet-based counterpart. In all interaction modes, the constraints are defined in the same manner.

The major change over the wavelet-based approach is in the granularity of the constraints. Instead of snapping user-constraint boundaries to the nearest leaf coefficient of the wavelet-tree, they are snapped to the closest grain boundary. This results in a coarser match between the region that the user defines and that is actually employed by the system. An obvious way to increase the snapping fineness is to decrease the grain size. Simply lowering the segmentation threshold in Section 3.4.1.1 has this effect. The negative side-effect of smaller grains is that more audio discontinuity at the higher grain-level is introduced (further discussed in Section 4.1). In practice, only limited grain-size reduction is necessary. Since grains try to capture separate audio events, the *natural* points of transition tend to match up to user-selection boundaries.

3.4.4 Sound Transfer

Sound transfer is conceptually similar to texture transfer [Ashikhmin 2001, Efros and Freeman 2001, Hertzmann et al. 2001], where an image is re-synthesized with elements of another given texture. The goal here is to use audio as the synthesis constraint so as to produce a new sound texture that exhibits some similar property (such as volume or pitch) to that of a separate guiding soundtrack A_{guide}. Figure 41 and Figure 42 show the conceptual similarity between texture transfer and sound transfer. A related problem was addressed in [Zils and Pachet 2001] where a new sequence was constructed by assembling sound samples in a large sound database, by specifying only high-level properties. The difference here is that sounds are generated by finding the sound that best fits the current high-level properties, while our work generates new sequences based on the statistics of a given input sound.

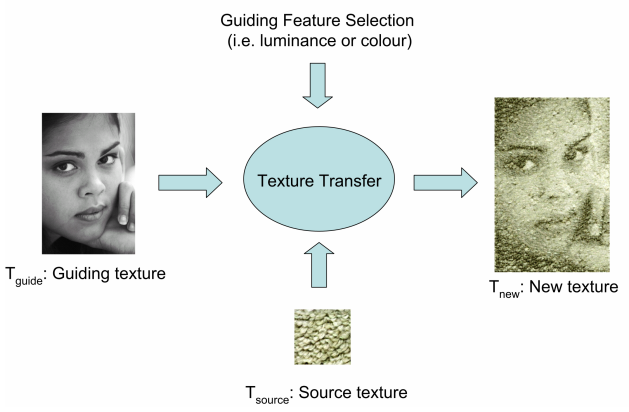

Figure 41: **Texture Transfer Process Overview.**

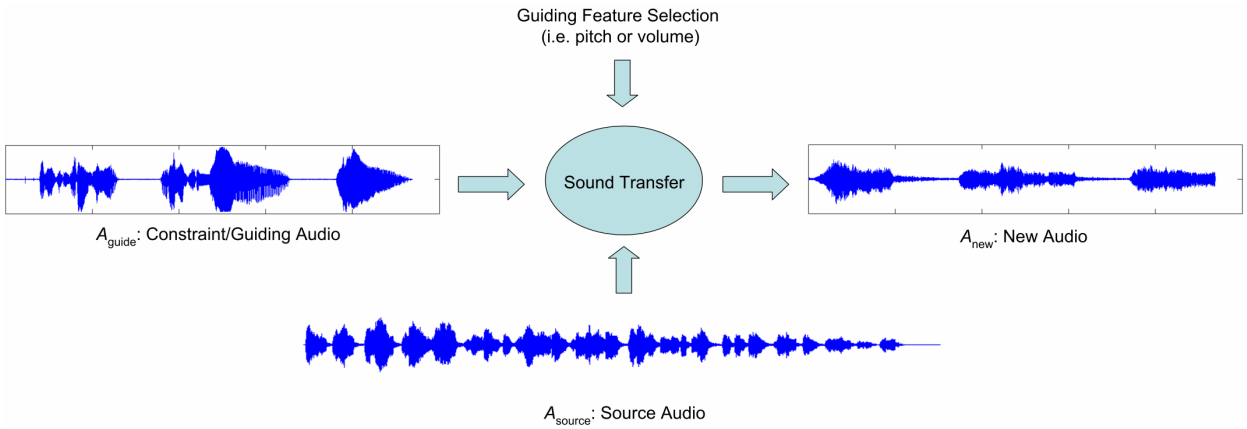

Figure 42: **Sound Transfer Process Overview**

Up until now, to synthesize a new sound A_{new}, a source sound A_{source} and a set of source/target constraints were required. Instead here, we replace the soundtrack A_{guide} with another one A_{new}, built up from sounds in A_{source}, by simply matching some feature of the old soundtrack A_{guide} with that of the new soundtrack A_{new}. That way, we can change the building blocks of a soundtrack, without changing its overall properties. The process is summarized below in Figure 43 for a voice-driven example. In this example, the RMS volume of voice sample is used to drive the synthesis from a *screaming* source soundtrack. A plot of the guiding and synthesized sounds, along with their RMS features, is shown in Figure 44.

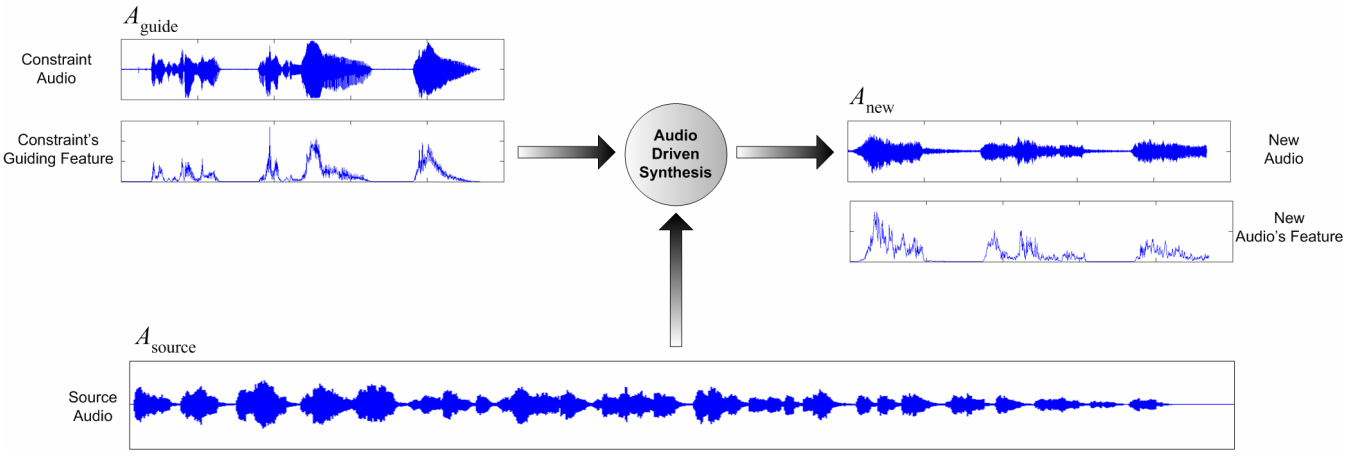

Figure 43: **Audio-driven synthesis steps**. A guiding audio's audio feature constrains the synthesis from a source audio sample.

Let i be the last generated grain and t the current temporal synthesis position in A_{new}, then we must rescale all the probabilities P_{ij} to the next grain j so as to maximize the likelihood that grain j will exhibit similar audio properties to that of A_{guide} at the same position t.

The user first defines the audio feature, or weighted combination of audio features, to use for the matching. This could be the Mel-Frequency Cepstral Coefficients, RMS volume, short time energy, zero crossing rates, brightness, bandwidth or spectrum flux. These features are then pre-calculated for every sliding window position in A_{source} previously used in the segmentation algorithm. The same is also carried out over A_{guide}. The features are used to calculate the distance $D_j(t)$ between all the windows from A_{source} forming the potential next grain j and the corresponding windows in A_{guide}:

$$D_j(t) = \frac{\sum_{p=0}^{n_j-1}\sum_{k=1}^{K}\left(\omega_k(t+p)\cdot\left(\psi_k\left(A_{guide}(t+p)\right)-\psi_k\left(A_{source}(t+p)\right)\right)^2\right)}{n_j}$$

where:

$A_{guide}(t)$ is the window from A_{guide} at position t
$A_{source}(t)$ is the window from A_{ource} at position t
j is the potential next grain
n_j is the total number of windows forming grain j
K is the total number of constraining audio features
$\psi_k(x)$ extracts audio feature k from the given window x
$\omega_k(x)$ is the weighting curve for each constraining audio feature k for the given window x

A set of audio features can potentially influence the final distance $D_j(t)$. The impact of each constraining audio feature can be changed over time in much the same way as for source/target constraints detailed earlier in Section 3.4.2.1 . The weighting curve $\omega_k(x)$, assigned to each constraint audio feature k, is used for this purpose. Individual features can be given more impact on the calculated distance so that their influence on the final synthesized output is conveniently configurable. Weights can be negative so that grains that are similar to the target sound are discouraged.

Before synthesizing each new grain, $D_j(t)$ is evaluated for all potential next grains and its renormalized value is used instead of weight ${}^{j}W''(t)$ in the probability rescaling equation in Section 3.4.2.2 . This yields the following probability rescaling function:

$$P'_{ij} = \left(D_j\left(t\right)+1\right)^{\eta} \cdot P_{ij}$$

where η controls the overall weight influence. The synthesis then proceeds as normal with these rescaled weights.

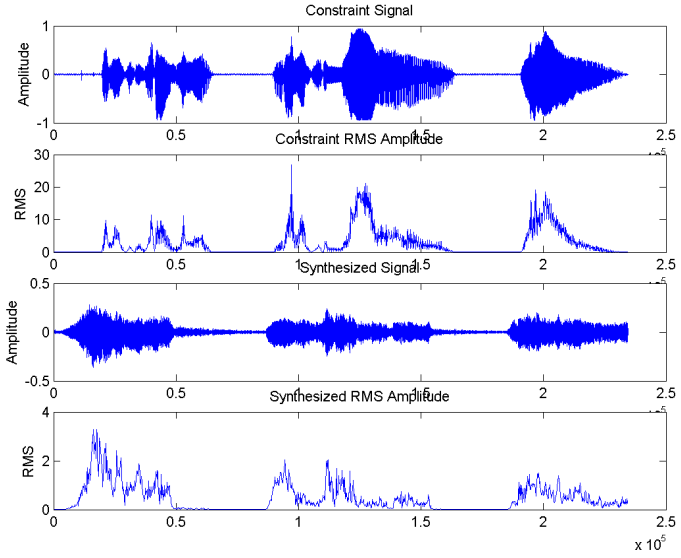

Figure 44: **Example audio-driven synthesis output**. The RMS volume of voice sample is used to constrain the synthesis of the new sound from a *screaming* source soundtrack. Notice how the RMS volume of the synthesized soundtrack roughly matches that of the guiding voice sample.

3.5 Self-Similarity-Based Sound Synthesis

The potential of self-similarity-based sound synthesis is investigated. This synthesis paradigm is extended to a controllable model in much the same way as for our previous natural-grain based model. The goal is to investigate if better synthesis quality results over a wider-variety of sound types can be obtained compared to both the wavelet and natural grain based approaches. This section follows the same structure as for the natural-grain approach in Section 3.4.

3.5.1 Self-Similarity-Based Unconstrained Sound Synthesis

Lu et al. [2002] presented a new sound texture approach that makes use of the inherent self-similarity in sounds. The premise is analogous to that of Hoskinson and Pai [2001] where the sound is segmented into smaller chunks and statistically recombined to limit audio discontinuities. The difference between the two algorithms lies in their segmentation and sound continuity measures. The natural grain-based approach segments along points of least change, whilst the self-similarity approach uses auto-correlation to find onsets of distinct sound events. Transition probabilities are now based on perceptual continuity across whole grains, and not over the amount of audio change in the first and last portions of grains. The self-similarity algorithm therefore divides into two stages: the analysis phase and the synthesis, or recombination, phase.

3.5.1.1 Analysis Phase

In the analysis stage, Lu et al. [2002] find the building patterns in an input soundtrack by identifying the pattern breakpoints using auto-correlation. A spectral measure, called Mel-Frequency Cepstral Coefficients (MFCC) [Logan 2000], is used to characterize the sounds during correlation. A self-similarity matrix is then derived by calculating the difference from every frame's MFCC of the input audio to every other. The self-similarity novelty curve, describing the audio change in the input sound, is then extracted from the matrix. The maxima of this curve correspond to points of maximum audio change such as pattern breakpoints. By breaking up the input sound along these points, the resulting grains form the input's characteristic building patterns.

3.5.1.1.1 Mel-Frequency Cepstral Coefficients (MFCC)

The segmentation scheme first extracts an MFCC representation of audio before the similarity between two audio instants is calculated. MFCC are short-term spectral-based features and are the dominant features used for speech recognition [Gold and Morgan 2000]. They have also been used successfully for timbre analysis in the music domain [Cosi et al. 1994] and for content-based retrieval [Foote 1997]. MFCC are frequently preferred over the standard FFT since it is a perceptually motivated and more compact parameterisation of sound. Figure 45 depicts the process of creating MFCC features for an input soundtrack.

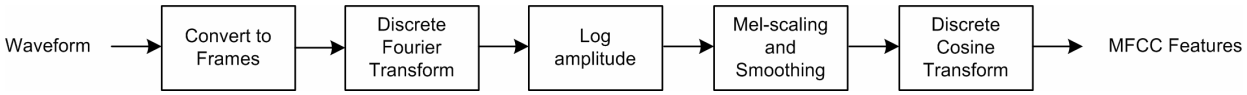

Figure 45: MFCC creation process.

The first step is to convert the signal into short 23.2ms frames. The aim is to generate a cepstral feature vector for each frame. The Discrete Fourier Transform (DFT) is calculated for each frame. We then retain only the logarithm of the amplitude spectrum because the perceived loudness of a signal has been found to be approximately logarithmic. The next step is spectral smoothing and Mel scaling to emphasize perceptually meaningful frequencies. The Mel-scale is based on a mapping between actual frequency and perceived pitch as the human auditory system does not perceive pitch in a linear manner. The spectral components are combined into a smaller number of frequency channels, whose spacing complies with the Mel-scale of perceived pitch. The mapping is approximately linear below 1kHz and logarithmic above. This is achieved by passing the spectrum through a filter bank consisting of 13 linearly spaced filters (133.33 Hz between centre frequencies) and 27 logarithmically-spaced filters (separated by a factor of 1.0711 in frequency). For a window length of 256 samples, the DFT generates 256 frequency bins that are reduced to 40 Mel-scale components, defined as the Mel spectrum. The process is depicted in Figure 46.

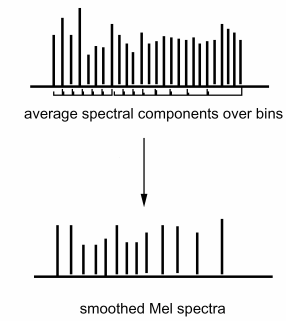

Figure 46: **Rescaled log amplitude spectrum**. Each new bin is the averaged local spectral components, spaced following the Mel frequency scale. The resulting spectrum is therefore smaller and smoother.

The components of Mel-spectral vectors for each frame are highly correlated. So the last step of MFCC feature generation is to de-correlate and compress their components by applying Discrete Cosine Transform (DCT). This process yields 13 cepstral coefficients for each frame, defined as the Mel cepstrum. Since the DCT is performed with respect to frequency instead of time, the terminology is changed from spectrum to cepstrum. A plot of both the amplitude waveform and corresponding MFCC representation is given in Figure 47.

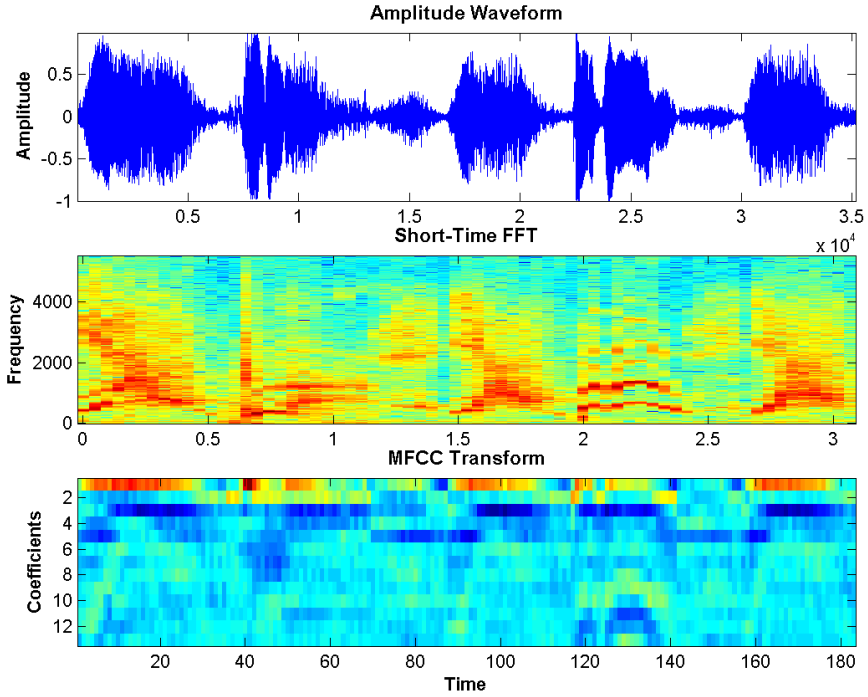

Figure 47: **MFCC parameterisation**. (*Top*) The amplitude waveform of an example sound. (*Middle*) A short-time Fourier transform of the sound. (*Bottom*) The corresponding MFCC transform of that sound with 13 coefficients per frame.

3.5.1.1.2 Similarity Measure and Self-Similarity Matrix

Given the MFCC for every frame of the input sound, the self-similarity matrix is then extracted. The matrix S_{ij} measures the similarity between pairs of cepstral vectors V_i and V_j calculated from frames *i* and *j*. V_i and V_j are the MFCC feature vectors of frames *i* and *j*. S_{ij} is based on the scalar (dot) product of the vectors, also referred to as vector auto-correlation. It will be large if the vectors are both large and similarly oriented. To remove the dependence on magnitude, the product is normalized to give the cosine of the angle between the parameter vectors. Therefore, S_{ij} is defined as:

$$S_{ij} = \frac{V_i \cdot V_j}{\|V_i\| \cdot \|V_j\|}$$

Frames and their associated feature vectors occur at a rate much faster than typical perceptual events, so a better similarity measure is obtained by computing the vector auto-correlation over a window of neighbouring frames. This also captures the time dependence of the vectors. S therefore becomes S' defined as the weighted sum of the autocorrelation vectors over the previous m and next m neighbouring temporal frames with binomial weights $[-w_m,\ldots,w_m]$:

$$S'_{ij} = \sum_{k=-m}^{m} w_k S_{i+k,\, j+k}$$

To result in a high similarity score, vectors in a window must not only be similar but their sequence must be similar also. The analysis proceeds by comparing all pair-wise combinations of audio frames and embedding the results in the self-similarity matrix S', as shown in Figure 48. The axes of S' represent time, running both horizontally (left to right) and vertically (top to bottom). Self-similarity is maximal and symmetric along its main diagonal.

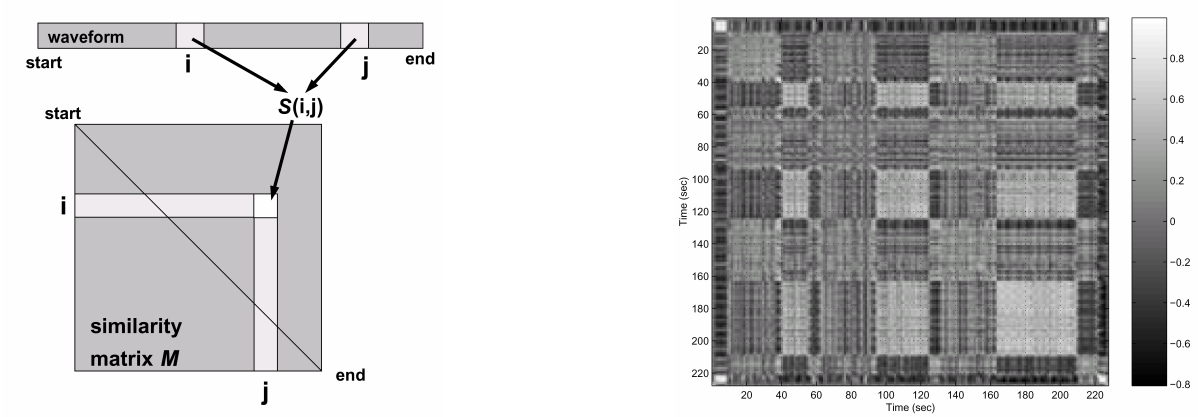

Figure 48: (Left) Process to create the self-similarity matrix S'. The pair-wise similarity between all frames *i* and *j*. *(Right)* Example self-similarity matrix computed for U2's *Wild Honey* soundtrack.

The visualisation of S' in Figure 48 uses the similarity measure to determine the pixel's greyscale. Pixels are coloured brighter with increasing similarity, so that segments of similar audio samples appear as bright squares along the main diagonal (from top left to bottom right).

3.5.1.1.3 Self-Similarity-based Segmentation

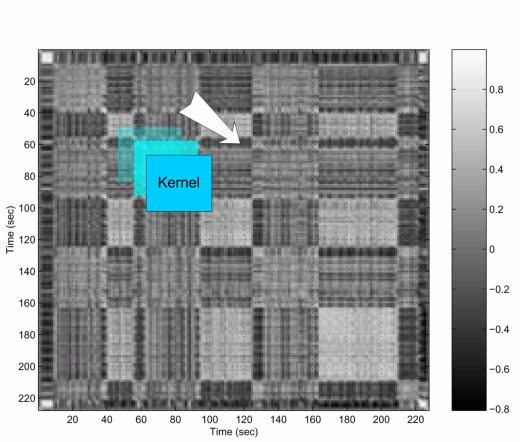

Figure 49: **Sliding Kernel Mechanism**. At each increment, the cross-correlation kernel slides one time unit along the main diagonal; the evaluated result is plotted on the novelty curve for that corresponding point in time.

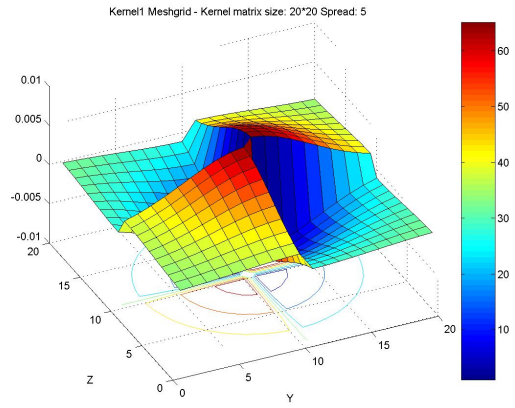

Figure 50: **Cross-correlation Kernel**. Plot of a 20 x 20 checkerboard correlation kernel with a radial Gaussian taper having $\sigma = 0.3$.

The self-similarity matrix is used to extract a novelty measure of the audio [Foote and Cooper 2001], which is the basis for the audio segmentation. It is extracted by sliding a cross-correlation kernel *K* along the main diagonal axis as portrayed in Figure 49. Points of greater change in the music are identified by maxima in the resulting novelty curve *N*. Hence *N* is defined as:

$$N(i) = \sum_{m=-w/2}^{w/2} \sum_{n=-w/2}^{w/2} K_{m,n} S_{i+m,i+n}$$

where *w* is the size of the kernel. The extrema in the subsequent novelty curve, shown in Figure 51, correspond to the largest change in the music. These points coincide with the contact point of two square regions in Figure 48.

The cross-correlation kernel *K* is approximated by a Gaussian-tapered *checkerboard* kernel, an example of which is given in Figure 50. *K* is simply a checkerboard matrix made up of coherence and anti-coherence terms. The former term measures the self-similarity on either side of the centre point of the matrix. If both these regions are self-similar, this term will be high. The other term evaluates the cross-similarity between the two regions. High outputs occur with regions of substantial similarity, with little difference across the centre point. The Gaussian smoothing tapers the kernel towards zero at the edges.

The input is then segmented into grains at the peaks of *N*, which correspond to points of greatest change in the audio. To enable the re-synthesis process, only the transition probabilities between each grain are now required.

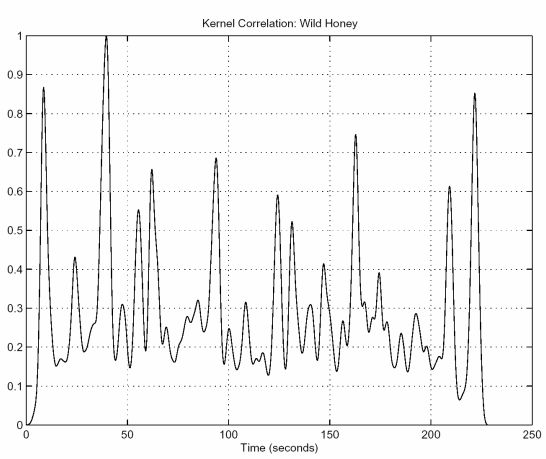

Figure 51: **Novelty score** for the self-similarity matrix in Figure 49.

3.5.1.2 Recombination Phase

The unconstrained synthesis process is much the same as that for the natural grain-based synthesis detailed in Section 3.4.1.2 . A distance metric allows us to construct transition probabilities between each grain. Instead of comparing the distance between the end of a grain to the beginning of all other grains as in Section 3.4.1.2 , the similarity over the total length of both grains is used. Unlike frames, the grains here are of varying lengths. Dynamic Time Warping could be used to compare grains, but a simplified method is used instead. Given that grain *i* contains *M* frames beginning from frame *i*, grain *j* contains *N* frames beginning from frame *j*, then the similarity T_{ij} between two grains is defined as:

$$T_{ij} = \sum_{k=1}^{M} w_k S'_{i+k,\, j+\left\lfloor k\frac{N}{M} \right\rfloor}$$

Better audio continuity is guaranteed if the temporally adjacent grains are considered in the final similarity measure. Therefore T_{ij} becomes

$$T'_{ij} = \sum_{k=-m}^{m} w'_k \, T_{i+k,\, j+k}$$

where the previous *m* and next *m* neighbouring temporal grains have binomial weights $[w'_{-m},\ldots,w'_m]$. $T'_{i+1,j}$ is mapped to probabilities through the exponential function:

$$P_{ij} \propto e^{\left(\frac{T'_{i+1,j}-1}{\sigma}\right)}$$

P_{ij} gives the transition probability from the i-th grains to the j-th grains where all the probabilities for a given row of P are normalized so that $\Sigma_j P_{ij} = 1$. The parameter σ controls the mapping between the similarity measure and the relative probability of taking a given grain. Smaller values of σ favour the most likely grains, while larger values of σ allow for greater variety at the cost of poorer transitions. The transition probability from grain i to grain j depends on the similarity between grain i+1 and j. Since the last grain l does not have a successor l+1, the probability for the last grain l depends on the similarity between grain l and j-1. The transition probability between the last grain l and the first grain f is undefined, so it is set to zero. A visualisation of T, T' and P is shown in Figure 52.

Again, the sound texture is essentially a Markov process, with each state corresponding to a single grain, and the probabilities P_{ij} corresponding to the likelihood of transitions from one grain i to another j. The synthesis step then does a Monte-Carlo sampling of P to decide which grain should be played after a given grain.

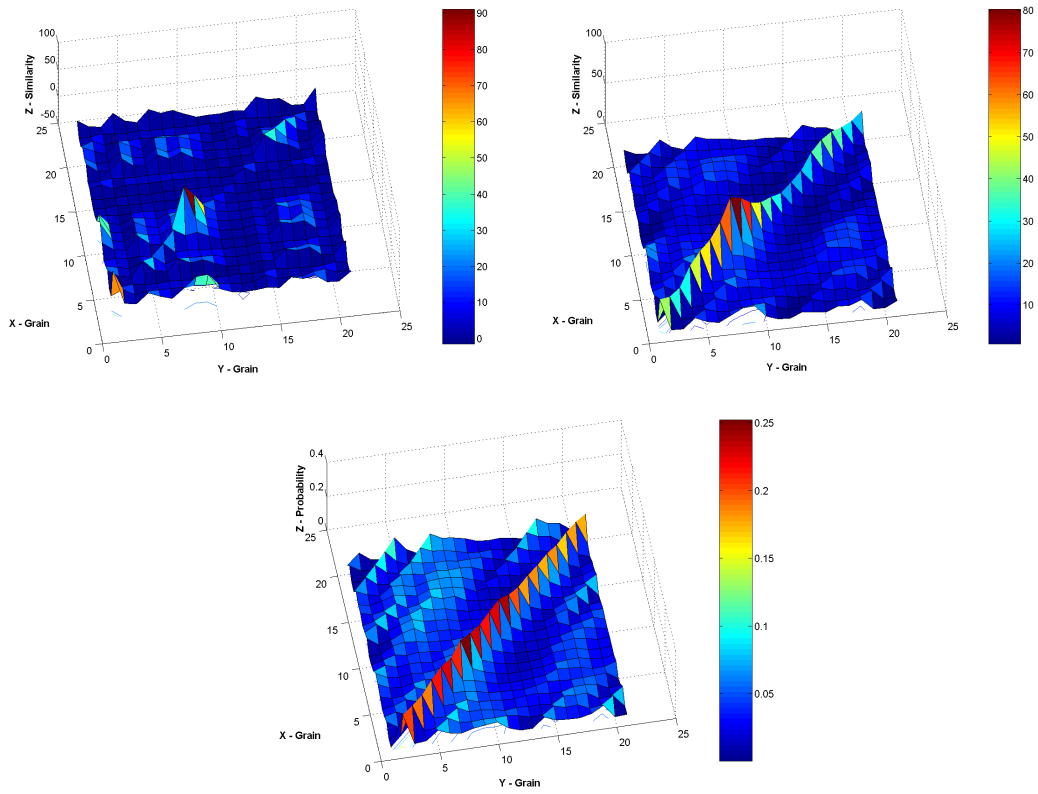

Figure 52: **Grain similarity and probability visualisation**. (*Top-left*) Height-field representation of the similarity table T. The similarity T_{ij} between grain i and grain j is indicated by the height Z at position X=i and Y=j. (*Top-Right*) T becomes T'' when the neighbouring grains are considered in the final similarity measure. (*Bottom*) P holds the grain transition probabilities obtained from T''.

3.5.2 Directed Sound Synthesis

Due to the similarity with the natural grain-based approach, extending the sound texture model above requires identical steps to those in Section 3.4.2. The same interaction modes are consequently available, including audio-driven synthesis.

3.6 Conclusion

The three novel sound synthesis methods are a step towards automated Foleying for film and television. Although it is feasible to use conventional audio software, such as *Pro Tools*, to manually generate short soundtracks, it rapidly becomes costly 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 *Pro Tools*. Also, in our system, constraints can later be changed on-the-fly leading to a full re-synthesis of the soundtrack.

The other major function of our system is to create soundtracks for games, animations and virtual environments based on the motions of objects within them. A correspondence between the motion and sound domains exists naturally in the real world. In physically-based sound models such as those of van den Doel [2001] and O'Brien et al. [2002], the correspondence between motion and sound parameter spaces is provided by physical laws. In such cases, the mapping is straightforward. However, when acoustic realism is not the goal, the mapping is not so direct. This is where our system becomes useful. The generality of our approach and the use of motion data allow dynamic synchronisation to visuals with little additional overhead. To this end, we present several interaction methods, with varying degrees of human intervention. The least automatic method still operates at a high level, requiring only that a few weighted correspondences between source and target regions be specified. The most automatic method requires only the setting of a few algorithm-tuning parameters. Hence, with minimal user effort, we can quickly create pleasing soundtracks appropriate to the motions they reflect.

Chapter 4

RESULTS

In the first part of this chapter, the three controllable sound texture models are evaluated with regard to sound quality, computational efficiency, controllability and applicability. We want to determine what conditions degrade the quality of the synthesized sounds and what can be done to prevent them. A series of examples demonstrating the control schemes are detailed.

4.1 Sound Editing Results

4.1.1 Segmentation

At first, it appears that the wavelet-based approach should work with a wider variety of sounds since no implicit sound model is used. However, some sounds might be swapped at perceptually inappropriate points such as in the middle of a laugh in our *Laughing* example on the DVD-ROM. Therefore perceptually-based models such as the natural grain and self-similarity based approaches, which identify natural points of transition, have an advantage. This depends on how hard it is to find these natural transition points, and if these points are model-dependent. In practice, we found that the suitability of automatically detected grain boundaries depends on the nature of the input sound.

The self-similarity-based segmentation is better at identifying pattern breakpoints. For example, in our *Jackhammer* example, it successfully groups jackhammer sounds together from the surrounding construction sounds. On the other hand, the natural-grain approach will segment at points of least audio change. The result is that the grouping of jackhammer sounds is lost since segmentation boundaries appear in the middle of the jackhammer sounds. Even though these boundaries might present better transition points from a local audio change perspective, the perceptual impact of breaking the jackhammer sounds up is more noticeable in the synthesized soundtrack.

By contrast, natural-grain segmentation is more effective in the case of our *Rain* example. The input soundtrack consists of tennis practice sounds during a rainy day. An instance of a ball striking sound, promptly followed by a net sound, is grouped into a single grain. This is because there is considerable audio change during this short extract. With self-similarity segmentation, this extract is broken into two separate grains: one for the ball striking sound and another for the ball hitting the net sound. The reason is that both these sounds are considerably different from a self-similarity perspective. The problem is that since the net sound is in its own grain, it might appear by itself without the prior striking sound in a synthesized soundtrack (although the probability would be small). This likelihood is eliminated with the natural-grain approach.

An advantage of the wavelet approach is that no segmentation threshold is necessary. With the coarser grained approaches, the default segmentation threshold is set to the best 25% of all detected grain boundaries. Increasing the segmentation threshold has the advantage that more variability is introduced in synthesized sounds, in turn allowing for better constraint satisfaction. The disadvantage is that too much segmentation can break audio continuity (such as in breaking up the ball strike from the net sound in the *Rain* example). Therefore, the default threshold typically needs readjusting because the optimum number of grains for a given input sound varies with its size, its inherent separability and the granularity of its synthesis constraints.

4.1.2 Continuity

Audio continuity in the wavelet-based approach is ensured simultaneously at a global, as well as a local audio event level. Since the synthesized tree is synthesized level-by-level from the root to the leaves, the algorithm can enforce continuity at multiple levels of resolution. It has no *a priori* knowledge of the input sound which is a strength as well as a weakness. Every node of the tree is given equal weight in the synthesis, be it perceptually significant or not. This increases the possibility that some sounds might be swapped at perceptually inappropriate points. Another problem with the wavelet approach is the introduction of convolution artefacts. These convolution artefacts arise because switching coefficients of the wavelet transform can sometimes have unpredictable results. Unless the changes are on the dyadic boundaries, it can slightly change the timbre of the input sound instead of switching the sound events seamlessly. Although infrequent, these changes are difficult to predict. They have to do with the wavelet filter choice and the position of the coefficients. Their frequency increases as the enforcement level of conflicting synthesis constraints is increased. The wavelet soundtrack synthesized for the *Cheering* example presents such artefacts.

Transitions, in both the natural grain and self-similarity models, are chosen between distant grains of the sample audio so that concatenating these grains does not introduce any discontinuity. Discontinuities may be classified by the granularity of their occurrence:

- **Sample-level discontinuities**

  At the smallest granularity, the audio sample, discontinuity is in the time domain. If the last few samples of the first grain are at their maximum value and the first samples of the second grain are at their

minimum value, an audible *click* may be heard. Both our coarse-grained approaches suppress sample-level discontinuities by cross-fading between adjacent grains.

- **Frame-level discontinuities**

In the natural grain model, only the first and last two audio frames between grains are examined to establish transition probabilities. We are guaranteed with this approach that the continuity between the last two frames and first two frames of sequential re-combined grains will be the greatest. It is therefore effective at limiting frame-level discontinuities. The self-similarity approach is less effective at limiting frame-level discontinuities since continuity is enforced at the grain level, not at the frame level. This is because, during synthesis, each new grain is picked to maximize its similarity across its *entire* duration to that of the last synthesized grain's following grain in the original soundtrack.

- **Grain-level discontinuities**

At the granularity of a grain, discontinuity can be described as the overall dissimilarity of the joined grains. Unnatural timbre fluctuations represents one aspect of dissimilarity. For instance, an explosion sounds very different from a scream. Transitioning from the middle of an explosion to a scream would create an abrupt change in timbre, awkward to the listener. The self-similarity approach is less prone to grain-level discontinuities since the nature of past and present grains determines future grains. Higher-level temporal continuity is enforced compared to the natural grain approach, which only considers the first and last two audio frames between grains to establish transition probabilities.

The appropriateness of each grain-based method depends on the input sound. If preserving the temporal continuity is crucial, then the grain-based approach is better suited since it will provide the smoothest transitions. For example, the *Medley* example's input sound consists of a series of unrelated cartoon sounds. Preserving the grain-level continuity is not essential since there is no extensive temporal continuity. What is important to enforce is the best possible frame-level continuity in this case. The self-similarity approach is better for sounds where the temporal continuity is preferable over frame-level continuity. For example, in the *Laughing* example, it is essential that the synthesized laughing sounds follow their original temporal order.

4.1.3 Repetitions

Repetitions are a particularly noticeable synthesis artefact. Natural-grain based synthesis is particularly prone to repetitions. For example, if the starting frames of a grain are similar to its ending frames, then that grain will tend to be repeated. Repetitions are less of a problem with both the wavelet and the self-similarity approach since temporal continuity is enforced. They will only occur if they appear in the original sound.

During constrained synthesis the candidate set of grains or nodes is typically limited increasing the possibility of repetitions. Therefore, in all approaches repetitions are discouraged by simply lowering the weights of grain or nodal candidates that have just been picked.

4.1.4 Controllability

In addition to sound quality, the sound texture must be steerable to comply with the user constraints. The first issue is how well the constraints are translated into the model. In the case of wavelet-based synthesis, very fine-grained (close to sample-level) snapping to the user-constraint boundaries is possible. Less accuracy is available in the natural-grain and self-similarity-based approaches since constraints are snapped to the closest grain boundary. Grain-size, in the natural grain approach, can be reduced to increase the probability of fitting the user-constraints, at the cost of longer term discontinuities. This effect is less apparent in the self-similarity approach since neighbouring grains are taken into account.

In highly-constrained cases where grain-level discontinuities are difficult to avoid, the self-similarity approach is not as effective. More frame-level discontinuities are introduced than in the natural-grain approach. This is due to the fact that the natural-grain approach minimizes frame-level discontinuities regardless of the nature of previously synthesized grains. It thus guarantees locally smoother sounds, but not necessarily with longer term meaningfulness and continuity.

Better results are obtained if the source and target regions are similar in length, otherwise unexpected results can occur. For example, a laughing sequence sounds unnatural if it is prolonged for too long using hard-constraints. The output quality is further improved by using audio matching to find as many examples of the selected source sound as possible. This is because more transitions from non-source sounds to the source sound are made available to the algorithm, making it possible to get smoother transitions when a target area is encountered. The DVD-ROM examples in the *More Manual Synthesis Examples* section exhibit a variety of constraint circumstances such as one or more hard constraints, soft constraints and constraint overlaps.

4.1.5 Applicability

Existing sound models, especially physically-based ones, do not support sound textures [Saint-Arnaud and Popat 1997]. Our sound synthesis techniques, on the other hand, are particularly well-suited for these sound types. In the concept of a sound texture, we include both the stochastic nature of sound over a long stretch of time, and the perceptual similarity of any bit of that sound. For example, the ambient noise in a cocktail party is always changing in a very complex pattern, but it is globally perceived as the sound of a cocktail party. Other examples are fan noise, traffic noise, or even the sound of a (possibly foreign) language. Sound textures still comprise a very wide variety of sounds. They are different from timbres, which are *one-shot* sounds. Sound textures can be made up of timbres, but the higher-level organisation of the timbres should obey a random distribution, rather than having a musical structure.

The method does not work in every case. However, there is no clear-cut universal rule defining which sounds will or will not work. This is more of a case-by-case issue, also faced by image texture synthesis algorithms. All the same, simple heuristics can help users anticipate the appropriateness of a sound with regard to its potential re-synthesis quality. Hence, good candidate input sources for our methods are sounds that allow themselves to be temporally 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. In particular, the algorithms perform especially well on stochastic, or non-pitched, sounds. These sounds are defined as any non-pitched sound having a characteristically identifiable structure. For example, the sound of rain has a characteristic structure that makes it easily identifiable as rain and easily distinguishable from random noise. Humans almost continuously hear stochastic sounds such as wind, rain or motor sounds. Because of their prevalence in real-world environments and consequently video and animation, it is important that these sounds are well-supported in our system. This is especially valuable since existing controllable sound synthesis techniques tend to have trouble dealing with stochastic sounds.

Sounds with clear progressions, such as a slowly increasing siren sound, cannot be meaningfully re-arranged by hand and therefore, neither by our algorithm. Note that some sounds could be manually re-arranged by a user but might still fail our re-synthesis. This is to be expected due to the lack of semantic understanding by our algorithm of the nature of the original sound. Similarly, our methods are 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. In some cases, we found that constrained, versus

random, sound synthesis increases the set of re-synthesizable sounds. By careful definition of the synthesis constraints, we can ensure that certain sound configurations are never encountered. For example, we can forbid a *plane free-falling* sound, using exclusion constraints, after a *plane crashing* sound constraint.

The DVD-ROM examples in all four interaction modes show our synthesis methods working on a wide variety of sounds including cheering, laughing, baseball practice, industrial machines, clapping, motor sounds and nature sounds.

4.1.6 Computational Efficiency

Figure 53, Figure 54 and Figure 55 respectively show the generation times (on a 1.6GHz processor) for the wavelet approach, self-similarity approach and natural grain approach. The goal is to synthesize output soundtracks of increasing lengths for 12 second, 48 second and 96 second of 44Khz input sounds.

For the grain-based approaches, the analysis phase corresponds to the time taken to segment the sound into grains and calculating the transition probabilities. For the wavelet approach, this is the time to build the source wavelet tree. The natural-grain approach is the fastest since only neighbouring frame distances are extracted. This performance is closely followed by the wavelet approach. The self-similarity based approach is noticeably slower to segment the sounds (even on small sounds) since extracting the self-similarity matrix is $O(n^2)$ where n is the number of audio frames. This can be done in an off-line batch manner if necessary.

The generation phase corresponds to the time taken to synthesize the new sound once the analysis phase is completed. The wavelet approach is the fastest, only taking a fraction of the final duration of the synthesized soundtrack. Both grain-based approaches have similar, slower performance, taking slightly less time than the duration of the synthesized soundtrack. In all approaches, the impact of the number of constraints was minimal. The examples here use three synthesis constraints each with three separate sources and two target segments. When increased to 13 constraints, a 10% decrease in synthesis performance was encountered.

A change in the segmentation threshold for the grain-based approach only necessitates a recalculation of the grain transition probabilities. Although the operation is $O(n^2)$, where n is the number of grains, the costs are insignificant for a few dozen grains such as in our example. However, for longer soundtracks (i.e. over 10 minutes) with thousands of grains, this would be more costly. The transitions cannot be efficiently pre-calculated since the grains are not known until the segmentation threshold is given.

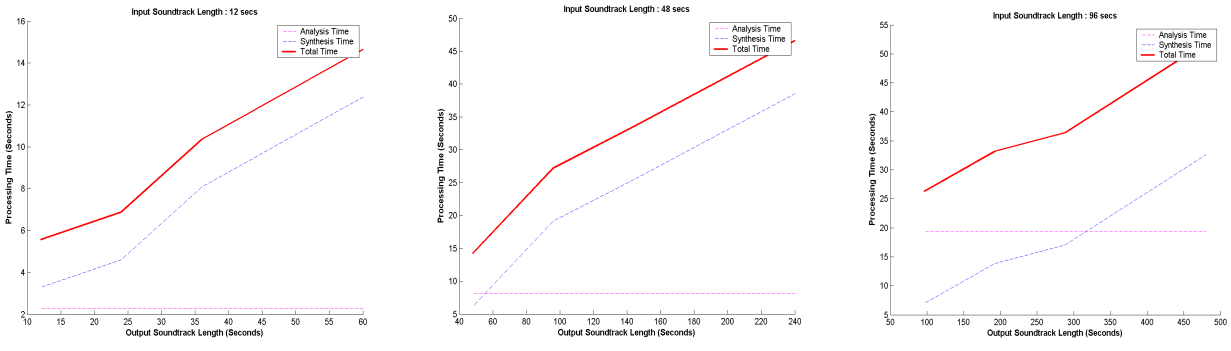

Figure 53: **Wavelet-based synthesis generation times**. (*Left*) 12 second input sound. (*Middle*) 48 second input sound. (*Right*) 96 second input sound.

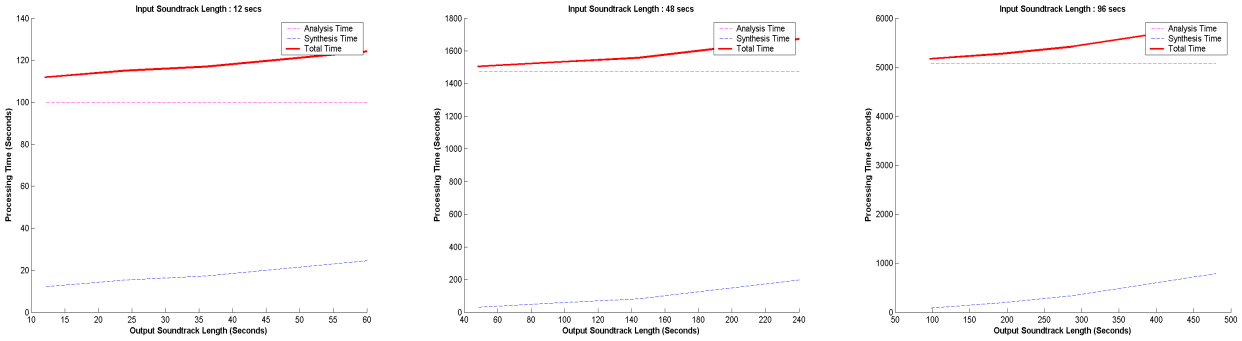

Figure 54: **Self-Similarity-based synthesis generation times**. (*Left*) 12 second input sound. (*Middle*) 48 second input sound. (*Right*) 96 second input sound.

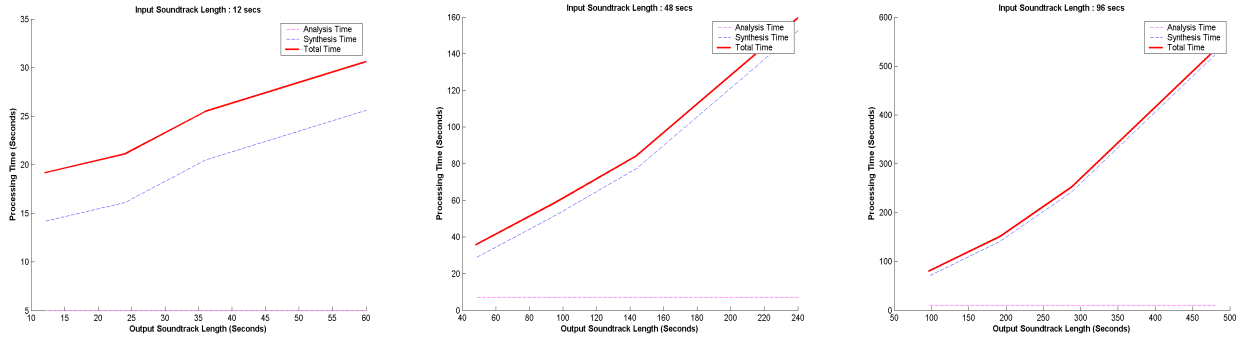

Figure 55: **Natural grain based synthesis generation times**. (*Left*) 12 second input sound. (*Middle*) 48 second input sound. (*Right*) 96 second input sound.

4.1.7 Interaction Mode Examples

- **Manual Control**

An example of soundtrack creation for video is given in the *System Overview* video. Using the manual control interface, a battle soundtrack is retargeted from its original video sequence to a new edited sequence. The creation process is depicted and only takes a few simple manipulations. In the *Machine 2* example, the clicking and popping sounds contained in the original soundtrack are completely remove in the synthesized one using hard exclusion constraints. The rest of the examples show an assortment of different manual constraint combinations to achieve very different effects.

- **Semi-Automatic Control**

Semi-automatic control is used to produce the soundtrack of a flying bird animation in the *System Overview* video. All flight patterns similar to the user-selected ones are assigned to the same target sounds, whilst the rest of the soundtrack conveys the jungle background sounds. The *Ballet Walk* example shows semi-automatic control over a motion capture sequence so that the sound associated to the right-leg stepping motion is assigned to all other similar instances in the rest of the sequence. Our high-dimensional mocap matching techniques were used to find the matches.

- **Fully-Automated Control**

The *Car* example takes advantage of the fully automated approach to generate new racing car sounds to accompany the edited version of the original racing car animation (see Figure 56). We do not have to limit ourselves to a single motion constraining the synthesis of a single soundtrack. As this example shows, we can combine multiple soundtracks from multiple objects at once. Two examples are given demonstrating the automatic generation of soundtracks synchronized to a mocap shooting sequence and a standing-up sequence in respectively, the *Cowboy* and *Standing-up* examples. In the *Cowboy* example, the RMS volume is used to automatically give more importance to the louder shooting sounds in the synthesis, since keeping the synchronicity between the shooting sound and its associate motion yields better results.

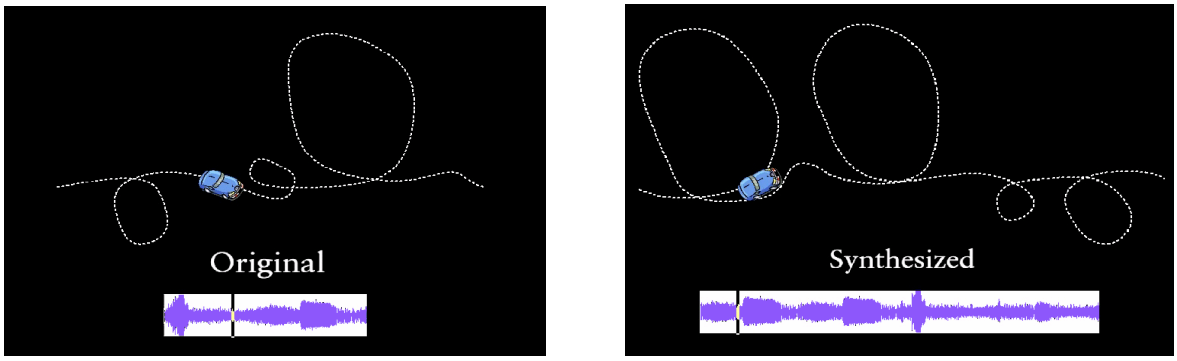

Figure 56: **Fully-automated Sound Synthesis.** Given a source animation of a car and its associated motor soundtrack (*Left*), an unseen target car animation of the same nature (*Right*) is analyzed to synthesize a new motor soundtrack automatically, with a high probability of having appropriate sounds for similar motion events.

- **Sound transfer**

Our sound transfer technique is used to rebuild a motor-boat soundtrack from jeep engine sounds in the *Boat 1* example. The RMS volume is used as the matching feature. In the *Jeep* example, voice drives the synthesis. A voice recording of the user imitating the type of jeep engine sounds targeted at any given point in the new soundtrack is made. The recording is used as the constraining soundtrack in the synthesis. This is a rapid way of generating controlled sound textures using a convenient interface. Less straightforward mapping is also possible such as replacing the jeep engine sounds by dog growling sounds in the *Dog* example, or replacing bird sounds with jeep engine sounds.

4.1.8 Conclusion

Sound textures are a relatively new area of research compared to the maturing field of image texture synthesis. Still, after over ten years of research, there has yet to be a single definitive algorithm for synthesizing new images. Each model has drawbacks and limited applicability. The same is true for sound texture synthesis. The wavelet model's performance in terms of audio quality is on the whole inferior to the grain-based approaches, but it is the fastest. However, out of our two grain-based models, our examples show that there is no clear-cut winner. It is difficult to quantify which approach is the best since each model generates slightly different output for each synthesis run. As also noted in [van den Doel 2001], there are no objective perceptual measures to help evaluate and compare the quality of our results. Additionally, output quality is heavily dependent on the source sound, the targeted constraints, the selected thresholds and the utilized model. Since the constraint specification phase is the same for all three approaches, users can generate multiple candidate soundtracks from all three models with the same constraints. The best result is then selected.

Chapter 5

CONCLUSION

5.1 Summary of Achievements

This work set out to simplify the production of sound for computer animation and related areas. Although only a subset of the overall issues involved in production are addressed in this work, the suggested automation techniques are nonetheless valuable as shown in our discussions.

5.1.1 Contributions to Sound Production

The primary contribution of our sound editing work is the introduction of the concept of controllable sound synthesis-by-example. The proposed methods address an important, yet untouched issue in audio effects engineering for film, television, games, animation and VR. There are therefore no techniques to which it can be directly compared. The methods allow the creation of sound generation models which are learnt from source examples, circumventing the difficulty of creating proper physical or phenomenological models of sound generation. These techniques do not apply to all types of sounds, such as speech or music, and are better suited to model sound textures such as background sounds and sound effects.

The other major contribution consists of the different ways of setting the constraints for soundtrack generation. Contributions from regions of the source signal can be set either manually, or by semi-automatically segmenting source and target animation and using motion matching algorithms to make source-to-target correspondences. We describe a novel operation of associating sounds with motion parameters of an animation by example. Finally, sound transfer (i.e. audio-driven sound synthesis) opens up a completely new way of defining and creating sound. The goal was to make our methods simple enough to be used in consumer-level software and powerful enough to be used by audio professionals.

Most of these contributions have already been published; the publications cover the wavelet approach [Cardle et al. 2003a], the natural grain approach [Cardle et al. 2003c] and the self-similarity approach [Cardle et al. 2003b].

5.2 Future Work

Sound and motion production branches into many interesting sub-problems. In this section, we identify possible avenues for future research extending the capabilities of our system.

5.2.1 Sound Synthesis

With regards to our sound synthesis work, there are still many opportunities for future work, a few of which we list below.

Using Video to Direct the Sound Synthesis. The idea here is to extract features from the video and use these to control the synthesis. This would allow constraint definition schemes similar to the semi and fully-automatic ones we developed for animation data. For example, motion of objects in the video could be extracted using computer vision-based motion tracking. These trajectories would then be used to find the sound correspondences from one video to another. Other features could be used to guide the matching such as inter-frame differences characterizing the change in video, colour palette variations, global panning motions, scene complexity, video segmentation and similarity measures [Foote et al. 2002].

Motion and Video Synthesis Integration. Our system could be combined with recent motion capture and video synthesis methods, such as presented in [Akira and Forsyth 2002] and [Schödl et al. 2000], to synthesize motions or video sequences with meaningful soundtracks. There is currently no work in multi-modal approaches to synthesizing both visuals and sound in parallel. Given an example animation or video and its associated soundtrack, we synthesize a new visual sequence along with the appropriate sounds.

Sound Morphing. Currently our algorithm deals with overlapping constraint regions by giving the source sound with the highest associated weighting a higher probability of being chosen. Hence, we hear one sound or the other but not both at the same time. Perhaps the sound morphing techniques described in [Fitz et al. 2002] could be used to construct an intermediate sound from each sound's weightings. More simply, we perform a mix with the percentage of both sounds being proportional to the weighting of each region.

Audio Filtering. In this work, we try to preserve the qualities of the original audio as much as possible. However, it is possible to generate more varied sound textures by applying audio filters such as time scaling, amplitude setting and pitch shifting as in [Lu et al. 2002; Miner and Caudell 1997]. These could be randomly controlled by the synthesis and smoothly introduced to ensure continuity.

Extending Recent Sound Texture Models. It would be interesting to extend the recent work in sound texture synthesis [Athineos and Ellis 2003; Lu et al. 2003; Recht and Whitman 2003; Schwarz 2003; Parker and Chan 2003] to support our presented synthesis control. Experimentation would reveal whether such approaches can produce better quality results, or work on a larger variety of input sounds.

Quasi-silent Segment Removal. Identifying quasi-silent portions [Tadamura and Nakamae 1998] in the source and giving them lower priority during constrained synthesis would improve constraints satisfaction and therefore control.

ACKNOWLEDGEMENTS

This work was supported by the Engineering and Physical Sciences Research Council. I would also like to thank Stephen Brooks, Carsten Moenning, Malcom Sabin, Mark Grundland, Neil Dodgson, Andras Belokosztolszki, Ziv Bar-Joseph, Loic Barthe, Stephen Rymill, Stephen Hainsworth, Simon Dixon, Dinesh Pai and George Tzanetakis for their valuable help and ideas.

BIBLIOGRAPHY

Arikan O., and Forsyth D.A. (2002). Interactive Motion Generation From Examples. *In Proceedings of ACM SIGGRAPH 2002*, San Antonio, Texas, August 22-27.

Arikan O., Forsyth D.A and O'Brien J. (2003). Motion Synthesis from Annotations. *In Proceedings of ACM SIGGRAPH 2003*, San Diego, California, August.

Ashikhmin M. (2001) Synthesizing Natural Textures. *ACM Symposium on Interactive 3D Graphics*, pp. 217–226, 2001.

Athineos M. and Ellis D. (2003) Sound Texture Modelling with Linear Prediction in both time and frequency domains. *ICASSP 2003*.

Bar-Joseph Z., Lischinski D., Werman M., Dubnov S., and El-Yaniv R. (1999). Granular Synthesis of Sound Textures using Statistical Learning. *In Proceedings of the International Computer Music Conference*, 178-181.

Boyd A. (2002) Lord of The Rings: The Two Towers. Stormfront Studios Interview. *music4games.net*. 2002.

Brand M. and Hertzmann A. (2000) Style machines. *In The Proc. of ACM SIGGRAPH 2000*, pages 183--192, 2000.

Brooks S. and Dodgson N. (2002). Self-Similarity Based Texture Editing. *ACM SIGGRAPH 2002*.

Cardle M., Barthe L., Brooks S. and Robinson P. (2002a) Local Motion Transformations Guided by Music Analysis. *Eurographics UK 2002*, Leicester, June 2002.

Cardle M., Barthe L., Brooks S., Hassan M. and Robinson P. (2002b) Music-Driven Motion Editing. *ACM SIGGRAPH 2002* Sketches and Applications, San Antonio, July 2002.

Cardle M., Brooks S., Bar-Joseph Z. and Robinson P. (2003a) Sound-by-Numbers: Motion-Driven Sound Synthesis. *SIGGRAPH Symposium on Computer Animation*, San Diego, July 2003.

Cardle M., Brooks S. and Robinson P. (2003b) Directed Sound Synthesis with Natural Grains. *Cambridge Music Processing Colloquium 2003 (CMPC 2003)*, Cambridge University Engineering Department, March 2003.

Cardle M, Brooks S. and Robinson P. (2003c) Audio and User Directed Sound Synthesis. *International Computer Music Conference (ICMC 2003)*, Singapore, October 2003.

Cardle M., Vlachos M., Brooks S., Keogh E. and Gunopulos D. (2003d) Fast Motion Capture Matching with Replicated Motion Editing. *SIGGRAPH 2003* Sketches and Applications, San Diego, July 2003.

Carlsson S. (2002) Film Sound Design. *filmsound.org*. Visited in 2002.

Cosi P., De Poli G. and Prandoni P. (1994). Timbre characterization with mel-cepstrum and neural nets. *In: ICMC (1994)*, pages 42.45.

Costa M., Zhao L., Chi D., and Badler N. (2000) The EMOTE model for effort and shape. *In Proceedings of SIGGRAPH 2000*, 2000

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.

Efros A. and Leung T. (1999) Texture Synthesis by Non-parametric Sampling. *IEEE International Conference on Computer Vision (ICCV'99)*, Corfu, Greece, September 1999.

Fitz K., Haken L., Lefvert S., and O'Donnel M. (2002) Sound Morphing using Loris and the Reassigned Bandwdith-Enhanced Additive Sound Model: Practice and Applications, *Computer Music Conference*, April 2002.

Foote J. (1997). Content-based retrieval of music and audio. *In: Conference on Multimedia Storage and Archiving Systems II, Proceedings of SPIE*, pages 138.147.

Foote J. and Cooper M. (2001). Visualizing Musical Structure and Rhythm via Self-Similarity. *Proc. International Conference on Computer Music (ICMC 2001)*, Habana, Cuba, September 2001.

Foote J., Cooper M. and Girgensohn A, (2002) Creating Music Videos using Automatic Media Analysis. *In Proc. ACM Multimedia 2002*, pp. 553-60, December 2002, Juan-les-Pins, France.

Funkhouser T., Jot J-M and Tsingos N. (2002) Sounds Good To Me!, Computational Sound for Graphics, Virtual Reality, and Interactive Systems *SIGGRAPH 2002 Course Notes*. 2002.

Gold B. and Morgan N. (2000). Speech and Audio Signal Processing: Processing and Perception of Speech and Music. John Wiley & Sons.

Hahn James, 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.

Hahn J., Hesham F., Gritz L., and Lee J.W. (1995). Integrating Sounds in Virtual Environments, *Presence Journal*.

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.

Home Theater Buyer's Guide Magazine, (1998) David Lynch Interview. Fall 1998.

Hoskinson R., and Pai D. (2001). Manipulation and Resynthesis with Natural Grains. *In Proceedings of the International Computer Music Conference 2001*.

Itakura F. (1975). Minimum prediction residual principle applied to speech recognition. *IEEE Trans. Acoustics, Speech, and Signal Proc.*, Vol. ASSP-23, pp. 52-72

Keller D., and Truax B. (1998). Ecologically-based granular synthesis. *Proceedings of the International Computer Music Conference*, Ann Arbor, IL: ICMA.

Keogh E. and Pazzani M. (1999). Relevance feedback retrieval of time series data. *In Proceedings of the 22th Annual International ACM-SIGIR Conference on Research and Development in Information Retrieva*l. pp 183-190.

Keogh E. and Pazzani M. (2000a). A simple dimensionality reduction technique for fast similarity search in large time series databases. *In Proceedings of Pacific- Asia Conf. on Knowledge Discovery and Data Mining*, pp 122-133.

Keogh E., and Pazzani M. (2000b). Scaling up dynamic time warping for data mining applications. *In 6th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. Boston.

Kovar L., Gleicher M., and Pighin F. (2002) Motion graphs. *In Proceedings of ACM SIGGRAPH 2002, Annual Conference Series. ACM SIGGRAPH*, July 2002. (a)

Kovar L., Schreiner J., and Gleicher M. (2002) Footskate cleanup for motion capture editing. *In ACM Symposium on Computer Animation 2002*. (b)

Lasseter J. (1987) Principles of Traditional Animation Applied to 3D Computer Graphics, *SIGGRAPH'87*, 1987

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.

Litwinowicz P. (1991) Inkwell: A 2 1/2-d animation system. *SIGGRAPH 1991 Proceedings*, 25:pages 113–122, July 1991.

Lu L.., Li S., Liu W.Y. and Zhang H.J. (2002). Audio Textures. *Proc. of IEEE International Conference on Acoustics, Speech and Signal Processing 2002* (ICASSP02).

Lu L., Mao Y., Wenyin L. and Zhang H-J. (2003). Audio Restoration by Constrained Audio Texture Synthesis. *Proc. of IEEE International Conference on Acoustics, Speech and Signal Processing 2003* (ICASSP03).

Maitland F. (2001) Sound: The Risk Factor. May 2001. Gignews.com

Menache A. (2000) Understanding Motion Capture for Computer Animation and Video Games. 2000. Morgan Kaufman.

Meredith M. and Maddock S. (2000). Motion Capture File Formats Explained. University of Sheffield. *Technical Report.*

Miner N. and Caudell T. (1997) Using Wavelets to Synthesize Stochastic-based Sounds for Immersive Virtual Environments. *Int. Conf. on Auditory Display 1997*.

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.

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*.

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.

O'Donnell Marty. (2002) Producing Audio for Halo. *Game Developer Conference* 2002.

Parker J.R. and Chan S. (2003) Sound Synthesis for the Web, Games and Virtual Reality. *SIGGRAPH 2003* Sketches and Applications, San Diego, July 2003.

Pavlovi'c V. I., Berry G. A., and Huang T. S. (1997) Fusion of audio and visual information for use in human--computer interaction. *In Proceedings of Perceptual User Interface, PUI'97*, pp 68--70, Banff, Alberta, Canada, Oct. 1997

Peck. (2001) Beyond the Library: Applying Film Post-Production Techniques to Game Sound Design – *Game Developer Conference* 2001 Lecture.

Perlin K. (1995) Real-time Responsive Animation with Personality. *In IEEE Transactions on Visualization and Computer Graphics*, 1(1), March 1995, pp.5-15.

Polichroniadis T. (2000) High Level Control of Virtual Actors. University of Cambridge. *Ph.D Thesis.*

Popovic Z. and Witkin A. (1999). Physically based motion transformation. *In Proceedings of ACM SIGGRAPH 99*, pages 11–20.

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.

Rabiner L. and Juang B. (1993). Fundamentals of speech recognition. Englewood Cliffs, N.J, Prentice Hall.

Recht B. and Whitman B. (2003) Musically Expressive Sound Textures from Generalized Audio. *Proc. of the 6th Int. Conference on Digital Audio Effects (DAFx-03)*, London, UK, September 8-11, 2003.

Roads C. (1988). Introduction to granular synthesis, *Computer Music Journal,* 12(2):11–13.

Robertson B. Bad to the Bone. (1999) *Computer Graphics World.*, Vol. 22, No. 5, p34 May 99.

Rose C., Cohen M., and Bodenheimer B. (1998) Verbs and adverbs: Multidimensional motion interpolation. *IEEE Computer Graphics and Application*, 18(5):32–40, 1998.

Saint-Arnaud N. and Popat K. (1997) Analysis and Synthesis of Sound Textures. in *Computational Auditory Scene Analysis*, D.F. Rosenthal and H.G. Okuno, Eds., pp. 293–308. LEA, 1997.

Schwartz D. (2003) The Caterpillar system for data-driven concatenate sound synthesis. *Proc. of the 6th Int. Conference on Digital Audio Effects (DAFx-03)*, London, UK, September 8-11, 2003.

Schödl A., Szeliski R., Salesin and D., Essa I. (2000). Video textures. *In Proceedings of SIGGRAPH 2000*, pages 489-498, July.

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.*

Tadamura K., and Nakamae E. (1998). Synchronizing Computer Graphics Animation and Audio, *IEEE Multimedia*, October-December, Vol. 5, No. 4.

Tak S, Song H-Y. and Seok Ko H. (2000). Motion Balance Filtering. *EUROGRAPHICS 2000,* Computer Graphics Forum, Vol. 19, No. 3.

Takala T., and Hahn J. (1992). Sound rendering. *In Proceedings of SIGGRAPH 92,* Computer Graphics Proceedings, Annual Conference Series, 211-220.

Truax B. (1988) Real-time granular synthesis with a digital signal processor. *Computer Music Journal*, 12(2), 14-26

Unuma M., Anjyo K. and Takeuchi R. (1995) Fourier Principles for Emotion-based Human Figure Animation. *SIGGRAPH '95,* 1995.

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.

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.*

Waibel A. and Lee K.-F., editors. (1990) Readings in Speech Recognition. Morgan Kaufmann Publishers, San Mateo, CA, 1990.

Witkin A. and Popovic. Z. (1995) Motion warping. *In SIGGRAPH 95 Proceedings, Annual Conference Series*, pages 105--108. ACM SIGGRAPH

Zaza T. (1991) Audio design: Sound Recording Techniques for Film and Video. PrenticeHall, 1991.

Zils A. and Pachet F. (2001) Musical mosaicing. *In Proc. Of G-6 Conference Digital Audio Effects DAFX-01*, Limerick, Ireland, 2001.

APPENDIX B

GRANULAR SYNTHESIS

Sound is ordinarily expressed in terms of its wave-like properties or as changes in air pressure over time. It can alternatively be expressed in terms of particles or grains just as, for example, the descriptions of light differ between its wave and particle properties. Dennis Gabor proposed this granular theory of sound over 50 years ago. Gabor believed that any sound could be synthesized with the correct combination of numerous simple sonic grains. The French composer Iannis Xenakis is commonly cited as one of the pioneers of granular synthesis. The first computer-based granular synthesis system did not appear, however, until Curtis Roads [Roads 1988] and Barry Truax [Truax 1988] began to investigate the potential of the technique systematically.

Grains of sound are short bursts of sonic energy that combine to form larger sounds. The duration of a grain or granule might be of the order of 1ms to 100ms. Grains not only describe existing sounds at an atomic level but can also be recombined to form new sounds. This is the basis of granular synthesis where the construction of sounds is done from a collection of very short segments of sound. Most systems have used probabilities to control the production of the granules; for example, to control the waveform and the duration of the individual grains. Granular synthesis techniques vary, but all are built on the same particle description of sound in a two-stage approach [Keller and Truax 1998]. First, we establish a time-frequency grid of grains, then we produce the sound by placing either synthesized grains (e.g., sine waves, FM synthesis or filter parameters) or sampled-sound grains (from one or several sound files).

Hoskinson and Pai [2001] and Lu et al. [2002] use more traditional lines of granular synthesis. Their analysis phase uses perceptual measures of sound to determine the appropriate grain segmentation. Hoskinson and Pai [2001] find points of least audio change, whilst Lu et al. [2002] use a measure of self-similarity to find troughs of the signal between events. In both cases, grain duration is relatively large, although the coarseness is adjustable.

The BJ algorithm [Bar-Joseph et al. 1999] is slightly different in that a multi-resolution representation of grains is utilized. The swapping of these multi-resolution grains (or more specifically wavelet coefficients)

ranges between very fine-grained to coarse-grained synthesis. Also, no implicit model of sound is assumed in the analysis phase so, in theory, a larger set of sound types should be supported.