

**The “*mkmusic*” System -  
Automated Soundtrack Generation  
for Computer Animations and Virtual Environments**

By

Suneil Mishra

B. Sc. (Hons.) in Computer Science and Mathematics,  
University of Glasgow, Scotland. June, 1992.

A Masters thesis submitted to

The Faculty of

The School of Engineering and Applied Science  
of the George Washington University in partial satisfaction  
of the requirements for the degree of Master of Science.

May 5, 1999.

Thesis directed by

Dr. James K. Hahn

Associate Professor of Engineering and Applied Science

## ABSTRACT

The “*mkmusic*” System -  
Automated Soundtrack Generation  
for Computer Animations and Virtual Environments

by Suneil Mishra

Directed by Associate Professor James K.Hahn.

Traditional techniques of soundtrack production for computer animations are based on methods used in film and traditional animation. These methods fail to exploit the potential for automation of the creative process using computers. In particular, the data supplied to create the visuals of a computer animation are currently ignored or underutilized in soundtrack production. The lack of a suitable soundtrack tool for computer animators has been a major cause of this neglect in the past. Described here is the *mkmusic* system which seeks to bring sound effects and musical accompaniments within the reach of animators. Its creative, compositional process involves filtering the data generated by an animation’s motion control system and mapping those values to aural constructs. A simple user interface and animator-programmable filters allow flexible control over the nature of the sound effects or music produced. Rendering can be in the form of common aural scores (e.g. CSound, MIDI), or as real-time performance of composed pieces. Using this methodology, the system has created aesthetically pleasing, appropriate music and sound effects for a variety of animated sequences, with minimal animator expense.

## Table of Contents

<b>CHAPTER ONE: INTRODUCTION.....</b>	<b>1</b>
<b>CHAPTER TWO: EXISTING SOUNDTRACK PRODUCTION TECHNIQUES.....</b>	<b>4</b>
2.1 FILM PRODUCTION.....	4
2.2 TRADITIONAL ANIMATION TECHNIQUES .....	6
2.3 COMPUTER MUSIC SYSTEMS .....	7
2.4 ANIMATION AND VIRTUAL ENVIRONMENT SYSTEMS .....	8
2.5 SONIFICATION RESEARCH.....	11
2.6 DESIRED FEATURES OF A SOUNDTRACK SYSTEM .....	12
<b>CHAPTER THREE: INTRA-STREAM SOUND PROCESSING .....</b>	<b>14</b>
3.1 OVERVIEW OF INTRA-STREAM SOUND PROCESSING.....	14
3.2 SYNCHRONIZATION AND TIMING .....	16
3.2.1 <i>Dynamic Modification and Local Control</i> .....	16
3.2.2 <i>Realism</i> .....	18
3.3 DATA RELIANCE: UNDERLYING ISSUES.....	19
3.3.1 <i>Uniqueness of Sound-Causing Events</i> .....	19
3.3.2 DYNAMIC PARAMETERIZATION.....	19
3.3.3 <i>Realism</i> .....	19
3.4 MOTION-TO-SOUND BINDING.....	20
3.4.1 <i>Motion Parameters</i> .....	20
3.4.2 <i>Sound Parameters</i> .....	21
3.4.3 <i>Defining the Mapping</i> .....	23
3.5 SOUND AND MUSIC ISSUES .....	24
3.5.1 <i>Commonalities between Sound and Music</i> .....	24
3.5.2 <i>Issues Specific to Sound</i> .....	25
3.5.3 <i>Issues Specific to Music</i> .....	25
<b>CHAPTER FOUR: INTER-STREAM SOUND PROCESSING.....</b>	<b>27</b>
4.1 OVERVIEW OF INTER-STREAM SOUND PROCESSING .....	27
4.2 APPLYING GLOBAL SOUND EFFECTS .....	29
4.2.1 <i>Realistic Effects Design</i> .....	29
4.2.2 <i>Emphatic effects design</i> .....	31
4.3 MUSICAL CONSTRAINTS .....	32
4.3.1 <i>Basic Musical Rules</i> .....	32
4.3.2 <i>Compositional Effects</i> .....	33
4.3.3 <i>Level of Compositional Control</i> .....	34
4.4 COMBINING REALISTIC AND MUSICAL SOUND STREAMS .....	34
4.4.1 <i>Applying Global/Environmental Effects</i> .....	34
4.4.2 <i>Over-constraining sound streams</i> .....	35
<b>CHAPTER FIVE: RENDERING ISSUES.....</b>	<b>36</b>
5.1 RENDERER REQUIREMENTS AND DESIRED FEATURES.....	36
5.2 SUPPORTED RENDERERS .....	37
5.2.1 <i>CSound</i> .....	37
5.2.2 <i>MIDI</i> .....	40
5.2.3 <i>Other supported renderers</i> .....	42
5.3 REAL-TIME RENDERING CAPABILITY AND VES.....	44
<b>CHAPTER SIX: USING MKMUSIC: RESULTS AND EXAMPLES.....</b>	<b>49</b>
6.1 GUIDELINES FOR USE.....	49
6.1.1 <i>Data-File Format</i> .....	49
6.1.2 <i>Command-line controls</i> .....	51
6.2 ANIMATION EXAMPLES .....	58

6.3 SONIFICATION PROCESSING .....	61
<b>CHAPTER SEVEN: FUTURE WORK AND CONCLUSIONS.....</b>	<b>64</b>
7.1 FUTURE WORK .....	64
7.2 CONCLUSIONS.....	65
7.3 ACKNOWLEDGEMENTS .....	66
<b>REFERENCES.....</b>	<b>68</b>
<b>APPENDIX A: MKMUSIC COMMAND-LINE OPTIONS.....</b>	<b>75</b>
<b>APPENDIX B: CUSTOM DATA-MAP EXAMPLES.....</b>	<b>77</b>
<b>APPENDIX C: MUSICAL SCORE EXAMPLES.....</b>	<b>79</b>

### Table of Figures

<b>FIGURE 2.6.1: MKMUSIC SOUNDTRACK PRODUCTION PIPELINE. ....</b>	<b>13</b>
<b>FIGURE 3.1.1: FILTERING SUPPLIED DATA INTO STREAMS. ....</b>	<b>15</b>
<b>FIGURE 3.1.2: DISCONTINUOUS DATA AND TRIGGER-FILTERS.....</b>	<b>15</b>
<b>FIGURE 3.2.1: TIMING PLOTS FOR COLLIDING OBJECTS IN AN ANIMATION. ....</b>	<b>17</b>
<b>FIGURE 3.2.2: EVENT-PLOTS PER OBJECT FOR COLLISION ANIMATION. ....</b>	<b>18</b>
<b>FIGURE 3.4.2A: FRAMES FROM WINDCHIMES ANIMATION. ....</b>	<b>24</b>
<b>FIGURE 3.4.2B: FILTER CODE EXTRACT FOR WINDCHIMES ANIMATION. ....</b>	<b>24</b>
<b>FIGURE 3.4.2C: MUSICAL SCORE GENERATED FOR WINDCHIMES ANIMATION.....</b>	<b>24</b>
<b>FIGURE 4.1.1: SCORE EXTRACT FROM AN ANIMATION OF WALTZING FIGURES.....</b>	<b>28</b>
<b>FIGURE 4.3.1A: EMOTION TO SCALE MAPPING. ....</b>	<b>32</b>
<b>FIGURE 4.3.1B: EMOTION TO NOTE LENGTH MAPPING. ....</b>	<b>33</b>
<b>FIGURE 5.2.1: CSOUND ORCHESTRA FILE FOR A GUITAR INSTRUMENT. ....</b>	<b>38</b>
<b>FIGURE 5.2.2: CSOUND SCOREFILE EXAMPLE FOR 2-PART, SINGLE INSTRUMENT PERFORMANCE. ....</b>	<b>39</b>
<b>FIGURE 5.2.3: TEXT-FORMAT SCORE EXTRACT. ....</b>	<b>43</b>
<b>FIGURE 5.2.4: SREND FORMAT SCORE. ....</b>	<b>44</b>
<b>FIGURE 5.3..1: CSOUND SCORE EXTRACT. ....</b>	<b>46</b>
<b>FIGURE 5.3.2: VE COMPOSED SCORE.....</b>	<b>47</b>
<b>FIGURE 6.1.1: DATA-FILE EXTRACT WITH SIX PARAMETERS. ....</b>	<b>49</b>
<b>FIGURE 6.1.2: DATA-FILE SHOWING TIME-STAMPED COLLISION EVENTS. ....</b>	<b>50</b>
<b>FIGURE 6.1: PROMPTED INTERFACE FOR MKMUSIC. ....</b>	<b>51</b>
<b>FIGURE 6.1.4: SAMPLE COMMAND-LINE EXECUTION OF MKMUSIC. ....</b>	<b>53</b>
<b>FIGURE 6.1.5A: CUSTOM HEADER FILE FOR FILTER DESIGN PROCESS.....</b>	<b>55</b>
<b>FIGURE 6.1.5B: CUSTOM SOURCE FILE FOR FILTER DESIGN PROCESS. ....</b>	<b>56</b>
<b>FIGURE 6.1.6: TRIGGER-FILTER EXAMPLE FOR COLLISION-EVENT SOUNDS. ....</b>	<b>57</b>
<b>FIGURE 6.2.1: DATA-FILE FOR LAMP ANIMATION.....</b>	<b>58</b>
<b>FIGURE 6.2.2: DATA-FILTER FOR LAMP ANIMATION. ....</b>	<b>59</b>
<b>FIGURE 6.2.3: DATA-FILTER APPLIED TO TEXTURE INFORMATION FOR CLOUD OBJECTS. ....</b>	<b>60</b>
<b>FIGURE 6.3.1: DATA-FILE EXTRACT FOR MORPHING ANIMATION. ....</b>	<b>61</b>
<b>FIGURE 6.3.2: SCORE EXTRACT FOR 3-D MORPHING SONIFICATION. ....</b>	<b>62</b>

## **Chapter One: Introduction**

The development of soundtrack production techniques for computer animations has come from a variety of related fields. Most influential is that of film-soundtrack synchronization. In some ways, computer animation and multimedia in general appear to be following the development of film. Silent movies were for a long time the mainstay of cinema, as visual development was paramount in this new medium. In 1926, Warner Brothers introduced the first “*talkie*” short: “Don Juan,” followed a year later by the first full feature with sound: “The Jazz Singer.” Within three years, not one of the major Hollywood studios was producing *any* silent films. Silent movies seemed outdated and lacking in sophistication, and were destined to extinction soon after [Walk79]. Similarly, the next generation of film technology saw colour first become a novelty and then an expected element visually. With audio, the early technologies saw little change until the more recent introduction of Dolby, DTS, SDDS and THX theatre surround-sound. In the same way, in the infancy of computer animation and multimedia, visuals have taken precedence, with sound an addition rather than a requirement. This is somewhat curious, as the technologies to create computer animation images are still developing, but the methods to create soundtracks have been until now based on the now well established film-soundtrack techniques.

A major reason for this apparent developmental lag in audio over visuals is that the creators of animations and many multimedia products are predominantly from a visual arts/graphics background. Their skills and concentration are on image production, and the aural realm is beyond their expertise. This is compounded by a second problem: given the limitations of animators within the aural domain, no suitably designed tool exists to allow animators to create appropriate soundtracks. A further reason is multimedia designs often have several media *competing* in simultaneous presentation. rather than mutually reinforcing each other. In such cases, the user’s attention has been found to be selectively engaged by visual stimuli in preference to aural effects [Barg93].

With this background it is understandable that most animators have difficulty in developing soundtracks to their animations independently. Even if the motivation existed, the traditional methods to create soundtracks are often mysterious and require time to learn and master. With research projects or production deadlines, it is difficult to justify the expense of adding sound to an animation “in-house.” The alternative is to hire outside professionals to supply a soundtrack satisfying certain requirements of the animator. This has its own inherent expense, and an associated loss of animator control over the exact nature of the soundtrack created. An additional problem is that modifications to an animation’s visuals or motions usually force the entire soundtrack to be regenerated, with all the added time and expense that entails.

The lack of overall aural expertise amongst animators is a difficult one to resolve. Cross-disciplinary teaching and training of both visual arts and audio/music may help to alleviate some of the fear and unfamiliarity of the aural medium from animators (as well as bringing sound researchers and producers closer to the visual arts in the same way). The emergence of multimedia as a research field is beginning to lay the foundations for this emphasis on the combining of sight and sound. However, without the appropriate tools, animators - regardless of their competence and familiarity with the aural medium - will be unable to produce quality soundtracks appropriate to their animations.

The *mkmusic* system described here aims to address this problem. Specifically, the system attempts to provide a means for animators to create appropriate musical accompaniments, and simple sound-effects, to animations without the need for outside sound professionals, or highly specialized equipment [Mish95]. The system uses data supplied from an animation motion-control system to compose both musical orchestrations, and to synchronize sound-effects to motion-events. This data-reliance emphasizes the motion-to-sound mapping, dynamically binding synchronization-to-motion as a concurrent processing step. In doing so, the

appropriateness of the soundtrack to the specific animation is enhanced. Although automation in this way does reduce animator control over the production, this is also a problem with existing techniques, as discussed above. Using the *mkmusic* system, the interactivity of the composition process allows the animator at least some higher-level directorial control.

The goals of the *mkmusic* system are two-fold; first, it aims to be appropriately designed for the use of computer animators and be a flexible and simple development tool for soundtrack production. Second, and more importantly, the soundtracks produced must be of a standard that the system is practical to use by animators. The soundtracks composed should be aesthetically pleasing and appropriate to the animation visuals, with minimal animator effort. In this way, the system can fulfill its ultimate goal of bringing soundtrack production directly to computer animators.

## **Chapter Two: Existing Soundtrack Production Techniques**

### **2.1 Film Production**

The majority of soundtracks for animations have been created using traditional techniques, and synchronized using methods similar to those in film and video production [Zaza91], [Frat79]. Generally, sound-effects and musical scoring are considered independent until final editing of the film is done. This separation was applied early in the development of soundtracks for film (by 1929) [Altm85].

Musical scoring [Karl94] is often done as a somewhat concurrent process with visual production: a musical director (in charge of scoring the film) is provided with a screenplay, and generally shown rough-cuts of some shot scenes. The director of the film usually discusses his/her ideas for how the soundtrack should develop, with the musical director. The musical director then proceeds to compose a main theme for the film, and other incidental music, often character-thematically (recognizable pieces that reoccur when particular characters are on-screen). When visual-editing is complete, the musical score is laid to match the visual action. The musical themes and incidentals are added as per the director and musical director's wishes, and in accordance to the editing [Lust80].

Meanwhile, on-location sound is recorded along with the visuals using standard techniques with microphones and recording media such as analogue tape, or linear film. On completion of the filming, at the editing stage, additional sound-effects and dialogue can be recorded. Foley is the standard way of post-producing sound-effects, while ADR (Automated Dialogue Replacement) or looping techniques are used for dialogue. Foley is the creation of 'faked' sound effects on a sound-stage after visuals have been completed. By visual inspection, Foley-artists determine the timing of motion events, and create sounds to match with the actions. Sound-stages are replete with props and sound-spaces to simulate common sounds. This



Foley process is highly specialized, and good Foley is considered highly demanding. The soundtrack synchronization is post-processed as visual editing often destroys the continuity of sound-to-visuals as recorded at the time of filming. A pre-processing method was also sometimes used in film, but this was usually a necessity of recorded dialogue, which the development of looping made unnecessary. Looping involves splicing segments of the film together and repeatedly looping through with actors providing voice-overs on original dialogue which has to be changed. This manual splicing process has been overtaken by ADR in which a computer is used to control this replacement timing process by marking start and end points, and allowing up to three dialogue tracks to be stored for later selection. Most current sound production techniques have changed little over the past seventy years; few technological advances in computer-assistance have been made until the recent development of digital recording and editing, with ADR being the main computer-aided addition to post-production [Came80].

The combining of sound-effects and musical accompaniments is done in a final mix-down, where the relative amplitudes of dialogue, sound-effects, and music are determined, based on the importance of each at particular times, and on what the most effective complement to the visuals will be. This relationship between the visual and aural elements has been described in narrative terms as “mutual implication” [Gorb87]. This traditional stance is often bucked by avant-garde cinema, with overlaid narration of a visual storyline [Orr93], or the use of ideational sound [Mint85]. Such decisions are actively controlled by director, music director, and chief sound-editor, producing a final soundtrack for the edited visuals. The development process of both sound effects and musical scores is a dynamic one. The creation of sounds or music may at any time be preempted by changes made in the continuity of the visuals through editing. Thus, both effects and music must be somewhat flexible and modifiable in nature.

## 2.2 Traditional Animation Techniques

The techniques described above are also predominant in animation soundtrack production. The use of pre-processing of soundtracks is more widespread in animations however. This is illustrated by the classic Disney animated features which use a prerecorded score, and timing-sheets for the soundtrack in order to create animation frames synchronized to the visuals. For computer animations, the generation of images, or frames, is an automated process. The timing-control is dependent on the motion-control scheme used. With dynamic simulations for instance, it may be impossible for the animator to create motions specifically synchronized to a pre-generated soundtrack. As computer animators often want to demonstrate some novel research concept, they usually want to retain control of motions within the visual domain. Therefore, pre-processing is a severely limiting technique.

With the post-processing method, sounds are synchronized to existing visual events. Thus, the motions of objects are determined before any sounds are generated. The sound synchronization is often done manually through Foley, with visual inspection being augmented with a timing-sheet with information on key action events. As keyframing is used as the motion-control technique with traditional cel-type animation, the timing-sheet for the animation may in fact be the key-frame specification as used by the key-animators and in-betweeners. A deficiency of this approach, as with its film complement, is that a change to visuals usually means complete post-processing of the soundtrack again. This can be time-consuming and expensive, especially when the soundtrack has been contracted out to a sound-production house. Traditional animation techniques give a foundation to the concept of synchronization and timing with a frame-by-frame animation production. They also serve to illustrate the limitations of the film-paradigm; with computer animation, the amount of visual data available to aid synchronization and sound descriptions is far superior, but until now, under-utilized.

### 2.3 Computer Music Systems

The main computer-audio research until recently has been restricted to the field of computer music. Systems developed here are meant specifically for musical applications, with no inherent design methodologies to include a binding to the visual medium, or to objects moving through a virtual space. What these systems do bring is the concept of parameterization of sound structures, and a hierarchy of control over aural (specifically musical) constructs. Several computer music systems have been instrumental in the development of sound renderers in general. The *Music-N* family of computer music systems was initially developed at AT&T labs. Many systems have descended from it. *Music-V* [Matt69] in particular has been a seminal work in computer music. This synthesis system is a score language: it represents music as a simple sequence of notes. There is no concept of hierarchical structuring in *Music-V*. The contemporary view of music representation is of multiple hierarchies [Dann93], [Wigg93]. Such hierarchies are also apparent in the motion world, so representations supporting these structures are beneficial. The CSound system [Verc86] is a widely used aural renderer, with versions on many operating systems (Mac UNIX, PC) beyond its original NeXT version. This system allows *instruments* to be written as parameterizable structures, using C-language type calls. The parameters can be time-varying, allowing dynamic changes in the output sounds. These instruments are passed parameters by means of a *score* which binds values to instruments using a scripted, time-stamped format. This system is almost ideal in its structure and generality. Two main drawbacks remain however. First, the processing overhead of synthesizing the sounds from instruments is high. This makes true multi-timbral, real-time performance difficult. Second, the parameterizations allowable still remain within the aural/musical field, and binding visual/virtual parameters to the musical instrument ones remains problematical. Other similar computer music systems include Platypus [Scal89], CMusic: an extension to CSound, and Fugue [Dann91] which uses functional expressions as a basis for its sound synthesis.

## 2.4 Animation and Virtual Environment Systems

Systems grounded in computer animation have traditionally involved pre- or post-processing, and in general do not dynamically bind visual and aural media. There are three main exceptions to this: Wayne Lytle's: "More Bells and Whistles" animation [Lyt91] demonstrated a mapping from sound to motion parameters. This is the reverse of how animators normally create, and what most would prefer: namely a generation of images with a corresponding soundtrack. The Background Music Generator (BGM) [Naka93] allows synchronization to visuals of a musical soundtrack composed from a database of short melodies. In addition, some simple sample-based sound effects can be rendered. A major shortcoming of this system is that its use of visual data is limited. The synchronization is scene-based and for the most part manually specified by the animator. The music produced also is not linked to the motions involved in any way, other than a temporal synchronization over the length of a specified scene. The control mechanism does use a similar high-level emotional context to the *mkmusic* system described below. Finally, the Sound-Renderer [Taka92], developed at the George Washington University, attempts to explicitly bind motion control values to parameters in sound structures. This dynamic linking allows synchronization and modifiable sounds to be generated. The original system uses a sound structure known as *timbre-trees*, similar to shade-trees [Cook84] in image rendering. Developments within the system have aimed to bridge the gap between animation-parameter schemes and aural-parameter schemes, as the Sound Renderer designs sounds specifically with animations in mind [Hahn95a]. One major drawback to the system is that in synthesizing sounds using timbre-trees, the processing overhead is too great to allow real-time performance unless a parallel implementation is utilized. Even with such an implementation, multi-timbrality remains a problem. However, an extension of the system to VEs (known as VAS) can real-time synthesize a simple sound in a VE using timbre-trees given enough processing power [Hahn95b], so this seems a promising system. The main features that the Sound-Renderer/VAS systems deliver is their linkage to visuals and object motions, and their attempt to control soundtracks from a visual perspective. In particular, specialized audio knowledge is not required of the user.

The system has been limited by the difficulty in creating suitable parameterizable sound structures (timbre-trees), and has thus far been restricted to realistic sound effects, rather than music. The *mkmusic* system is an extension of the Sound-Renderer paradigm, and supports the timbre-tree binding format.

The latter two systems seek to solve many of the problems faced by animators in creating soundtracks, but have been unable to achieve all the goals of a widely utilized system. The BGM suffers from a deficient binding between visuals and sound; the Sound-Renderer has so far been limited to realistic sound-effects, and has great processing overhead. What neither of these systems provides is true real-time performance at the level necessary for multipurpose, flexible usage in a virtual environment. (Though the SoundRenderer/VAS system is close to achieving this.) Rendering capabilities should allow for multiple concurrent sounds, with independent controls, and the merging of both realistic and musical sounds. Additionally, in a virtual environment simulation, no concept of scripting should exist if the environment is truly interactive. This leads to binding problems, and interpolation problems since the future parameter states are unpredictable. The VAS system addresses this problem, but in doing so has been forced to deviate from the goals of an interactive tool for animators.

Several aural simulators exist for use in virtual environments. In general, they can be grouped according to their particular research objectives; these are localization systems and feedback-and-navigation systems. Each of these systems does attempt to provide real-time performance of auralization, since this is a perceptual requirement of realistic-seeming sounds.

Localization systems fall into two sub-categories: loudspeaker-based, and headphone-based techniques. Each attempts to simulate sounds as being positioned in 3-space using channel-loudness balancing and phase information, either through loudspeakers, or stereo-headphones. With headphones, the most successful method to date involves Head Related

Transfer Functions (HRTFs) [Blau83], which simulate listener effects due to head, body, and pinnae differences as well as inter-aural intensity and time delays of sounds through space to the listener's two ears. The best known, and most important work in this area was done at the NASA-Ames Research facility, leading to the development of the *Convolvotron* hardware-DSP which was capable of localizing up to four independent sounds in an environment through headphones. Many papers have been written by the developers, Beth Wenzel [Wenz88], [Wenz90], [Wenz91], [Wenz92], and Scott Foster, of Crystal River Engineering [Fost88], [Fost91]. Other systems include Pope's DIVE Auralizer [Pope93], Gehring's FocalPoint [Gehr90], and the U.S. Air Force's cockpit localizer developed at Wright-Patterson AFB, Dayton, Ohio [Doll86].

Feedback and navigation systems tend to often be extensions of user interface work in the 2-D desktop metaphor. The primary papers in this field are by Gaver, [Gave93], and Blattner [Blat89], on auditory icons. Many other designs are feedback systems for complex-task environment information systems, including [Furn86], [Calh87], [Gave91], and [Patt82]. These are really specialized versions of sonification projects since they intend to auralize information in some way.

The major limitation of applying such work to animations is there has long been a reliance in VE sound development only on the localization problem. Navigation systems have most often been limited in their sound generation, as they usually attempt just to provide simple temporal feedback, such as generic warning-sounds (e.g. sirens, beeps) when a particular event is triggered. For example, a user is too close to a virtual obstacle, or a measured quantity surpasses a set safety-level. What these systems do provide is a template for real-time sound production, with the possibility of added perceptual cues such as localization, which increase the effectiveness of the soundtrack.

## 2.5 Sonification Research

Sonification is the aural analogue of visualization. The mapping between data and sound is the consideration here. Such work again stresses the importance of appropriately parameterizable sound structures. These systems are generally not designed to work interactively in real-time, although the advent of the virtual environment paradigm makes this a logical next step in development. Examples of sonification work are Smith's exploratory data analysis [Smit90], and the Chua project which auralizes chaotic Chua Circuits [Maye92]. Other sonification research includes Kyma [Scal92], [Scal93], Kramer's "Sonification Toolkit" [Kram91], [Kram94], and the sonic maps work by Evans [Evan89].

Sonification is typically bound with visualization, so the overall data displayed is increased. The sonic display may mirror the visual data, in which case the aural medium reinforces the visualized data by the observer, increasing the impact and reliability of the information reception [Bly85]. Alternatively, the sonification may be of data separate from the visualized set. In this case the total number of displayed parameters is increased.

Much of the work done in the field of acoustics theory involves sonification in simulating acoustical phenomena. For example, [Bori85] and [Roma87] give accurate computer simulations of acoustic environments such as auditoria. These simulations are often more rigorous than necessary for the more figurative nature of animation soundtracks, but are certainly relevant for virtual environments, where environmental effects of the virtual space, and realistic modeling are often important.

The production of soundtracks for animations can be considered simply a specialized subset of sonification in general. We often attach sounds to visual events (such as collisions), but also add effects and music to provide information not visually perceptible (such as wind-sounds or "sad" music to set the mood of a scene). What sonification research provides is a generalized

framework for binding data to sonic constructs. However, the selection and effectiveness of the bindings is the crucial element, and this is application specific to animations, or motion-control in general. We can examine the underlying issues from the standpoint of sonification, but we must find solutions appropriate to our animation requirements in particular.

## **2.6 Desired Features of a Soundtrack System**

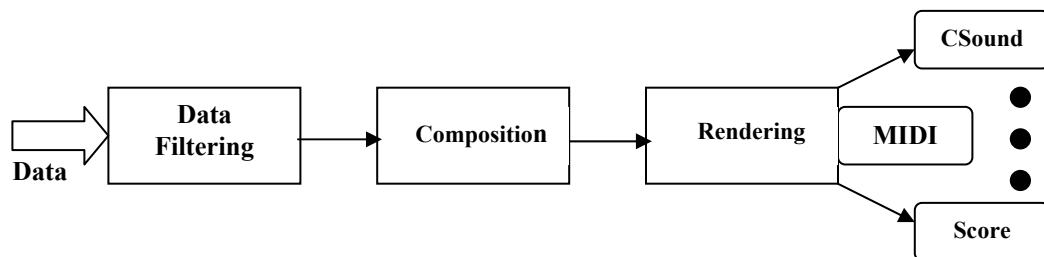
From the above research and production fields, the technologies and methodologies for a variety of soundtrack development techniques have been established. We can identify the desirable features of an animation soundtrack system both from the advantageous elements, and from the drawbacks and deficiencies of the previous work. What is clear is that no genuinely effective system yet exists, as there is no automated system in widespread animation usage.

The *mkmusic* system attempts to include the features identified here of a practical soundtrack system. A simple, intuitive interface for animators is important; controls and constructs must be recognizable to the user, and technical aural constructs should be avoided or abstracted from the animator. Synchronization between motions and sounds must be established and maintained in order for the system to be useful. This synchronization must be dynamic, so that changes to motions bring corresponding adjustments to the soundtrack. Moreover, these adjustments should require minimal additional effort on the part of the animator, provided said changes are not wholesale. In particular, the measure of this effort should be related to the effort involved in the changes to the motion. Slight tweaking of motions should result in correspondingly minor adjustments being required to the soundtrack. A more general requirement of the system is that it produce high quality, appropriate soundtracks. In order to do this, the parameterizable sound constructs and hierarchical score descriptions of computer music systems is preferred. These sound constructs should be bound to motion parameters as supplied by the motion-control system of the visual simulation. Such bindings should go beyond mere timing, as in the BGM, but to actual sound descriptors, as in the Sound-Rendering systems.



While speed is not essential to soundtrack production, it is clearly desirable; close to real-time performance enhances interactivity, and allows refinements and modifications to be made with less effort. If real-time performance can be maintained, the system can be extended to use within virtual environments. In order to be an effective tool however, the functionality of animation and VE systems should be transparent to users. A common interface-design philosophy should therefore be used. Other minor desirable features include the ability to produce soundtracks in a variety of formats in addition to aural playback. This allows easier storage and modification of the soundtrack, as well as making it accessible to sound/music experts if commonly used formats are supported.

Of the features identified, a few are essential while most are simply desirable in order to make production easier, or of higher quality. The two factors most important are that the system should provide links to the animation/motion world, and links to the sound world. The *mkmusic* system attempts to achieve this by application of two types of processing: *intra-stream* processing which attaches visual-context to data from a motion-controller, and *inter-stream* processing which attaches aural-context to the data streams supplied. These processes are applied by the *data-filtering* and *composition* modules of the system, respectively. The result of this context-association is a set of auditory control streams; these are passed to the *rendering* module for output as supported in a variety of formats. **Figure 2.6.1** shows the *mkmusic* production pipeline, from data-input to render-formatted output.



**Figure 2.6.1:** *Mkmusic* soundtrack production pipeline.

## Chapter Three: Intra-Stream Sound Processing

### 3.1 Overview of Intra-Stream Sound Processing

The creation of soundtracks by the *mkmusic* system is based on data-sonification. This data is accepted temporally, and may be continuous or discrete in nature. In either case, this data is considered to consist of meaningful motion control parameter values. Each data parameter will be contained within an associated data *stream*. Which stream a parameter belongs in, what that stream itself represents, and whether it is used in the composition is determined by the intra-stream sound processor, or *data filtering* module.

Intra-stream processing applies to filtering of the data, with each input-stream considered a separate entity. This process provides *application-context* to the input data. Input streams are actually defined by this application context, so a single input-stream may actually consist of more than one input parameter. An example of this would be a stream representing an object's position in a virtual space. The xyz-coordinates supplied as three separate values would be considered conceptually as a single 'position stream.' Exactly how input data-values are grouped as streams is controlled by the data-filters applied. Thus, the data-filtering process may be considered the stream-specification process. The filtering is non-destructive in nature, allowing streams output from the data-filtering process to consist of overlapping data components.

**Figure 3.1.1** shows filtering of input data to produce data-streams with application context. Data supplied may be directly passed-out (**D<sub>1</sub>->S<sub>1</sub>**), blocked (**D<sub>5</sub>**), filtered once (**D<sub>2</sub>->S<sub>2</sub>**, **D<sub>4</sub>->S<sub>4</sub>**), or multiple times (**D<sub>2</sub>->S<sub>3</sub>**, **D<sub>3</sub>->S<sub>3</sub>**, **D<sub>4</sub>->S<sub>3</sub>**).

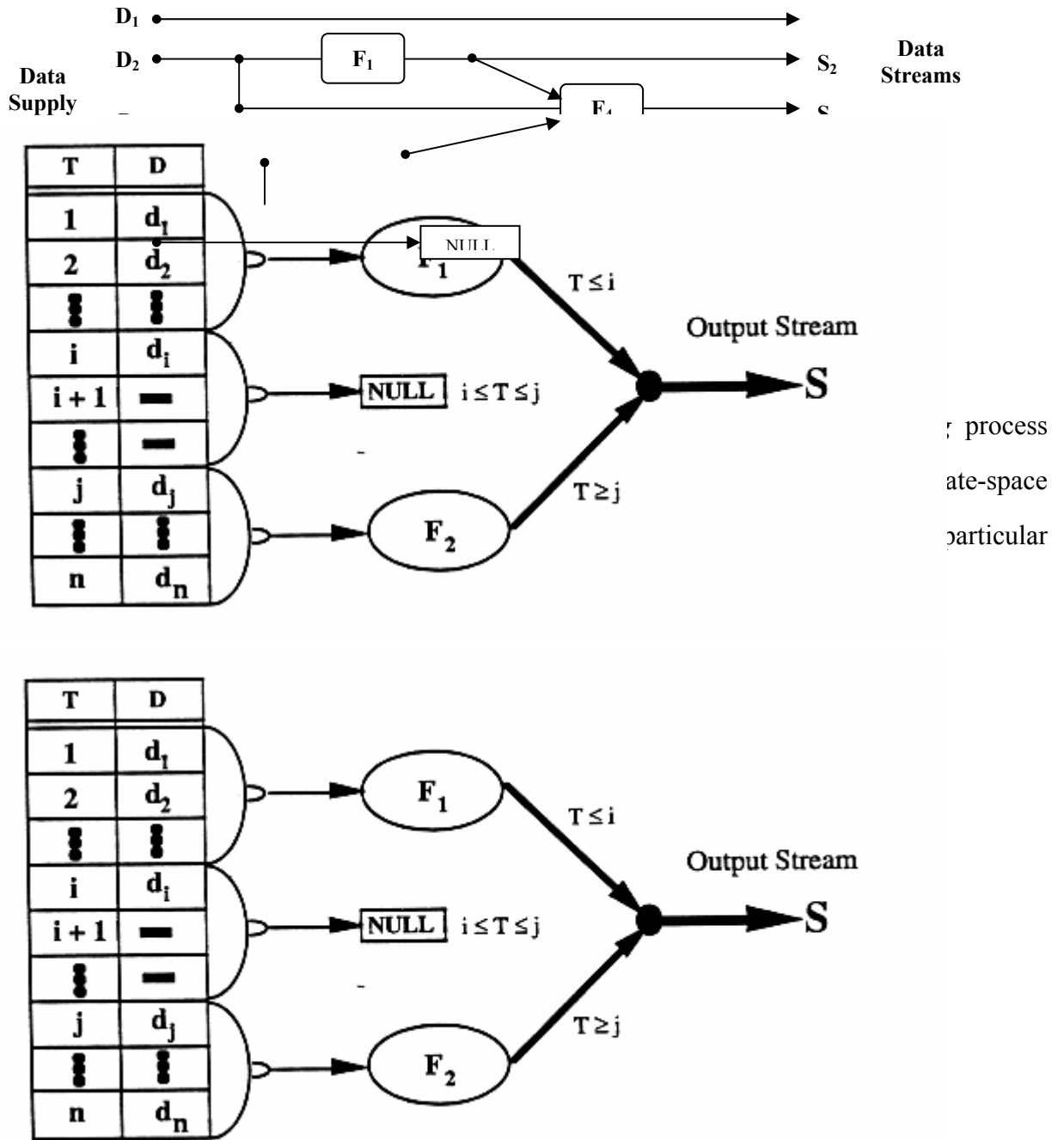


Figure 3.1.2: Discontinuous data and trigger-filters.

The data input for a particular parameter  $D$ , is supplied from an initial time-index  $1$ , continuously until a later time  $i$ . It then ceases to supply data until time  $j$ . Data is then again supplied continuously until time  $n$ . Over the time-span of the animation ( $1..n$ ), the data-supply is thus discontinuous. The filtering operation is applied when the data is present, and the stream is  $NULL$ -defined in-between. The remaining system processing will handle this  $NULL$  stream by

ignoring its aural contribution. In addition to handling of discontinuous data, **Figure 3.1.2** shows we can also filter the input data differently based on a triggering mechanism. In this case, we apply filter  $F_1$  initially, and then trigger filter  $F_2$  when the time-index =  $j$ . The triggering mechanism could equally well be based on any recognizable control event, as specified by the user in the filter-design process.

This filtering process provides meaning to the input data. In order to effectively do this, a knowledge of the parameters the data represents in the motion world is necessary. It is expected that the filtering-design is user-specified and customized, based on the nature of the application and data provided.

What the intra-stream filtering provides is synchronization to visuals, *and animation-context*. Later we will see how we must also apply *auditory-context* to produce an effective soundtrack. The importance of the synchronization and context, and how they relate to designing filters is now discussed.

## **3.2 Synchronization and Timing**

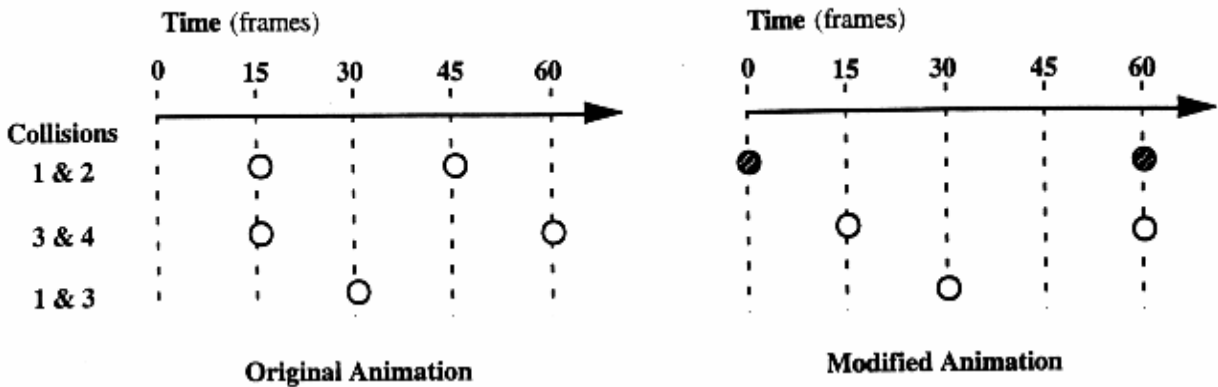
### **3.2.1 Dynamic Modification and Local Control**

The multiple-synchronization problem is further exacerbated by modifications<sup>1</sup> made to an animation. With a synchronization based on visual inspection (Foley), these changes cause the entire synchronization process to be undertaken again. In contrast, using motion control data, modifications mean only a numerical difference in the data supplied.

---

<sup>1</sup> By modifications we refer to slight adjustments, or “tweaking” of given motions. Wholesale changes would most likely require the redesign of data-filters, just as they would involve a complete Foley restart.

Once the synchronization pipeline has been established, any modification to motions can be dynamically handled. Additionally, since the data supplied will remain the same for unmodified sections of the animation, local control over the soundtrack is maintained. This applies not only temporally (change at time  $A$  does not affect soundtrack at time  $B$ ), but elementally also (change to object  $I$  at time  $A$  does not affect object  $2$  at time  $A$ ). **Figure 3.2.1** demonstrates both these instances.

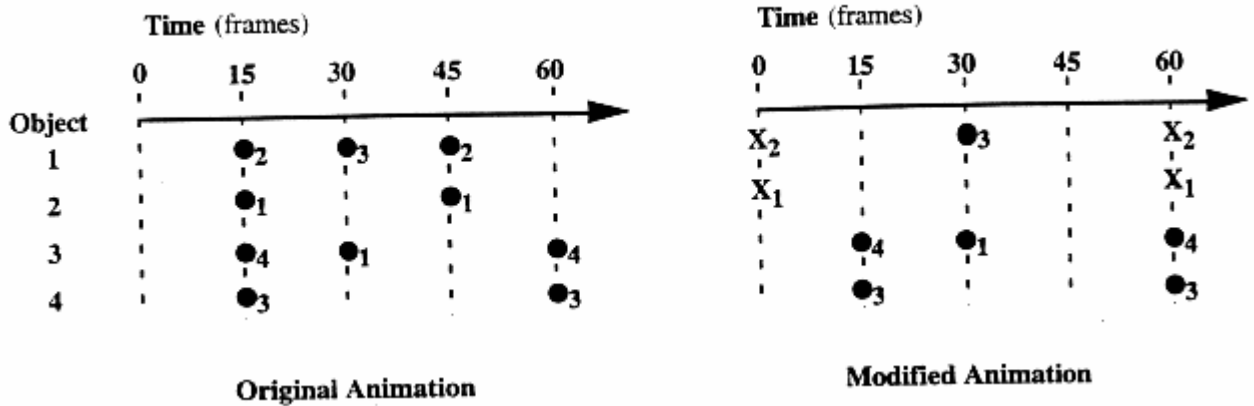


**Figure 3.2.1:** *Timing plots for colliding objects in an animation.*

In the original animation, we have four objects involved in several collisions. We modify the collisions between objects  $1$  and  $2$  but leave all other collisions as in the original. The O's represent collision events (and their corresponding sounds). The ●'s represent all the events from the original that are affected by the modification. Using manual Foley on an audio track, any modification to any element of the animation would force a re-recording of the entire soundtrack, as the individual tracks for each object will be overlaid and cannot be individually extracted and replaced.

As **Figure 3.2.2** shows, the only differences to the data supplied would be to the collision events for objects  $1$  and  $2$ . Only the X-ed events would be reprocessed differently using the *mkmusic* system, constituting four of the ten events here, or 40% modification. With manual Foley, in the modified animation all the event plots would X's, representing complete re-processing (100% modification). Using *mkmusic*, the individual events maintain an

independence as they are described by data control streams instead of auditory signals. This independence between data streams is an important characteristic; it allows local refinement to be performed with minimal effort, using standard data manipulation techniques rather than audio



signal-processing. In the above example, the savings are 60% over manual Foley-ing with direct audio-track manipulation.

Figure 3.2.2: Event-plots per object for collision animation.

### 3.2.2 Realism

Synchronization is a major factor in the realism of an animation soundtrack and its associated visuals. When a misalignment is evident, the animation as a whole seems unnatural. Human pattern-recognition and perception creates expectations of the elements necessary to identify an event as ‘real.’ If these expectations are not met, the animation will be dismissed as unconvincing. In fact, misalignments may prove distracting and make an animation worse-off perceptually than the animation would be without a soundtrack at all. It is thus very important that an accurate synchronization between sound and visuals be established and maintained. The automation and dynamism of this process through *mkmusic* is therefore an attractive feature of the system.

### **3.3 Data Reliance: Underlying Issues**

#### **3.3.1 Uniqueness of Sound-Causing Events**

Sounds are caused by physical interaction between objects and the environment they reside in. The nature of the sound produced is based on the state of the environment at the time of the event causing the sound. Some of these state-parameters may be static (such as masses of objects, shapes of rigid-bodies, density of the environmental medium), while others may dynamically change through time. (Forces on objects, relative positions and orientations of objects, change in listener position.) Other factors may or may not affect the sound an object may make, although they are crucial to image-rendering. (For example, object colour, material and surface-texture characteristics.) In supplying data to the soundtrack system, the choice of which of the potential data parameters to output to *mkmusic* can be difficult to ascertain immediately. The design process to make this determination will be discussed in more detail below.

#### **3.3.2 Dynamic Parameterization**

Sound-causing events are uniquely defined by the state of the environment at the time; this environment is dynamic in nature in the motions and interactions of its constituent elements. It is desirable to base the sounds or music created for an animation on the relevant parameters of the environment (or its elements) for a specific sonic event. In particular, the sound should be described in some way using the updated current values of the chosen parameters at the time of the event. The data supply to the soundtrack system is inherently dynamic in nature. Thus we are providing the current state-space for the environment (or the relevant subset) continuously.

#### **3.3.3 Realism**

In addition to synchronization, realism of a soundtrack is dependent on the quality of the sound produced and its resemblance to the expected sound for a given sonic-event. As sounds are

physically caused by events in a dynamically changing environment, a creation process modeled on such an approach should be effective. While the underlying methodology of utilizing the state-space description in generating sounds is certainly valid, it must be stressed that a purely physically-based approach is inadequate for all soundtrack needs. The reasons for this will be described in the discussion of the motion-to-sound binding given below.

Sounds created with regard to the motions causing them are more intimately bound to their visual analogues, and more easily identifiable and separable by listeners as a result. This in turn leads to increased perceptual confidence and the sense of realism, or effectiveness of the soundtrack. This is a finding that is valid for any sonification with temporally-varying data [Kram91]. With musical accompaniments, the quality directly affects effectiveness also, but the goal of a musical composition is not acousto-realism but rather an appropriate aesthetic flow with the visuals.

There are three contributory factors to producing an effective binding between motion and sound: the parameters available in the motion world, the parameters available in the sound world, and the mapping(s) between them.

### **3.4 Motion-to-Sound Binding**

#### **3.4.1 Motion Parameters**

Parameters available from the motion-control system depend greatly on the sophistication and abstraction available in the motion-control schema used. Kinematics techniques (e.g. key-framing) will provide position and orientation data, and associated values such as joint angles in articulated figures. In physically-based simulations, higher level information, such as the impulse response to collision events, may be supplied directly. Behavioral techniques and user control (such as gesture or locomotion in VEs) may provide a further level of information abstraction.



Typically, the *mkmusic* system is supplied low level motion constructs such as positions or orientations of objects, or rotational values for joints of articulated figures. Higher order constructs such as velocities or forces may be calculated within the filtering process. Such additional, *secondary* processing is applied after the *primary*, stream-definition processing on the input data parameters. While these high-level parameters are usually not provided, and must be computed internal to the system, it is completely the animator's decision which data to select, and how much to provide.

In addition to these types of motion parameters, attributes of objects can be provided to attach characteristics to the sound. For instance, the length of an object, or its colour may distinguish it from its neighbors. When similarly constructed objects move with closely matched motions, some variant feature is desirable to identify the contribution of each to the aural track on output. Such *tertiary* features can also be included for attachment at the intra-stream processing stage.

Several issues factor into the selection of data to be supplied to the system from the motion-control program (or other external source). The data supplied should be representative of the main focus of the visuals (or more particularly of the desired main focus of the soundtrack); for instance, in a scene with dancers in a ballroom, the music should represent the dancers. The exact nature of the data (e.g. joint angles, position/orientation values, etc.) is a more difficult determination. Two strategies have been found to work well: first, parameters that are most indicative of the motion in the visual realm are often best to work with aurally too; second, "the more data the better." It is easier to select useful data and ignore data streams contributing little in the *mkmusic* system than to attempt to create subtleties that do not exist in inadequate data-sets.

### **3.4.2 Sound Parameters**

The second factor in deciding on appropriate bindings is in the available sound parameters in the range-set. The exact nature of the sound representation plays a major factor in this. With sampled sounds for instance, there is a strong limitation on the parameterization available within the sound. Effectively, only phase and amplitude changes are allowed. Additionally, effects such as attenuation and delay can be applied to the sample as a whole. This limits the sophistication of the mapping process. Timbre-trees are hierarchical structures, the nodes of which can represent constant values, variable parameters, mathematical functions, or sub-trees. This allows a highly flexible parameterization to be specified. These structures are rendered by synthesizing instantiations of a tree according to the current parameter state. The design problem of which synthesis parameters are most appropriate has been a drawback of this technique, as the tree must describe the entire sound synthesis process within its structure. Methods utilizing ‘external’ sound representations (such as MIDI) are limited by the specific protocol and scheme used by the device attached, and control-mechanisms available. With MIDI, a formal protocol exists which makes it easy to code for. While users should consider the renderer they intend to apply, it should be somewhat transparent to them what the limitations of a particular renderer are during the filtering or composition processes. Unfortunately, this coupling between modules is unavoidable as the resultant soundtrack is only as effective as the renderer.

In addition to the particular internal sound structure parameterizations that are possible, generalized constructs such as amplitude, pitch, delay, reverberation, and pitch-bend (Doppler shift), are all included. These low-level elements are utilized by higher abstractions which may be specified by the user. Such constraints include the musical composition constraints that are illustrated in the *mkmusic* system.

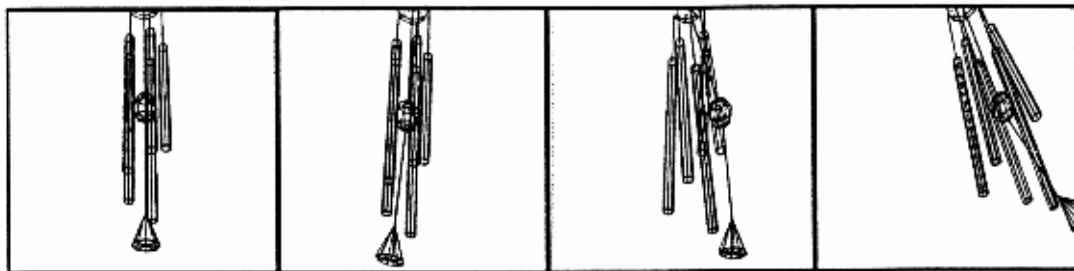
Whereas, in a purely physical simulation [Hahn88], the sound parameters mapped to would directly correspond with those in the real physical interaction, abstract bindings such as for musical score production clearly are not governed by the laws of the physical world.

Additionally, in emphasizing an action, animators - just as in the visual domain - often exaggerate motion response, or use sound effects to perceptually reinforce movements. Thus, a purely physically-based approach is impractical. In allowing a more flexible paradigm, the design space increases greatly. One approach in searching this space is the use of genetic techniques [Taka93], in which an evolutionary targeting of a desired sub-space is done. This method does not necessarily lead to a singular target solution, but rather guides the search to an appropriate area. The interactive, incremental approach we use also allows a search over this space, but control remains directly with the user. The genetic approach is an interesting one, however, and its integration would be part of any anticipated future work on the system.

### 3.4.3 Defining the Mapping

In physical simulations, the correspondence between motion and sound parameter spaces is provided by physical laws. In such cases, the mapping is straightforward. However, when acousto-realism is not the goal, the mapping is not so direct. Identifying the most effective or appropriate linkage is still more an art than a science currently, but common sense dictates that certain parameters are most appropriately bound to particular sound constructs.

A combination of heuristic selection, intuitive binding, and trial-and-error has succeeded in creating aesthetically pleasing soundtracks. Even a simple linear mapping can lead to complex and subtle results. The example below is of an animation of windchimes blowing due to a wind force. Each chime has a closely matched oscillating motion. Two filters are applied to create a soundtrack from their motions. The first ties musical note values to the distance of each chime from a static origin. The second filter is a weighting function based on the relative length of each chime. This weight is applied to the result of the distance filter, and a musical part is created for each chime. **Figure 3.4.2a** shows frames from the animation; **Figure 3.4.2b**, gives example coding for the filter functions and **Figure 3.4.2c** illustrates the five-part score generated.



**Figure 3.4.2a:** *Frames from WindChimes animation.*

```
int ChimeMap (float x, float y, float z, float length, float longest)
{
    float dist = sqrt (x * x + y * y + z * z);
    float weight = length / longest;
    return ((int) dist * weight);
}
```

**Figure 3.4.2b:** *Filter code extract for WindChimes animation.*



**Figure 3.4.2c:** *Musical score generated for WindChimes animation.*

### 3.5 Sound and Music Issues

#### 3.5.1 Commonalities between Sound and Music

In creating filters for realistic sound effects and for musical accompaniments, there are both commonalities and idiosyncrasies in the designs for each to address. First, as we are simply attaching application-context at this stage, the resultant filtered streams are mathematically similar. Each stream is just a flow of numerical values, representing the range-set for the mathematical functions defining the filter as applied to the supplied data domain. In designing the specific filters, composition of functions is used to create the filtering pipelines for the

corresponding input data parameters.

One especial concern is the continuity of streams. While both realistic and musical sounds may be discrete or continuous in nature, the mapping of discontinuous input to continuous flowing sound is fairly simplistic presently. This should be remembered when assigning streams for auditory output; in particular, when designing custom filters, the user should take into account the intended type of sound (realistic or musical) the output stream will map to. This aids the composition process in attaching constraints on the streams, as the current composition processor is limited in its sophistication.

### **3.5.2 Issues Specific to Sound**

With regard to data continuity, realistic sound effects tend to be more often discrete in nature. With this in mind, filters are more likely to be short-lived in their application to the sound streams. The use of trigger-filters is particularly prevalent in these cases, with discrete motion-events (such as collisions) having an associated filter; parameters of the space-state then describe the particular instance of the event (such as the force and direction of impact). The stream will then have the same filter applied, but will output identifiable filtered patterns based on the instancing parameters. By using such higher level, more generic filters, with triggers and instancing, we can reduce the total number of filters necessary to describe the soundtrack. Additionally, such abstractions into classes of sounds aid in ease-of-use, modularity, and reusability. Libraries of generalized filter-classes can be established, with modifications made to the class templates to produce event-specific streams.

### **3.5.3 Issues Specific to Music**

With musical accompaniments, output will tend to be continuous in nature. Staccato music may be warranted and can be highly effective in reflecting jerky, discontinuous motions.

However, since most motions will have more than  $C_0$  continuity, objects will visually move with smooth trajectories. Musical output streams appear from results so far to be more effective if continuous, time-varying parameter values are used. Objects with little relative changes to their associated data parameters tend to produce uninteresting music. As customized filters can be composed, the limitations of the data can be overcome with elaborate filtering, but this can detract from the inherent motion-to-sound binding as represented by the original data-set. Essentially, to get aesthetically pleasing, flowing music. Data should be filtered to moderate any excessive jumps or oscillations. The composition process bounds musical output in this way also, mitigating the necessity of making such constraints in the intra-stream processing unless the data is particularly unstable.

One general finding for musical score generation has been that at least three output parts are desirable if the music is to create and maintain an interesting sound. If fewer than three data streams are input, this can be accomplished by using multiple filters on each stream to produce  $N$  parts from  $M$  streams, where  $N > M$ . For instance, for a positional data stream, one part could output the direct position, a second could be based on computed velocities, while a third could be on accelerations. With many objects and their associated data parameter streams, the problem becomes one of reducing the data streams to a more controllable output orchestration. In such a case, we want  $N$  output parts from  $M$  input data streams, where  $N < M$ . Here, we can combine data-streams using filters, or aggregate multiple streams into single ones at any stage of the filtering process. Such secondary filtering is attached after the primary, stream-defining filters are applied to input parameters.

## **Chapter Four: Inter-Stream Sound Processing**

### **4.1 Overview of Inter-Stream Sound Processing**

At the data-filtering stage of the *mkmusic* system, though the input data streams may have been combined and filtered, the concurrent data-streams corresponding to the various *output* sounds have remained independent. Just as data-filtering regulates application-context (what each data parameter represents in the motion world), the *composition* process attempts to place auditory-context to the streams. It is likely that there is some relationship *between* the sounds, as they will generally at least share the same environment. Constructs such as attenuation and delay due to distance from the virtual listener position are attached here. Additionally, effects such as mixing can be attached to the streams, allowing purely auditory control with no linkage to the data supplier at all. At this stage, such constructs are merely encoded to allow for translation to the particular renderer format in the rendering stage. This process can be seen as packaging the aural data into understandable and coherent information, including the concept of timing, and high-level instrumentation.

In the musical composition process, the filtered data values are mapped to constructs including, but not restricted to, musical notes. These musical notes can be further constrained by simple musical rules that force the mood of the music to reflect emotional or stylistic requirements. A simple UI has been provided to allow control over the musical output, by means of high-level emotional and scale controls. These are based on well-established music theory; for instance the emotional control “sad” will constrain output to a “minor” musical scale. The exact nature of the notes and music composed will be determined from the data-stream passed through from the data-filtering module, however.

The score in **Figure 4.1.1** was composed from a scene of waltzing articulated figures within a ballroom. Here, it was appropriate that the music should reflect and match in tempo, the

dancers' motions. (This is technically the reverse of what would happen in reality, where the dancers would follow the music.) The figures move around the room, rotate full-circle about themselves once per second, and around the ballroom at a slower rate. Thus three output streams were generated. At this stage however, these streams have no musical constraints placed upon them to suggest the style of music being choreographed to. Since the animator wished a waltzing motion to be represented, the music therefore was to take on the stylistic constraints of that form



of music.

**Figure 4.1.1:** *Score extract from an animation of waltzing figures.*

These constraints (by no means of a complete or truly definitive nature) were that the music should be in 'three-time' as waltzes are composed in this way (three beats within a bar), and that the music be flowing but have an underlying rhythm. Additionally, the music was meant to be happy and lively in mood. These musical specifics were then applied by the following: the tempo was constrained by sampling the filtered streams at 3Hz. so that each bar in the above score represents one second of soundtrack time, and there are three notes per second. The repetitive rotation about the dancers' centers itself lender to the rhythmic tempo of the underlying bass part (the lowest score part in the example). The emotive constraints were supplied by setting the emotion and scale controls to "happy" and "major" respectively. The rotation stream around the room also provided some temporal binding, giving both flow and rhythm in parts. Finally, the positional mapping gave a sweeping flow to the music as the figures both visually and aurally glide around the ballroom, seemingly synchronizing their movements to the music. In addition to the purely musical constraints, attenuation of the music due to the distance of the dancers from the centre of the ballroom was applied as a further



compositional bound. Finally, an arbitrary amount of reverberation was added to suggest the echoic nature of the ballroom.

## **4.2 Applying Global Sound Effects**

### **4.2.1 Realistic Effects Design**

Realistic effects attached to the auditory streams tend to be environmental in nature. Effects such as attenuation and delay of sounds due to their distance from the listener position are two examples. Another would be the echoic quality of an enclosed space, represented by the amount of reverberation in the volume. Some concept of localization may also be required. These effects will eventually depend on the renderer chosen to auralize the auditory control streams, but the encoding of such effects is handled at this stage, abstracting the rendering process from the user.

Environmental effects have one factor in common: information on the environment and the objects' positions within it must be provided. This includes the listener's position if it is not considered central to the space and non-moving (the default). Descriptors may be supplied within the data also used to define the input streams, or may be provided by command-line controls. For instance, for attenuation, the auditory bounds of the space must be specified. This is input through the command-line. If bounding is required and this value is not provided, the *mkmusic* system has several built-in attenuation handlers. Bounding can be based on a static default, dynamic on a floating-range based on current values, or range-kicked to avoid linearly increasing or decreasing distances forcing boundary-edge constant values. Delay effects assume the transmitting medium to be of air-density. Other density values such as that of water, or fog can be easily provided to override the default. The delay values are based simply on an inverse-square of distance. Presently, occlusion by intermediate objects is not implemented though the

related sound-rendering work of the timbre-tree based systems would be a logical method of supporting this effect.

Reverberation is currently heuristic in nature, based on an overall determination of the volume of a space and an averaged linear distance bound between source, listener, and boundaries. As reverberation is just a product of delayed reflections from boundaries, a more rigorous model could be implemented; the heuristic approach appears sufficient however, given that soundtracks for animations and VEs tend to be more for reinforcement of visuals than precise simulations of the sonic space.

Localization eventually will be provided by a suitable renderer. Presently, as the available renderers to *mkmusic* have had limited localization potential. Only stereo-placement has been supported. With more sophisticated localization schemes becoming accessible, higher-order spatial placement will be a future enhancement.

These effects are considered global in the sense that they will apply to all the sounds in the environment. The effects are dynamically processed, so that changes to the environment are reflected in the appropriate realistic global effects. For instance, changes to the listener position will produce relative changes in the attenuation and delay for sound-producing objects. Updating of such effects on each sound currently is dependent on the sampling-rate of the data components of each auditory stream. This may mean a slight lag in the updating of some streams over others, but retains consistency with the data rather than having to update sounds based on estimated intermediate data values for the state-space of a stream.

All realistic effects attempt to produce a more natural perception of the soundtrack, further binding the visual and aural simulations. By using methods grounded in acoustics and real, physical interaction between objects in an environment, the system provides some of the

additional sensory cues that aid in human recognition of experienced sonic events. In order to effectively simulate a sound, it must appear to be as close to the listener's expectation of what is heard when the corresponding real world event causes the actual sound. The more of these cues available, and the more effective each individually is, the more likely the listener will believe the sound to be real.

#### **4.2.2 Emphatic effects design**

While realistic effects are meant to correspond accurately with listener's expectations of real events and their corresponding sounds, emphatic sounds are intended to reinforce events more abstractly. These sounds may be exaggerated beyond physical realism, in order to draw attention to particular events. For example, in traditional film, often fight scenes have blows with heavily over-emphasized impact sounds. This provides greater perceptual impact as to the force of the blows, adding a feeling of heightened action to the scene. Animations can similarly use such techniques to stress particular motions or events.

When exaggerating sounds for emphasis, we wish them to attach to sonic events without the realistic global effects described previously. The *mkmusic* system allows such sounds to bypass the realistic constraints. The emphasis we require could be attached at this composition stage similarly to the realistic effects, but with differing parameter limits. However, the system actually allows such control only to be applied at the intra-stream processing stage. Conceptually this is consistent, as the composition stage should only apply global effects to streams. Thus we only allow application of globals or complete bypass of those effects. While some auditory streams may share common emphatic features, they are considered independent in nature and must be applied internally to the sound streams.

### 4.3 Musical Constraints

#### 4.3.1 Basic Musical Rules

The use of music in animations and more particularly in film has been researched extensively. Music can provide many levels of context or information about the scene being observed. In [Burt94], George Burt describes the influence of musical accompaniments thusly:-

*“Music has the power to open the frame of reference of a story and to reveal its inner life...can deepen the effect of a scene or...have a telling effect on how characters in the story come across... Further, music being a temporal art...can have an enormous impact on the pacing of events.”*

Music is therefore an extremely forceful method of conveying mood or information on a scene, and can colour our attitudes towards the events observed. It is important that such power is accessible to animators in a regulated way so as to be most effective.

The *mkmusic* system composes musical accompaniments based on the data supplied from the motion-control system, and constrains the composition using simple musical rules. The controls available are based on high-level emotional labels, and scale and key-signature descriptors. This simplistic control is deliberate in that animators are assumed to have no musical knowledge.

Emotional labels apply directly to scale constraints. The link between emotion and scale is based on established musical knowledge of mood-transference [Senj87]. Sampling-rates may also be affected by the emotional-control mechanism.

Emotion	Scale
Happy	Major
Sad	Minor
Evil	No constraint

**Figure 4.3.1a:** *Emotion to Scale mapping.*

Emotion	Note Length
Bright	Short; staccato
Peaceful	Long; flowing

**Figure 4.3.1b:** *Emotion to Note Length mapping.*

**Figure 4.3.1** shows two of the constraint mappings as part of the musical composition process. In addition to scale control, note length (tempo) can play an effective role in changing the aesthetic colour of a soundtrack. For instance, “peaceful” music has more flowing, relatively longer, slower notes, whereas “bright” (or very peppy, cheery) music is more short, staccato, and jumpy in nature.

These simple controls are limited, but have resulted in complex, subtle musical compositions. Future work will attempt to develop the creative controls beyond these elementary emotional constraints, while maintaining the non-specialized interface mechanism for animators.

### 4.3.2 Compositional Effects

Emotional labels adjust the lower level scale controls and may alter sampling rates dependent on the selection. These controls have the greatest effect on the musical parts composed at the note-level. Direct scale and key controls then have a lower-level effect. Other influential controls include instrumentation and part-selectors. These two controls represent the orchestral selection mechanism. The number of individual parts (auditory streams), the choice of which subset of the available parts, and the instrument which performs each part, are important in directing the overall effect of musical accompaniment. If objects within an animation have individual musical parts associated, identifiable instruments for each can aid in the flow of music, and its reflectance of motions visually. For instance, the motions of a red sphere and a blue cube in a simple animation could be reflected by musical parts for each, with a piano performing the sphere-part and a guitar reflecting the cube motion. In combination, the two parts

may produce some pleasing and flowing music, but the contribution of each object will be immediately identifiable.

### **4.3.3 Level of Compositional Control**

Although the controls available within the composition process are limited, even with this small constraint space, the variety of possible compositions, and the associated combinations of constraints is large enough to have produced aesthetically pleasing, complex soundtracks. The auditory streams supplied to the composition module themselves are dynamically changing so the effects of the combined constraints can be esoteric in nature.

As with the data-filtering design process, the choice of compositional constraints is often most successful through a combination of trial-and-error and progressive refinement. The choice of constraints can lead to results that were not initially desired, but which may prove more pleasing to the listener. And the interactivity of the process makes it an enjoyable exploration in most cases.

## **4.4 Combining Realistic and Musical Sound Streams**

### **4.4.1 Applying Global/Environmental Effects**

Environmental effects applied for realistic effects are certainly appropriate, and can create a more natural, and effective soundtrack. With musical effects, such environmental effects are extrinsic; musical streams do not belong to the physical environment, and thus are outside the associated environmental effects. However, in some cases, application of those effects can benefit the soundtrack. For instance, simple stereo-placement can add to the extraction of each object's contribution to a soundtrack by the listener. The effect can be distracting if the object crosses the plane perpendicular to the listener between the ears too often, however. Delay effects have been found to be too distracting for musical application as the inherent tempo of a flowing motion is disrupted by the delay effects. Reverberation and attenuation can be effective in giving

a sense of the volume of a space without being distracting. The levels of each effect may be disquieting if over-applied, however.

The system allows combination of musical and realistic effects through the inter-stream processing by applying global effects to the input auditory streams in a binary (on/off) way. Musical streams may bypass the realistic global effects completely and have only musical constraints applied; similarly, realistic effects will have no musical context applied, but will have environmental effects attached. Other streams may be provided both sets of constraints.

#### **4.4.2 Over-constraining sound streams**

A problem with applying several effects on sound streams is that of over-constraining the streams. If too many effects are applied, the effects of some may be diluted or obscured by others. This can lead to soundtracks where subtlety is diminished and the overall effect is “washed-out.” The removal of one or more constraints solves this problem; the determination of which constraints to remove may be evident, or may be difficult to ascertain. A trial-and-error method can always be employed in the latter case as the interactivity of the process allows users to listen to their soundtracks and then make adjustments with little delay (dependent on the rendering power available). This has not been perceived to be a particularly extensive problem however, and can be avoided if the composition constraints are applied incrementally.

## **Chapter Five: Rendering Issues**

### **5.1 Renderer Requirements and Desired Features**

Since the resultant soundtrack is all the user is interested in, the rendering system should be a “black-box” to the animator. Several renderers have been – or are currently – interfaced to, including MIDI and CSound renderers, which together encompass much of the computer music research community’s rendering capabilities. Additionally, notated musical scores can be produced using third party applications. In an ideal world, such composed scores could be “rendered” by a live orchestra with these notated parts provided to the musicians. This is beyond the scope of most animation productions, however, so renderers of a more accessible nature remain the mainstays of the system.

The rendering process can actually be looked upon as two separate sub-processes depending on the type of renderer used. For non-real-time applications, the *mkmusic* system actually produces formatted score-files to be read by third party applications (including MIDI, CSOUND, TEXT and VAS renderers). This separates actual sound output from the system itself, and the use of scores allows for storage and later performance. In real-time applications, two current methods are implemented to produce sound output at interactive rates: these will be discussed in more detail later.

For musical soundtracks, MIDI [Roth92] is an ideal rendering method, since the MIDI protocol was created with musical application in mind. Realistic effects can also be done however, with appropriate sampled sounds stored on the MIDI device, and triggered by the soundtrack system. The CSound renderer has been developed for computer music research so its sound quality is high also. Other supported renderers include the VAS renderer, developed locally utilizing timbre-tree sound structures, and a PC-based sequencer known as SRend.



The rendering of the soundtrack is all important (just as the final image produced with a particular shader is), so the rendering module is most influential. As aural rendering is a complex and specialized field, the *mkmusic* system deliberately tries only to support existing renderers rather than develop its own independent one. This should be left to sound experts rather than computer graphics and animation researchers.

The rendering module of the *mkmusic* system works by outputting audio control information; the format this information takes is dependent on the actual method of auralization into an analogue audio signal.

## **5.2 Supported Renderers**

### **5.2.1 CSound**

CSound was developed at the University of Leeds, UK, by Barry Vercoe. The system allows sound design and sequencing using C-language like functions. Users can specify *instrument* definitions and scores which describe note-sequences of instruments.

Using these scripted scores, the CSound system interprets instrument definitions to produce sound output. The instrument specifications are parameterizable, and the score-files can pass time-varying parameter values to the instruments. This is very suitable for the binding methodology the *mkmusic* system uses, as it provides a template for the available parameters in the sound domain to which motion parameters can map. Instrument specifications themselves are technical in nature, as they tend to describe sounds from their fundamental synthesis processes. The parameter-passing protocol defines the first four values passed from a score-file to be an instrument index, a time-stamp, a duration, an amplitude, and a frequency for a note-value. Other parameters can be passed through the expected values of the instrument specification.

```

        sr = 11025
        kr = 441
        ksmps = 25
        nchnls = 1
        ; p4 amps are here doubled
; guitar
    instr 1
    kamp linseg 0.0, 0.015, p4 * 2, p3 - 0.065, p4 * 2, 0.05, 0.0
    asig pluck kamp, p5, p5, 0, 1
    af1  reson asig, 110, 80
    af2  reson asig, 220, 100
    af3  reson asig, 440, 80
    aout balance 0.6 * af1 + af2 + 0.6 * af3 + 0.4 * asig, asig
    out aout
    endin

; hammer/pull
    instr2
    kamp linseg 0.0, 0.015, p4 * 2, p3 - 0.065, p4 * 2, 0.05, 0.0
    kfreq linseg p5, p7 * p3, p5, 0.005, p6, (1 - p7) * p3 - 0.005, p6
    asig pluck kamp, kfreq, p5, 0, 1
    af1  reson asig, 110, 80
    af2  reson asig, 220, 100
    af3  reson asig, 440, 80
    aout balance 0.6 * af1 + af2 + 0.6 * af3 + 0.4 * asig, asig
    out aout
    endin

; harmonics
    instr 3
    kamp linseg 0.0, 0.015, p4 * 2, p3 - 0.035, p4 * 2, 0.02, 0.0
    asig pluck kamp, p5, p5, 0, 6
    out asig
    endin

```

**Figure 5.2.1:** CSound Orchestra file for a guitar instrument.

**Figure 5.2.1** shows an example instrument definition (in this case a guitar). Both static and dynamic parameters exist: in this case additional frequency adjusters for hammer/pull effects in plucking the guitar strings ( $p5..p7$ ). Several instruments may be defined in a single file known as an *orchestra* file. The instrument descriptions are thus abstracted from the scoring of a musical piece.

**Figure 5.2.2** shows a score-file for CSound rendering. In addition to aural information, the delimiter marks comments in the score. As can be seen, instruments are addressed by index, and provided the note-specifier lines (each score-line specifies a note) are formatted with the

appropriate parameter set. No user manipulation of the orchestra file is necessary.

```
;Musical Score Generator
;-----

;Datafile contained 105 lines.
;105 lines processed in total.

;Input Emotion was: ANY. (Emotional processing is bypassed.)
;Scale is DIMINISHED.
;Performance is CSOUND.
;Style is CLASS.

;INST  TIME  DURATION  AMPL  FREQ  PART
i1     0.000  1.000    8000  261.632 ;Part1
i1     0.000  1.000    8000  261.632 ;Part2
i1     1.000  1.000    8000  261.632 ;Part1
i1     1.000  1.000    8000  261.632 ;Part2
i1     2.000  1.000    8000  440.000 ;Part1
i1     2.000  1.000    8000  440.000 ;Part2
i1     3.000  1.000    8000  440.000 ;Part1
i1     3.000  1.000    8000  349.232 ;Part2
i1     4.000  1.000    8000  440.000 ;Part1
i1     4.000  1.000    8000  349.232 ;Part2
i1     5.000  1.000    8000  261.632 ;Part1
i1     5.000  1.000    8000  261.632 ;Part2
i1     6.000  1.000    8000  440.000 ;Part1
i1     6.000  1.000    8000  440.000 ;Part2
```

**Figure 5.2.2:** CSound Scorefile example for 2-part, single instrument performance.

The major limitation of CSound for direct use as an animation soundtrack system is the non-intuitive nature of the parameters used in sound generation. The *mkmusic* system attempts to interface between these sound parameters and motion-control values so as to remove this burden from computer animators. In doing so, the high quality sound production of the CSound renderer becomes accessible. As CSound is an established sound rendering system, with cross-platform support, and is in wide usage amongst sound researchers, this is a powerful means of generation.

The CSound rendering format can take two forms: scripted output is provided both for regular animation work, and a slightly different format exists to provide “real-time” performance. The latter will be examined in more detail below.

### 5.1.2 MIDI

The Musical Instrument Digital Interface (MIDI) is a control protocol for communication between musical instruments and other related devices (including computers). The standard was developed in the early 1980's as a way of allowing musicians to control digital synthesizers. This provided greater creativity over composition by musicians playing instruments other than keyboards (for instance, guitarists wishing a piano accompaniment), as well as allowing keyboard players to increase their range in overlaying sounds from several source synthesizers. The development of the standard created an explosion in the power and accessibility of musical composition and performance in the same way low-cost personal computers and software brought desktop publishing into common use. MIDI allowed a low-cost introduction to music, with expandability through the interface. The protocol also gave birth to more widespread teaching and educational resources for music including computer-aided instruction. The protocol is based on 8-bit control messages passed between devices. Control is divided into MIDI messages common to all devices, and *system-exclusive* ("Sysex") commands specific to particular manufacturers or devices. In just over a decade, MIDI has become a worldwide standard in musical rendering. Some music researchers frown on the limitations of the protocol for describing intricate musical pieces, but the simplicity and speed of transmission has been a powerful addition to musical performance. Given the MIDI-standard is so pervasive, and with its potential for real-time performance, it is a natural rendering format to support. One limitation is the requirement of a MIDI device to interpret the control messages to actually auralize a score. With the low cost of MIDI-synthesizers (often under \$100), and the high quality and extensibility, this requirement is not extreme.

Further advantages of the MIDI standard include its low storage overhead (as files are 8-bit, binary format), and cross-platform and multi-manufacturer portability. Sequencing software and hardware devices are readily available, and with the newer General MIDI standard, and

Downloadable Samples (DLS), auralization is more consistent between systems and renderers.

Rendering output from *mkmusic* can take two forms, as with the CSound renderer: both scripted soundtrack files, and real-time piping are supported. With MIDI, control messages are exactly alike for both scripted and piped output, though script-files contain additional timing and miscellaneous information, such as the soundtrack title, and composer. Real-time performance will be described further below, but the process involves sending control messages directly to a MIDI device “on-the-fly.” The *mkmusic* system currently does this by passing these formatted control streams out of the serial port of the host machine, through a simple serial-to-MIDI interfacing box to the MIDI instrument. With the scripted method, these control streams are written to a file and on performance of the scorefile, a similar passing of the messages through the serial ports will be done.

The MIDI standard supports several hundred messages. Currently, the *mkmusic* system supports a core subset of these. Common messages include Note-On, Note-Off, Program-Change, and Pitch-Bend. The first two control actual playing of assigned notes on the MIDI device; the Program-Change allows instrumentation changes (dynamically); the Pitch-Bend message performs a frequency change on output notes. This can be useful in rendering effects such as Doppler shifting of moving sound-sources.

The sounds available to the MIDI renderer are limited to the device connected. One drawback is that these sounds tend to be musical in nature as MIDI was primarily designed for musical performance. Sampling synthesizers provide a way of extending the sounds available to the renderer, but their parameterization is limited. Physically-based modeling synthesizers are now available that generate sounds based on their real-world analogues. These synthesizers are capable of synthesizing sounds in real-time with a wider range of accessible parameters. One example is the Yamaha VL-1 synthesizer which physically models wind instruments. Common

instruments such as trumpets can be simulated, as can mile-long metallic tubes, with the available parameters for air passing through hollow-cylinders which the synthesis is based on. This technique uses virtual acoustic synthesis methods implemented in hardware on the Yamaha keyboard.

Such new developments to MIDI-controllable devices are bridging the gap between control and performance. While audio-DSP has greater flexibility and bandwidth [Smit85], the processing overhead and inadequacies of current physical models make MIDI a practical alternative. As we are dealing with event-driven sounds in animations, and MIDI is inherently an event-driven protocol (with its high-level note-triggering), it is an effective rendering technology.

### 5.2.3 Other supported renderers

Several other renderers are supported that are of some utility to animators. The TEXT renderer simply outputs note events and their associated indexing and timing information for human reading. This is useful in debugging scores when sampling-rates on different sound-streams are set. For instance, given the cryptic nature of some of the specialized renderer formats, their outputs may be difficult for animators to interpret. While listening to the soundtrack can provide information on changes necessary, it is often useful to see how individual streams are contributing at particular times; this is not always possible to determine from the layered audio tracks. **Figure 5.2.3** shows a text-format score.

### Musical Score Generator

-----  
Rendering = TEXT.  
Input Emotion was: HAPPY.  
Scale is: MAJOR.  
Performance is: SOLO.  
Style is: ROCK.

Frame:	15	Time: 0.5000	Part: 1 A-4	FREQ: 440.000
Frame:	30	Time: 1.0000	Part: 1 F-4	FREQ: 349.232
Frame:	45	Time: 1.5000	Part: 1 C-4	FREQ: 261.632
Frame:	60	Time: 2.0000	Part: 1 G-3	FREQ: 196.000
Frame:	75	Time: 2.5000	Part: 1 A-4	FREQ: 440.000
Frame:	90	Time: 3.0000	Part: 1 D-4	FREQ: 293.664
Frame:	105	Time: 3.5000	Part: 1 D-4	FREQ: 293.664
Frame:	120	Time: 4.0000	Part: 1 C-4	FREQ: 261.632
Frame:	135	Time: 4.5000	Part: 1 D-4	FREQ: 293.664
Frame:	150	Time: 5.0000	Part: 1 E-4	FREQ: 329.632
Frame:	165	Time: 5.5000	Part: 1 E-4	FREQ: 329.632
Frame:	180	Time: 6.0000	Part: 1 D-4	FREQ: 293.664

**Figure 5.2.3:** *Text-Format score extract.*

The VAS format is for use by the Virtual Audio Server system developed at the George Washington University. This renderer uses timbre-tree structures to describe sounds in a virtual environment. The VAS system can produce localized sounds through loudspeaker technology, and can maintain real-time performance for simple sampled-sounds, provided processing power is adequate. The VAS renderer is currently under development, and future advances should make this a particularly suitable renderer for virtual environment simulations.

The SONG renderer format is for the SREnd renderer which is a PC-based utility from which MIDI-format files can be obtained. This renderer now runs on the SGI systems on which *mkmusic* was originally developed. The format produced by SREnd is also readable by a DOS sequencing package, allowing manipulation in this environment as well. The SREnd format was developed by Hesham Fouad - a fellow audio researcher at the George Washington University.

```
do part 1000
dos 1 2

so 1 400 1 15
eo 1 6400
so 1 6400 1 17
eo 1 6800
so 1 6800 1 19
eo 1 7199
so 1 7199 1 22
eo 1 7599
so 1 7599 1 27
eo 1 8000
```

**Figure 5.2.4:** *SRend format score.*

**Figure 5.2.4** shows an SRend format score extract; the first two lines declare an instrument, and the following *so-eo* pairs start and end notes for the instrument. The values following the *so-eo* pairs index the instrument, time-stamp the event, and trigger the appropriate note-on for the *so*'s and stop the current note for the *eo*'s.

The support of a variety of renderers is deemed important as it makes the *mkmusic* system accessible to a wider set of users. The system itself has been successfully ported from UNIX systems, to both DOS and Macintosh environments. With supported renderers available on these platforms, the system is thus available beyond the users of high-end graphics workstations. Though interactivity may be more limited on less powerful machines, the underlying design processes of the soundtrack production are consistent. This should also allow users to transparently switch between platforms and operating systems while retaining a core rendering capability regardless of machine-type.

### **5.3 Real-Time Rendering Capability and VEs**

Using the *mkmusic* system, musical soundtracks can be composed and generated in real-time, both for animations and virtual environment simulations. This requires all three processing modules to maintain real-time performance, regardless of data input rates. Realistic effects based



on sampled-sounds are also currently handled. Synthesized sounds may be produced in real-time given sufficient processing power.

The data-filtering and composition modules are both designed to require no inherent batch-processing. This allows them to work in real-time, with no need for any determinism of how a simulation will progress. The customizable nature of the filtering process does allow for this, but it is assumed that the user will plan the filter design according to the application. We assume no lower limit on data supply for composition to take place. The system will continue to output the last composed note for each part unless explicitly countermanded. Even if there is a lag on data supply, the soundtrack should maintain its integrity. Unfortunately, this could mean that the soundtrack does not synchronize to visuals, due simply to delays in receiving data. This is addressed in two ways. First, the use of common data by motion controller and soundtrack producer should eliminate these effects. If for some reason this is not the case (such as passing the soundtrack data to a remote machine for *mkmusic* processing), timing constructs within the composition module can be made to re-synchronize the audio track to the motion controller.

Two renderers are supported which allow compositions to be performed at interactive rates. These are the MIDI renderer, which outputs directly to a serially connected MIDI device, and a score-based CSound renderer. The CSOUND\_RT renderer works by processing scores on a line-by-line basis. The *mkmusic* system pipes suitably formatted lines to the renderer allowing interactive composition and performance. One drawback is that the processing overhead of CSound is heavy, and real-time performance may not be maintained. The VAS renderer, also capable of real-time sound output, is not included in this discussion as its current state of development has not allowed sufficient testing. Early trials have suggested this renderer to be an ideal one for the future, due to its flexible parameterization scheme, and its own targeting of motion-control bindings with aural simulation.

il	0.000	0.500	5727	123.472
il	0.500	0.500	15506	146.528
il	1.000	1.000	15991	880.000
il	2.000	2.000	14627	440.000
e				

**Figure 5.3..1:** *CSOUND score extract.*

**Figure 5.3.1** shows an extract of piped output to the CSOUND\_RT renderer. Each line has several expected parameters for a particular instrument specification. (In this case a guitar instrument.) The first parameter is an instrument index and the numbers following represent a time-stamp, duration, amplitude, and frequency (note-value) for the instrument. Each line thus represents a single note in the performed ‘score.’ This format is very similar to the regular CSound output, and is still fundamentally a scripted rendering. With the non-deterministic nature of virtual environment simulations, scripting is a limiting technique, as foreknowledge of future events is often required.

In the case of the MIDI renderer, either a non-interactive MIDI-format file can be generated, or direct message-passing to an external MIDI device can be done. The file format method does not allow for interactive performance, but allows easy porting within the MIDI domain. The direct output method supports real-time applications, and virtual environment simulations. The great advantage of the MIDI renderer is that it offers real-time performance without placing too great a processing burden on the simulation machine. The direct MIDI rendering also avoids the batch-processing limitations of other renderers. Output is based purely on the current audio-state, with updating of sounds done on a frame-by-frame basis (constrained additionally by set sampling-rates).

Such interactive production also allows filters to be refined incrementally. In some sense this development can be compared to developing shaders in a system such as RenderMan, where

modifications to code lead to an image that can be viewed and inadequacies can then be addressed in the code again.

As the *mkmusic* system accepts data input, this supply can be extended beyond motion-control program output. In virtual environments, interaction devices such as data-gloves and head-mounted-displays (hmd's) often have positional-trackers on them. These trackers, often magnetic or ultra-sonic in design, may provide position and orientation information in 3-space. This data can be supplied to the *mkmusic* system in the same way as regular motion-control data is. In fact, the two sources can be combined so that both user action, and simulation feedback are sonified. The data used in this way can provide aural feedback within a VE which may be useful for tasks such as navigation, or used creatively to compose music through user motions in the virtual space.



**Figure 5.3.2:** *VE composed score.*

**Figure 5.3.2** shows an extract of a musical score composed interactively in a VE through use of a 6-DOF ultrasonic mouse. The musical parts were linked to the motions of the mouse through virtual space, binding position and orientation pairs, in each coordinate axis, to three musical parts. Tempo was controlled by linking each part to the average velocity of motion over that dimension. Actual auralization was through a Korg Wavestation MIDI synthesizer. Stereo localization, and attenuation through MIDI velocity controls to left and right channels on the synthesizer were also performed.

Using MIDI devices, or the CSound rendering software, animators and VE researchers can successfully produce soundtracks with performance rates suitable for real-time applications. These include accurately-timed previewing of soundtracks, rendered interactively, and in aural feedback generation for virtual environment simulations. The flexible nature of the filter design and composition processors also allows powerful and customizable control over aural output in such applications. Real-time performance also benefits the refinement and design processes for animation uses, as interactivity is increased with the immediate feedback that is provided.

## Chapter Six: Using *mkmusic*: Results and Examples

### 6.1 Guidelines for Use

#### 6.1.1 Data-File Format

The *mkmusic* system accepts data line-by-line, with no comments or content other than data values. Each parameter must have an entry for each line of the data-file. Each of these lines represents one frame of the animation. Default frame-rate is assumed to be 30 frames-per-second. This is modifiable through a command-line parameter. The system reads and processes each line as it is input so the resultant sound output is instantaneous. The data is sampled however, at a default rate of every 10 frames. This can also be set by command-line input, and further specification of individual sampling-rates for each sound-part can be made in this way also. This is especially useful when mixing realistic sound effects and musical accompaniments. Composing music requires to sample the data far less than at each frame (1/30 of a second) as musical notes of this length would be far too short. The default rate of 10 frames gives three notes-per-second of musical output, assuming the default frame-rate. With realistic effects, we normally wish to trigger discrete events exactly at a particular frame; we thus set the realistic parts' sampling-rates to be 1 - i.e. we process every frame. We then set triggers in the data-mapping to check whether sound output should proceed in a particular manner.

0.00	0.00	0.0	0.00	0.00	1.00	0.0	0.00
0.00	0.18	0.0	1.00	0.00	0.95	0.0	3.00
-0.02	0.37	0.0	2.00	0.01	0.90	0.0	6.00
-0.04	0.55	0.0	3.00	0.01	0.84	0.0	9.00
-0.08	0.73	0.0	4.00	0.02	0.79	0.0	12.00
-0.12	0.92	0.0	5.00	0.03	0.74	0.0	15.00
-0.17	1.10	0.0	6.00	0.05	0.69	0.0	18.00
-0.23	1.28	0.0	7.00	0.07	0.64	0.0	21.00
-0.31	1.46	0.0	8.00	0.09	0.59	0.0	24.00
-0.39	1.64	0.0	9.00	0.11	0.55	0.0	27.00
-0.48	1.82	0.0	10.00	0.13	0.50	0.0	30.00
-0.58	2.00	0.0	11.00	0.16	0.46	0.0	33.00

**Figure 6.1.1:** *Data-file extract with six parameters.*

**Figure 6.1.1** shows an example data-file, with six parameters. These were supplied for the animation of dancing figures described previously. The file itself contains just numbers, and it is the animator's responsibility to know what each parameter represents, and use that knowledge to filter accordingly.

```

-1    1    2
-1   -1   -1
-1   -1   -1
      .
      .
      .
30    2    3
31    1    3
-1   -1   -1
      .
      .
      .

```

**Figure 6.1.2:** *Data-file showing time-stamped collision events.*

**Figure 6.1.2** shows another data-file. This is an example of a file that may be used to produce realistic sound effects. The three parameter values represent the frame-number, and the indices of two colliding objects from a simulation. The -1 values refer to frames where no collisions occur. This data-file contains much redundant information (periods of time when no collisions occur), and this is an area in which the *mkmusic* system requires more interface improvements. As the system was originally designed for musical accompaniments only, the processing was assumed to be for continuously flowing music from the data, and hence a frame continuous input is required. A suitable data-filter handling this file will be described below.

### 6.1.2 Command-line controls

In early development of the *mkmusic* system, animator control was restricted to emotion, scale and instrumentation, and only limited file-IO was implemented. The original interface was prompt-based, as illustrated in **Figure 6.1.3**. At this stage, only a textual rendering could be produced which was then input by hand into a MIDI sequencer for playback. As the functionality has grown, the use of command-line parameters has completely superseded this prompted interface. The prompted interface does remain however, and can be invoked by running *mkmusic* without any parameters.

MKMUSIC

Table of Musical Selectors

EMOTIONS		SCALE		INSTRUMENTATION	
NONE	-1	NONE	-1	ANY (of 1-3)	0
ANY	0	ANY	0	SOLO (1-pt)	1
SAD	1	MAJOR	1	BAND (3-pt)	2
HAPPY	2	MINOR	2	ORCH (6-pt)	3
BRIGHT	3	BLUES	3		
EVIL	4	DOMINANT	4		
PEACEFUL	5	DIMINISH	5		
		HALF-DIM	6		

Enter name of input file:

**Figure 6.1:** *Prompted interface for mkmusic.*

The full range of command-Line options is given in **Appendix A**. Some of the more influential controls are described here. The *-i* and *-o* flags provide the input and output file specifiers. If the supplied input file does not exist, the system will switch to prompting the user for the filename before processing will take place. If no output file is supplied, output is assumed to be to *stdout*. Other flags can specify both input and output to be *stdin* and *stdout* respectively.

This is particularly useful when piping data from a motion control program directly to *mkmusic*, or piping the output score information from executing *mkmusic* to a renderer, such as CSound.

The *-C*, *-I* and *-S* flags control emotion, instrumentation, and scale settings respectively. Each has a default associated, and the full set of defaults can be specified by the *-D* flag. This also sets the renderer (*-R*), duration (*-d*) and part-selection (*-p*) defaults. The emotion default is happy, the instrumentation is solo (1-part per auditory stream), and the scale is major. The defaults for duration and parts are to set these according to the data in the file. Parts will correspond to the number of input data parameters in the file. Duration will be the number of frames in the file. The renderer defaults to the Song-format, as this is used to produce MIDI score-files for later performance, and thus is most useful for development.

The *-p* flag sets the number of parts to be output by the system. Additionally, with optional index inputs, delimited by the + character, which parts are to be output can be controlled. For instance, the setting: *-p3+2+4* will output 3 parts, namely parts 1,2 and 4.

Similarly, the *-s* flag sets the sample-rate for the data-filtering process. Although all data is parsed by the system, the filters are only applied at sampled frames. A single global state can be set by specifying a single number after the flag, or values for each part can be set with the + delimiter between. For example, *-s10+20+1* would set a sampling-rate of 10 frames for part 1, 20 frames for part 2 and 1 for part 3. All other rates would be the same as that for part 1, namely 10 frames in this case. This is useful when certain parameters and motions are more active than others and thus require higher sampling-rates. Note that the "sampling-rate" specifier is really a "sampling-period" as it actually defines the number of frames between samples rather than the number of samples-per-unit-time.



The frame-rate is set by the *-f* flag. This is a very useful control as animations are often laid to tape with duplicated frames in order to lengthen the animation without having to render all the frames. For instance, rendering 150 frames of an animation would provide a 5 second duration, at 30fps. By laying each frame to tape twice, the animation doubles in length to 10 seconds. The motions will all be slowed by a factor of two in this case. The data provided by the motion control system will still only represent the original 150 frames however. By setting the frame-rate to 15 instead of the default 30, *mkmusic* can compensate for this. The filter-application process is controlled by the *-N* flag. Several default filters exist that can be applied to the input data. These will be described more, below. The *-Ncustom* flag is the most likely setting for users who have designed their own, application-specific filters. This custom setting is also the default. Other flags that influence the output include volume controls, and attenuation bounds that allow attenuation based on the specified size of the environment (limit-of-audibility). Flags controlling particular renderer set-ups are also provided and can be reviewed in **Appendix A**.

**Figure 6.1.4** shows a typical command-line execution of the *mkmusic* system. In this example, an input file named "In.data" is processed to output a text-rendered, three-part score, saved to a file "Out.score." The music for the composition will be custom filtered, and will have sad, blues characteristics. The data will be sampled at a frame period, with a frame-rate of 10 fps. The use of the *-D* flag is optional, but is included as it guarantees any un-set controls will be at their default values.

```
mkmusic -iIn.data -oOut.score -D -Rtext -Ncustom -p3 -55 -f~0 -esad -Sblues
```

**Figure 6.1.4:** *Sample command-line execution of mkmusic.*

The *mkmusic* system has several default filters that can be applied generically to input data. These include *bound*, *cutoff*, and *kick* filters. The cutoff filter is a static bounding filter that constrains input data values to a range. Values below or above the range are set to the minimum or maximum range values respectively.

The *bound* filter takes this further by implementing a floating-bounded-range dependent on the previous input. Values exceeding the range become the new bounds so that a dynamic normalization is done on subsequent values. This is a more useful filter as the static settings of the cutoff filter are dependent on the initial data values read in, and this may not be representative of the data following.

The *kick* filter is similar to the *bound* filter but rather than setting the range bounds to the current maximum and minimum values, it 'kicks' these values beyond the current extremes. This filter was written in response to data supplied being monotonically increasing or decreasing. In this case, using the *bound* filter the range-bounds would constantly be the current value read. The filter would then output a constant value corresponding to the upper or lower normalization-bound value, and lead to music with a single note value. The *kick* filter, while still limited, at least replaces this with a sawtooth-like output.

In addition to these simple data manipulation filters, several utility functions are provided. All mathematical functions available within the C-language math library are available. System functions providing magnitude operations on vectors, checking whether values are contained within lists, and pseudo-velocity/acceleration calculators are some of the other utilities included.

These utility functions are provided for users in their custom filters. Custom filters are expected to be designed if the soundtrack produced is to be truly application-specific. The custom filter design process involves the user defining the body of a system-declared custom filter. The user modifies “custom.c” and “custom.h” files, and links them with the *mkmusic* library which contains the rest of the system functionality. **Figure 6.1.5** shows sample custom files, in this case for a sonification of 3D morphing data.

```

/*                                     */
/*                                     custom.h: Put user-custom definitions here. */
/*                                     */

#include "globals.h"

#ifndef GENERAL
#define GENERAL

#include <stdio.h>
#include <ctype.h>
#include <stdlib.h>
#include <stddef.h>
#include <string.h>
#include <math.h>
#endif

#ifndef CUSTOM
#define CUSTOM

/* Do not modify this. */
extern int CustomDataMap(int ndata, float *data,
                        int nparts, int *note, float *supp);

/*****/
/* Add stuff below. */
/*****/

/* Custom Defines */
#define USER1 100000
#define USER2 316.0

#endif
/* End of custom.h. */

```

**Figure 6.1.5a:** Custom header file for filter design process.

```

/*****/
/*                                     Music Generator                                     */
/*                                     (Custom.c)                                       */
/*                                     */
/* Map parameters from an input file to musical notes.                               */
/* This file should contain the module: CustomDataMap.                               */
/*****/

#include "music.h"
#include "datamap.h"
#include "custom.h"

/*****/
/* NB: The following module must be defined at least as a dummy. */
/*                                     */
/* CustomDataMap: This is the user-defined custom mapping. */

```

```

/* INPUTS:   int ndata    = number of data values.          */
/*           float *data  = data storage array.             */
/*           int nparts   = number of music parts.         */
/*           int *note=   array of output notes.           */
/*           float *supp   = array of supplemental info. (frame)*/
/* OUTPUT:   int ~ ignored for now. [Error checking.]      */
/*****/

int CustomDataMap(int ndata, float *data, int nparts, int *note, float *supp)
{
    /* Example custom map for datafile: cube/cyl.data.      */
    /* cube/cyl.data have 4 data values. data[0..2], [3..5] etc. */
    /* are xyz coordinates.                                  */
    int i,          /* Temp. counter variable. */
        vs = 3,    /* Data points are 3D vectors. */
        num_notes = 4; /* Output four notes. */

    /* We take magnitudes of xyz triplets for the point motions. */
    for (i = 0; i < num_notes; i++)
        /* VecMag is a system utility magnitude calculator function. */
        note[i] = (int) (VecMag(vs, data, (i * vs), USER1, USER2));

    return (num_notes); /* Return the number of parts we custom-set here. */
} /* End of CustomDataMap. */

/* Place local utility functions here. */

/* End of Custom.c. */

```

**Figure 6.1.5b:** Custom source file for filter design process.

This example will be discussed in more detail below. These two files are the only files expected to be modified by the user. The filter may be fully-described within the body of the CustomDataMap function, or be sub-specified by user functions included below it.

This is the most common way of designing the custom filters. More examples of these sub-filters can be found in **Appendix B**. The user can write a subfilter to handle all the output streams, or can write individual filters for each, and call them from the system filter. One thing of note is that the *supp* parameter to the custom filter contains supplementary information, most important of which is the current frame index.

In addition to filters such as the one seen above in **Figure 6.1.5** which produces a continuous data sonification, we can write discrete timing-filters for the data. For example, a

filter for the data-file described in **Figure 6.1.2** could be defined as in **Figure 6.1.6**.

```
int CollisionHandler(int frame, int obj1_ind, int obj2_ind)
{
    coll_ind = obj1_ind + obj2_ind.

    if (frame != -1)
    {
        switch (coll_ind)
        {
            case 3:
                return(COLL_3_NOTE);
                break;

            case 4:
                return(COLL_4_NOTE);
                break:

            case 5:
                return(COLL_5_NOTE);
                break:

            default:
                return(GENERIC_COLL_NOTE);
                break;
        }
    }
    else
        return(-1);
}
```

**Figure 6.1.6:** Trigger-filter example for collision-event sounds.

We can supply the *collision\_handler* function with the input parameters *in(O.2J)*, and assign the output *note notefof* based on the collision event. If we define different note values to the *COLL\_NOTE*'s and define a default sound for collisions we haven't planned for, *GENERIC\_COLL\_NOTE*, we can distinguish on output which collision occurred. By outputting -1 when no collisions are triggered, we define a *NULL-stream* in-between. We could equally well filter based on other combinations other than adding colliding-objects' indices. The above method would not distinguish between collision-object pairs of 2-3 and 1-4 for instance. This is a simple example of a trigger-filter. We do not use the frame-number for instance; we could add an additional parameter which changed the sound of a collision based on how far into the animation it occurred. Double triggering on both collision event and global-time in that case.

In designing filters, we are assuming no knowledge of the command-line options provided. The filtering-mode must of course be set to *custom*, but the user must be careful in setting parameters such as sample-rate. In a trigger filter where a sound is started at frame  $t$  for instance, the user must be careful that the sampling-rate is a factor of  $t$ , or the filter may be called at a frame preceding  $t$ , and then next called at a frame after  $t$ . Thus, formally, we want a sampling-rate  $S$ , where  $t \bmod S = 0$ .

## 6.2 Animation Examples

We now present a couple of scores produced with the system as soundtracks for animations. The image production was with RenderMan™ in the first example, and with SoftImage™ in the second.

```

Frame 0 Time 0
camera 0.2 2 0.5 0 0 0.3 0 0 1
hinge1 -0.291519 0.0 0.0718462 -2.88312
hinge2 -0.443678 0.0 0.201645 -0.305678
hinge3 -0.302622 0.0 0.34343 -1.1705
headforce 0
baseforce 0.447398

Frame 1 Time 0.0333333
camera 0.2 2 0.5 0 0 0.3 0 0 1
hinge1 -0.284728 0.0 0.0830806 -6.17764
hinge2 -0.458547 0.0 0.18201 0.675101
hinge3 -0.314729 0.0 0.320993 -2.1993
headforce 0
baseforce 0.152364

.
.
.

Frame 240 Time 7.99999
camera 0.2 2 0.5 0 0 0.3 0 0 1
hinge1 0.328212 0.0 0.0678433 -2.77636
hinge2 0.141395 0.0 0.139254 -0.715218
hinge3 0.319698 0.0 0.229851 -1.98297
headforce 0
baseforce 0.464356

```

**Figure 6.2.1:** Data-file for lamp animation.

This animation was demonstrating a motion-control technique utilizing genetic programming in searching a dynamic constraint space for an articulated figure. The lamp in the simulation learns how to move towards a certain goal, under physical constraint factors, using genetic evolution [Grit95]. The soundtrack was generated from the data-file in **Figure 6.2.1**. This data-file was pre-filtered prior to input to the *mkmusic* system to format it appropriately.

The target of the lamp was to jump to a specific spot in front of it. Given this goal, any motion not in the direction of the goal would be detrimental. With this simple musical direction, a filter was designed to map the lamp's motion (based on its joint positions and base force) to musical notes. If the motion was backwards, away from the target, the music should reflect this as something incorrect. The filter in **Figure 6.2.2** was used to map the motions to music.

```
int LuxoMap(int dir, float dis, float st)
{
    int val;
    float scale = 3.5f;

    if (dir == 1) val = (int) (dir * dis * st);
    else val = (int) (st + dir * dis * st * (scale * drand48()));
    return (val);
}
```

**Figure 6.2.2:** *Data-filter for lamp animation.*

The filter was based on three input parameters: a direction which was established from the position values of the joints relative to the starting position; the distance from the starting position; and finally, the baseforce parameter was used as a scaling factor. In the case of the lamp moving in the wrong direction, the same mapping was applied but with the additional attachment of a random element being multiplied into the output note value. This would throw the musical output several notes higher (with the additional scale value). and provide feedback that something was “not right.” The filter was designed heuristically and was more an intuitive consideration of what should happen, rather than a well-planned and tested theory. Adjustments were made to the scale-factor and to the amount of influence the random element had, during a refinement process. For example, originally, the random element was added rather than

multiplied, but this was found to produce an insignificant difference in output between the two directions of motion. The final soundtrack was fairly effective, with the music jumping erratically when the lamp falls backwards in the animation, but flowing more predictably when the motions were towards the target.

```
int TexMap(float val)
{
    int i;

    val = val / 10;

    for (i = 0; i < 2; i++)
        val = (val * val) + i;

    return((int) val);
}
```

**Figure 6.2.3:** *Data-filter applied to texture information for cloud objects.*

This animation extract was a segment of a longer animation containing windchimes. The rocking-chair is seen to be slowly rocking back-and-forth at the end of the animation, with a background cloud-filled sky rapidly turning dark as the sun sets. The soundtrack was composed using *mkmusic* for both the earlier chime movements and this chair-segment. The first requirement was that the switch to the chair from the chimes as the object of attention be reflected by the music. This was accomplished by an instrumentation change from a layered piano and choral sound for the chimes to a stringed-orchestral sound for the chair. The second requirement was that the music reflected both the chair and also gave some sense of the changing light of the sunset. Data was provided for both the angular changes in the orientation of the chair, and texture and lighting information for the clouds. These were layered together to produce a flowing musical output which was both peaceful and yet had an underlying dynamism consistent with the coming dusk. **Figure 6.2.3** shows the data-filter for the cloud-texture input data.

The exact nature of the filter is to first reduce the texture-data value to smooth the input changes, and then apply a repeated multiplication with the indexed-offset to produce an



abstracted note from the specific numerical value. Although a fairly arbitrary mapping, it was quite effective in concert with the related chair and lighting filters. The above filter however, came about as a series of adjustments and refinements. Initially, a linear filter was written and the output was then observed. From here, the smoothing of the input was introduced to the data values. At this point, the mapping was far too direct; the repeated multiplication came about as first a single application, and then incrementally to a single-repetition, and then the addition of the offset. At this point, the music was suitably abstracted but still flowed well with the visuals.

### 6.3 Sonification Processing

Finally, an example illustrating a slightly divergent use of the *mkmusic* system from its aesthetically-pleasing musical-accompaniment goals is presented. This is a direct sonification of a 3-D morphing animation. Here, no aesthetic quality to the output soundtrack was desired. Rather, a straight aural processing of the data was the aim of this soundtrack. The sonification of a morphing from a cube to a cylinder was done using the data-filter described in **Figure 6.1.5** previously. In the example below, this filter was applied to four input parameter points on each of the objects. **Figure 6.3.1** shows the input data file containing the positions of the points as they progress in the morph.

0.000000	1.000000	0.000000	-0.462333	0.731166	-0.462333
0.499617	-0.707007	-0.499617	0.460221	-0.730111	0.460221
0.000000	1.000000	0.000000	-0.468890	0.734445	-0.468890
0.505720	-0.709267	-0.505720	0.466804	-0.733402	0.466804
0.000000	1.000000	0.000000	-0.475447	0.737723	-0.475447
0.511822	-0.711526	-0.511822	0.473386	-0.736693	0.473386
0.000000	1.000000	0.000000	-0.482004	0.741002	-0.482004
0.517924	-0.713786	-0.517924	0.479969	-0.739985	0.479969
0.000000	1.000000	0.000000	-0.488560	0.744280	-0.488560
0.524026	-0.716045	-0.524026	0.486552	-0.743276	0.486552

**Figure 6.3.1:** Data-file extract for morphing animation.

Each pair of lines in the above extract represents a single line of the data-file (formatted to fit the page). The data values represent the xyz-coordinates of four points on the 3-D object. The changes in each parameter are very small, and are exaggerated in the VecMag function with the USER1 scaling-value. The USER2 value is then used to rescale the resultant magnitude back down. The resultant score for the two objects is given in **Figure 6.3.2**.



**Figure 6.3.2:** Score extract for 3-D morphing sonification.

One interesting feature is that in the scaling of the magnitude using the *USER\** values, we introduce some numerical error as we use *100000* to scale up our squared-values, but in using *316.0* in the scaling down, we are not precisely using the square-root of *100000*. This is reflected in the score: two of the data values for the morph are for constant positions on the objects. That

is to say, neither changes position throughout the animation. These should lead to constant musical notes on output. In the extract shown here, it can be seen that all eight parts change through time. Using `sqrt (USER1)` instead of `USER2` in the data-filter was found to remove this numerical anomaly.

More scores can be found in **Appendix C**. These include full-scores of some of the example extracts given throughout, as well as other scores which represent soundtracks created by varying either command-line parameters or data-filters on common data. All such scores can be seen to be similar in nature but are recognizably different on examination of each part individually.

## **Chapter Seven: Future Work and Conclusions**

### **7.1 Future Work**

The goals of the *mkmusic* system are to be a soundtrack development tool suitable for use by animators, and to create effective and pleasing soundtracks. Given these two goals, future work is aimed at enhancing the system's capabilities in these two areas.

The development of a "drag-and-drop" interface for filter design should aid in the creation and refinement of data-filters greatly. Currently, the development requires code changes followed by system processing to hear results. Code changes must then again be performed to make modifications. Often, this trial-and-error method is unintuitive and tedious. Allowing graphical interfacing, and the intuitive feedback of connecting filter icons into a viewable pipeline should benefit users' understanding of the filtering taking place. Users with little technical programming background would also be aided by such an addition.

The creation of more effective soundtracks involves two types of sound, but a similar solution. Sound effects would be improved by refining the parameter bindings between motion and sound worlds. With musical accompaniments, the infinite variety of compositions from a given data-set is limited to and by the simplistic musical constraints of the system. Each of these limitations can be considered as search-problems, either within parameter-space for realistic sounds, or musical construct-space for accompaniments. The use of genetic techniques has been shown to aid in both sound effects [Taka93] and music production [Uotri91]. Genetic algorithms are a method of searching a parameter space in a target-oriented way. By providing an interactive selection mechanism, users can choose at each generation from a shortlist of possibilities, and the most preferred choice can then be further refined. Successive application through generations of mutated choices lead to a progressive search of the space as directed by the user. With sound effects, the target is the "most realistic," or perhaps more pertinently, "most effective" sound for a given event. The criteria for what constitutes the most effective sound is purely perceptual by

the listener, although constraints can be placed to make sounds exhibit certain characteristics (such as specific envelope, frequency, bandwidth, etc). Slight mutations of classes of sounds can often add color to a soundtrack, seeming more natural than repetitions of a single sampled sound, for instance. This hyper-realism is often perceived in pure physical-modeling as well, where sounds seem too pure and cold for a natural environment. With musical composition, the current limitations of the emotional and scale constraints are obvious. The subtleties of musical composition between these simple labels are lost. By allowing a search over any number of musical constructs, such as pitch, instrumentation, harmony, etc. we can access the wider compositional space. Several concerns with this application exist however. Suitably encoding all the musical constructs desired may be difficult. An over-constraining of the musical output may make the use of input data negligible. We should at least maintain some control over tempo of music from the data to retain synchronization with visuals. The major advantage of this process is that animators may be able to create music more specific to their needs or desires, while being minimally involved in the creative process. A point-and-click interface, with a choice of perhaps five to ten genetically composed pieces would allow the animator to narrow down their selections quickly. Another advantage is that this refinement process may actually lead animators to compositions they had not considered, but which they may in fact prefer.

Finally, as the use of the system continues, I foresee a library of filters becoming available, where custom filters would be designed and layered with existing ones, allowing filters to be reusable in developing applications. This would be similar to the repositories of shaders developed using the RenderMan™ Shading Language.

## 7.2 Conclusions

While the *mkmusic* system was primarily developed to compose musical accompaniments, it uses high-level design methodologies that are applicable to all soundtrack

production needs. The system is versatile enough to be used as a data sonifier, or an aural feedback generator with real-time performance. Its primary function however, is to create soundtracks for animations and virtual environments based on the motions of the objects within them. A correspondence between the motion and sound domains exists naturally in the real world. Sounds are physically generated due to motions of objects within an environment. This system exploits that relationship and allows flexible application ideally suited for the motion control paradigms used within computer animations and virtual environment simulations. The generality of the approach, and the use of motion data allows dynamic synchronization to visuals with little additional overhead.

The *mkmusic* system has effectively created aesthetically pleasing musical soundtracks and appropriate sound effects, unique to the motions they reflect, with minimal user expense. It therefore has been shown to be a suitable tool for animators to produce soundtracks with. The system is flexible and customizable so as to allow more knowledgeable users to develop more complex control and composition modules. This provides a common tool from which experts and non-experts alike, can create soundtracks. This in turn should allow animators the opportunity to develop their audio-production skills using the extensions and techniques of more experienced sound professionals. The system does attempt to bridge the gap between sound and visual arts worlds in this way.

In the future, with tools of this type, along with closer collaboration in the teaching and production of graphic arts and sound/music, animations and multimedia of the past may seem as antiquated as silent films appear to modern moviegoers now.

### **7.3 Acknowledgements**

I would like to thank everyone who contributed to my research and writing of this thesis. In particular, my advisor, Professor James K. Hahn, for his guidance, support and perseverance

during my graduate studies. Special thanks also to Robert Lindeman, a fellow student at GW, for pushing me in so many ways, to make this thesis a reality. The existence of this thesis owes much to Rob's work ethic and friendship. I would also like to thank the other members of the Institute for Computer Graphics, for their numerous contributions to my research efforts. In particular, the active audio researchers, Hesham Fouad, Joe Geigel, and Won Lee, without whom none of this would have been possible. I'd also like to thank Amit Shalev, another GW student, for his friendship, timely advice and support. Finally, I want to especially thank my family, in particular my mother Rama, my late father, Gangadhar, and my brother Subhash, for providing me their support and love throughout the years, to allow me to research this field of study, and work in such an exciting field.

## References

- [Altm85] R. Altman: "The Evolution of Sound Technology",  
in "Film Sound: Theory and Practice", E. Weis and J. Belton, [Eds].  
Columbia University Press. NY. 1985.
- [Barg93] R. Bargar, "Pattern and Reference in Auditory Display"  
*SIGGRAPH'93 Course Notes #23: Applied Virtual Reality*, 1993.
- [Blat89] M. Blattner, et al. "Earcons and Icons: Their Structure and Common  
Design Principles."  
HCI, Vol, 4:1, pp11-44. 1989.
- [Blau83] S. Blauert, "Spatial Hearing"  
MIT Press, Cambridge, Mass. 1983.
- [Bly85] S.A. Bly, "Presenting Information in Sound"  
*Proc, CHI'85*. pp371-375. New York, 1985.
- [Bori85] J. Borish. "An Auditorium Simulator for Domestic Use."  
*Journal of the Acoustical Soc. of America* 33(5), pp330-341, 1985
- [Burt94] G. Burt: "The Art of Film Music"  
Northeastern University Press, Boston, 1994.
- [Calh87] G.L. Calhoun, et al. "3D Auditory Cue Simulation for Crew Station  
Design & Evaluation."  
*Proc. Human Factors Soc.* 31, pp1398-1402. 1987.
- [Came80] E. Cameron, W.F. Wilbert, and J. Evans-Cameron.  
"Sound and the Cinema."  
Redgrave Publishing Co, New York, 1980.



- [Cook84] R. Cook, "Shade Trees"  
*Proceedings of SIGGRAPH'84*,  
 Computer Graphics Vol.18:3. pp195-206.
- [Dann91] R.B.Dannenberg, C.Fraley, and P.Velikonj,  
 "Fugue: A Functional Language for Sound Synthesis",  
 IEEE Computer, Vol.24, No.7, pp36-42. 1991.
- [Dann93] R.B.Dannenberg,  
 "Music Representation Issues, Techniques, and Systems."  
*Computer Music Journal* 17(3):pp20-30. 1993.
- [Doll86] T. Doll, et al. "Development of Simulated Directional Audio for Cockpit  
 Applications"  
 AAMRL-TR-86-014, WPAFB. Dayton, OH. 1986.
- [Evan89] B-Evans.  
 "Enhancing Scientific Animations with Sonic Maps"  
*Proc. 1989 International Computer Music Conference. International  
 Computer Music Association, 1989.*
- [Fost88] S.Foster. "Convolutron Users Manual"  
 Crystal River Engineering, Groveland, CA.
- [Fost91] S.Foster, et al. "Realtime Synthesis of Complex  
 Acoustic Environments [Summary]."  
*Proc. ASSP Workshop, New Paltz, NY. 1991.*
- [Frat79] C. Frater: "Sound Recording for Motion Pictures"  
 A.S. Barnes, NY. 1979.

- [Furn86] T.A. Furness. "The Super Cockpit and its Human Factors Challenges"  
*Proc. Human Factors Soc.*, Vol.30: pp48-52. 1986.
- [Gave91] W.Gaver, et al.  
"Effective Sounds in Complex Systems: the ARKola Simulation"  
*Proc. CHI'91*. pp85-90. 1991.
- [Gave93] W.Gaver. "Synthesizing Auditory Icons",  
*Proc. INTERCHI*, 1993.
- [Gehr90] B.Gehring. "FocalPoint 3D Sound Users Manual"  
Gehring Research Corp. Toronto, Canada. 1990.
- [Gorb87] C. Gorbman. "Unheard Melodies."  
Indiana University Press: Bloomington, Indiana. 1987.
- [Grit95] L. Gritz, and J. Hahn,  
"Genetic Programming for Articulated Figure Motion"  
*Journal of Vis. and Comp. Animation*, Vol. 6: pp129-142. 1995.
- [Hahn88] J. Hahn. "Realistic Animation of Rigid Bodies."  
*Proc. SIGGRAPH'88*, ACM Computer Graphics,  
Vol. 22: No. 3, pp299-308. 1988.
- [Hahn95a] J. Hahn, J. Geigel, L. Gritz, J. Lee, and S. Mishra.  
"An Integrated Approach to Motion and Sound."  
*Journal of Visualization and Computer Animation*,  
Vol. 6: No. 2, pp109-123. 1995.
- [Hahn95b] J. Hahn, H. Fouad, L. Gritz, and J. Lee.  
"Integrating Sounds and Motions in Virtual Environments."  
*Presence Journal*, MIT Press. Vol. 7: No. 1, pp67-77.

- [Horn91] A. Horner, and D. Goldberg.  
“Genetic Algorithms and Computer-Assisted Music Composition.”  
Proceedings of the 1991 International Computer Music Conference,  
International Computer Music Association, pp479-482. 1991.
- [Karl94] F. Karlin and R. Wright.  
“On the Track: A Guide to Contemporary Film Scoring.”  
Collier MacMillan, London. 1990.
- [Kram91] G. Kramer and S. Ellison.  
“Audification: The Use of Sound to Display Multivariate Data.”  
Proceedings of the 1991 International Computer Music Conference,  
International Computer Music Association, pp214-221. 1991.
- [Kram94] G. Kramer [Editor], “Auditory Display: Sonification, Audification, and  
Auditory Interfaces.”  
Santa Fe Proceedings Vol. **XVIII**, pp185-221, 1994.
- [Lust80] M. Lustig, “Music Editing for Motion Pictures”.  
Hastings House, NY. 1980.
- [Lyt191] W. Lytle. “More Bells & Whistles.”  
[Video] in *SIGGRAPH'90* film show.  
Also described in *Computer*, Vol. **24**: No. 7, p4. 1991.
- [Matt69] M. Matthews, “The Technology of Computer Music.”  
MIT Press, Cambridge. Mass. 1969.
- [Maye92] G. Mayer-Kress, R. Bargar, and I. Choi,  
“Musical Structures in Data from Chaotic Attractors.”  
*Proc. ICAD'92*. Santa Fe, NM. 1992.

- [Mint85] P. Mintz: "Orson Welles's Use of Sound."  
In "Film Sound: Theory and Practice.", E. Weiss and J. Belton [Eds.]  
Columbia University Press, NY. 1985.
- [Mish95] S. Mishra, and J. Hahn,  
"Mapping Motion to Sound and Music in Comp. Animation & VEs."  
*Proceedings of the Pacific Graphics Conference '95*. S. Korea, 1995.
- [Naka93] J. Nakamura et al.  
"Automatic Background Music Generation based on Actors' Emotions  
and Motions"  
*Proceedings of the Pacific Graphics Conference '93*. S. Korea, 1993.
- [Orr93] J. Orr,  
"Cinema and Modernity"  
Polity, Cambridge, UK. 1993.
- [Patt82] R. R. Patterson,  
"Guidelines for Auditory Warning Systems on Civil Aircraft."  
Paper No. 82017, Civil Aviation Authority, London. UK. 1982.
- [Pope93] S. T. Pope & L. Fehlen,  
"The Use of 3D Audio in a Synthetic Environment."  
*Proc. IEEE VRAIS'93*, pp176-182.
- [Roma87] J. Romano,  
"Computer Simulation of the Acoustics of Large Halls."  
*Journal of Acoustical Engineering*, **11**(2): pp121-129. 1987.
- [Roth92] J. Rothstein,  
"MIDI – A Comprehensive Introduction."  
A-R Editions, Madison. WI. 1992.

- [Scal89] C. Scaletti,  
“The Kyma/Platypus Computer Music Workstation.”  
*Computer Music Journal* **13**(2): pp23-38. 1989.
- [Scal92] C. Scaletti,  
“Sound Synthesis Algorithms for Auditory Data Representations.”  
*Proc. ICAD '92*. Santa Fe, NM. Oct, 1992.
- [Scal93] C. Scaletti,  
“Using Sound to Extract Meaning from Complex Data.”  
*SIGGRAPH'93 Course Notes #81: Intro to Sonification*. 1993.
- [Senj87] M. Senju & K. Ohgushi,  
“How are the Player’s Ideas Conveyed to the Audience?”  
*Music Perception* **4**(4): pp311-324. 1987.
- [Smit90] S. Smith, et al.  
“Stereophonic and Surface Sound Generation for Exploratory Data  
Analysis.”  
*Proc. CHI'90*: pp125-132. 1990.
- [Taka92] T. Takala, J. Hahn, L. Gritz, J. Geigel, & J.W. Lee,  
“Using Physically Based Models and Genetic Algorithms for Functional  
Composition of Sound Signals, Synchronized Animation to Motion.”  
*Proceedings of the 1993 International Computer Music Conference*.  
International Computer Music Association.
- [Verc86] B. Vercoe,  
“CSound: A Manual for the Audio Processing System and Supporting  
Programs.”  
M.I.T. Media Lab, M.I.T, Cambridge, MA. 1986.

- [Walk79] A. Walker,  
“The Shattered Silents: How Talkies Came to Stay.”  
W. Morton. NY, 1979.
- [Wenz88] E. Wenzel,  
“Development of a 3D Auditory Display System.”  
*SIGCHI Bulletin*, **20**: pp52-57. 1988.
- [Wenz90] E. Wenzel,  
“A System for 3D Acoustic Visualization in a VE Workstation.”  
*Proc. IEEE Viz’90*: pp329-337. 1990.
- [Wenz91] E. Wenzel,  
“Localization of Non-Individualized Virtual Acoustic Display Cues.”  
*Proc. CHI’91*: pp351-359. 1991.
- [Wenz92] E. Wenzel,  
“Localization in Virtual Acoustic Displays.”  
*Presence*: Vol. **1**:1, pp80-107. 1992.
- [Wigg93] G. Wiggins, E. Miranda, A. Smaill, & M. Harris.  
“A Framework for the Evaluation of Music Representation Systems.”  
*Computer Music Journal*, **17** (3) pp31-42. 1993.
- [Zaza91] T. Zaza,  
“Audio Design: Sound Recording Techniques for Film and Video.”  
Prentice-Hall, Englewood Cliffs, NJ. 1991.

## Appendix A: *mkmusic* Command-Line Options

Usage: *mkmusic* [-flags]

Legal flags are:-

- h** Displays this help message.
- i**<filename> Data-input filename.
- o**<filename> Score-output filename.
- D** Use default values for: **-e -s -I -d -p**.
- N**<filtermode> Set data filter mode.  
(Options: custom, cutoff, kick, bound, none.)
- R**<renderer> Set output renderer mode.  
(Options: song, score, csound, scube, vas, afg, midi.)
- p**<pts>[+i1+...] Number of parts in CSOUND score.  
(Optional: Use +<i1>+... to specify part indices.)
- f**<framerate> Set sync rate (default = 30.0).
- S**<sr>[+sr2+...] Set data sampling rate (default = 10).  
(Optional: Use +<sr2>+... to specify part sampling rates.)
- e**<emotion> Emotional emphasis.  
(Options: happy, sad, bright, evil, peaceful, any, none.)
- s**<scale> Musical scale.  
(Options: major, minor, blues, dom, dim, hdim, any, none.)
- I**<instrument> (Recommend) instrumentation.  
(Options: solo, band, orch, csound, any, none.)
- d**<frames> Duration of score in frames.

<b>-v</b> <level>	Set volume level (1-9).
<b>-b</b> <bound>	Limit of listener world.
<b>-c</b>	Output in CSound score format.
<b>-t</b>	Throughput data from <u>stdin</u> .
<b>-C</b>	Same as setting <b>-t</b> and <b>-C</b> (i.e. Pipe <u>stdin</u> to CSound.)
<b>-P</b>	Set if piping output directly to CSound.
<b>-T</b>	Same as setting <b>-t</b> , <b>-c</b> and <b>-P</b> (i.e. Pipe in-> and out->).
<b>-A</b>	Same as setting <b>-c</b> and <b>-P</b> .



## Appendix B: Custom Data-Map Examples

```
int ChimeMap(float dir, float dis, float st, float length, float longest)
{
    return ( (int) (dir * dis * st * (length/longest)));
}
```

```
int GenMap(float n, int d)
{
    int val = (int) (1000 * n);

    return (val % d);
}
```

```
int TexMap(float val)
{
    int i;

    val = val / 10;

    for (i = 0; i < 2; i++)
        val = (val * val) + i;

    return((int) val);
}
```

```
int LuxMap(int dir, float dis, float base)
{
    int val;
    float scale = 3.5;

    if (dir == 1)
        val = (int) (dir * dis * base);
    else
        val = (int) (st + dir * dis * base * (scale * drand48( )));

    return(val);
}
```

```
int MboxPosMap(float indist, float scale)
{
    return((int) (indist * scale));
}
```

```
int MboxOrientMap(float rot_angle, float modval)
{
    return((int) (rot_angle % modval));
}
```

```
int DefaultMap(float inval)
{
    float tmp;
    int t;

    tmp = (inval + 1) * 10000;

    t = (int) tmp;

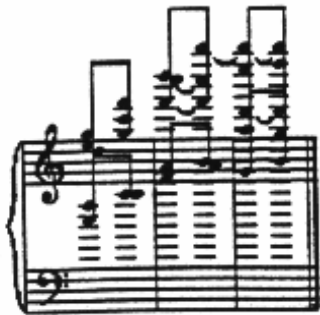
    return(t % 12);
}
```

```
int ScrapeMap(float inval)
{
    return(DefaultMap(inval));
}
```

Appendix C: Musical Score Examples

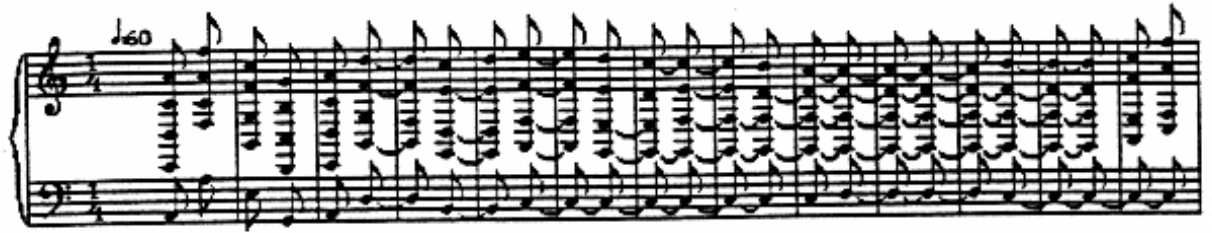
**Windchimes 1994**

Automatically generated from animation motion-control data.



# Windchimes 1995

Composition from modified Windchimes animation, from 1995.



## The Musicbox: Bound Filtering Composition

Based on the animation: "The Musicbox," © 1995, Robert W. Lindeman.

The image displays three staves of musical notation, each consisting of a treble and bass clef staff. The first staff begins with a tempo marking of *J. 180*. The notation is a piano accompaniment for a piece in 3/4 time. The melody in the treble clef is characterized by a sequence of eighth and quarter notes, often with slurs. The bass clef provides a steady accompaniment with chords and single notes. The three staves represent a continuous sequence of music, with the first staff containing measures 1-4, the second staff measures 5-8, and the third staff measures 9-12.

## The Musicbox: Normal Filtering

Default data filtering and composition for Musicbox data.



# The Musicbox: Kick Filtering

The image displays a handwritten musical score for a piece titled "The Musicbox: Kick Filtering". The score is organized into four systems, each consisting of three staves labeled "part1", "part2", and "part4".

- System 1:** Part 1 features a melodic line with eighth and sixteenth notes. Part 2 provides a harmonic accompaniment with sustained notes and some rhythmic movement. Part 4 contains a bass line with a prominent kick drum pattern.
- System 2:** Part 1 continues the melodic theme. Part 2 adds more complex rhythmic patterns. Part 4 maintains the bass line with a consistent kick drum presence.
- System 3:** Part 1 shows a more active melodic line. Part 2 has a dense texture with many notes. Part 4 features a more intricate bass line with a kick drum.
- System 4:** Part 1 has a steady melodic flow. Part 2 continues with a dense, rhythmic accompaniment. Part 4 has a complex bass line with a kick drum.

The notation is handwritten and includes various musical symbols such as notes, rests, and stems. The overall style is that of a working draft or a composer's sketch.

part1  
part2  
part3

This section of the musical score consists of three staves. The top staff, labeled 'part1', contains a melodic line with several slurs and accents. The middle staff, labeled 'part2', is mostly empty with some faint markings. The bottom staff, labeled 'part3', contains a complex rhythmic accompaniment with many beamed notes and rests.

part4  
part5  
part6

This section of the musical score consists of three staves. The top staff, labeled 'part4', has a few notes at the beginning. The middle staff, labeled 'part5', is empty. The bottom staff, labeled 'part6', contains a melodic line with two large slurs covering the first two measures.



# The Musicbox: Kicked and Slowed Filtering

The image displays a handwritten musical score for a piece titled "The Musicbox: Kicked and Slowed Filtering". The score is organized into three systems, each containing four staves labeled part2, part4, part5, and part7. The notation is written in black ink on white paper. Each staff begins with a clef (soprano, alto, or bass) and a key signature of one flat. The music features a variety of note values, including quarter, eighth, and sixteenth notes, as well as rests. Slurs and ties are used extensively to connect notes across measures. The first system shows parts 2 and 4 in the upper staves and parts 5 and 7 in the lower staves. The second system continues this arrangement. The third system shows a different grouping, with parts 2 and 4 in the upper staves and parts 5 and 7 in the lower staves. The overall style is that of a personal manuscript or a working draft.

part2

part6

part5

part7

A musical score system with four staves. The top staff (part2) is in bass clef and contains a melodic line with slurs and ties. The second staff (part6) is in treble clef and contains a melodic line with slurs and ties. The third staff (part5) is in bass clef and contains a melodic line with slurs and ties. The bottom staff (part7) is in treble clef and contains a melodic line with slurs and ties.

part2

part6

part5

part7

A musical score system with four staves. The top staff (part2) is in bass clef and contains a melodic line with slurs and ties. The second staff (part6) is in treble clef and contains a melodic line with slurs and ties. The third staff (part5) is in bass clef and contains a melodic line with slurs and ties. The bottom staff (part7) is in treble clef and contains a melodic line with slurs and ties.

part2

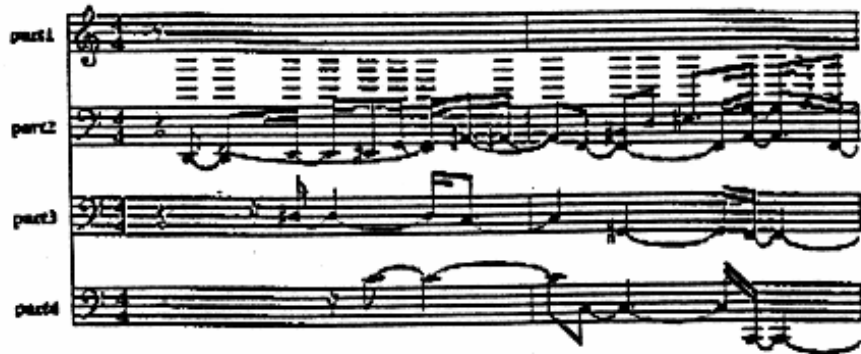
part6

part5

part7

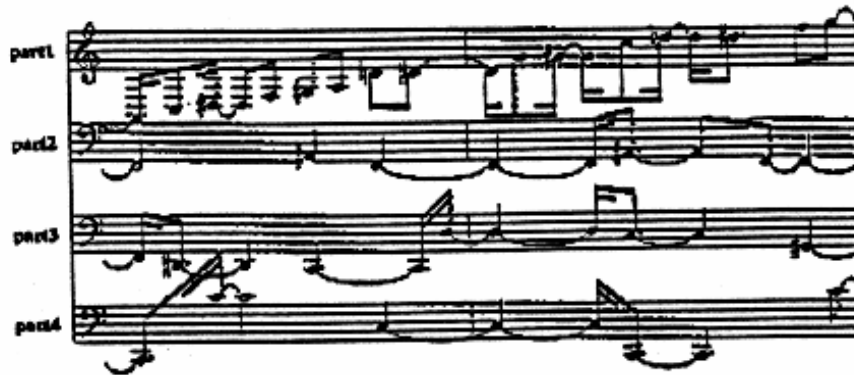
A musical score system with four staves. The top staff (part2) is in bass clef and contains a melodic line with slurs and ties. The second staff (part6) is in treble clef and contains a melodic line with slurs and ties. The third staff (part5) is in bass clef and contains a melodic line with slurs and ties. The bottom staff (part7) is in treble clef and contains a melodic line with slurs and ties.

# The Musicbox: Custom Data Filtering Applied



part1  
part2  
part3  
part4

This system shows the first four parts of a musical score. Part 1 is in treble clef and contains a series of vertical bars. Parts 2, 3, and 4 are in bass clef and contain melodic lines with various note values and rests.



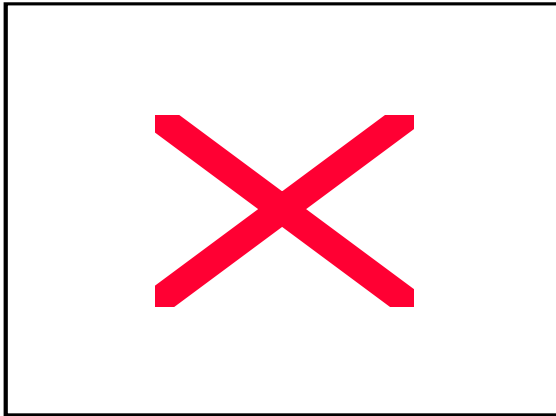
part1  
part2  
part3  
part4

This system shows the second four parts of a musical score. Part 1 is in treble clef and contains a melodic line. Parts 2, 3, and 4 are in bass clef and contain melodic lines with various note values and rests.



part1  
part2  
part3  
part4

This system shows the third four parts of a musical score. Part 1 is in treble clef and contains a melodic line. Parts 2, 3, and 4 are in bass clef and contain melodic lines with various note values and rests.



## Morphing Animation: Sonification (Partial)

Direct sonification of data-points in a 3D-morphing animation.

