MAPPING MOTION TO SOUND AND MUSIC IN COMPUTER ANIMATION AND VE

SUNEIL MISHRA and JAMES K. HAHN

Department of Electrical Engineering and Computer Science
The George Washington University
Washington, DC 20052, USA
suneil | hahn@seas.gwu.edu

Abstract

Sounds are generally associated with motion events in the real world. Therefore, there is an intimate linkage between the two. In most cases, however, motions and sounds in computer animations and virtual environments (VE) are treated separately. In this paper, we present a methodology to map sounds to motions. In particular, we present a system designed for soundtrack production which seeks to bring musical accompaniments within the reach of animators as well as for live performance using VE peripherals. The creative compositional process involves filtering the data being generated by the motion control system and mapping these values to musical constructs. A simple UI as well as animator-programmable filters allow flexible control over the nature of the music produced. The system's rendering can be in the form of common musical scores (e.g. CSOUND, MIDI), or as real-time performance of musical pieces. Using this methodology, we have produced aesthetically pleasing, appropriate music and sound effects for a variety of animated sequences, with minimal animator expense.

1 Introduction

In the real world, sounds are generated by physical interactions between objects and the environment they reside in. The characteristics of the sound are defined by the motions or events causing them, as well as the properties of the objects themselves. In computer animations and virtual environments, there has historically been a separation between motion and sounds, since most emphasis has been on the visual domain. Sounds are attached to simulations with minimal regard to synchronization or appropriateness to the motion events causing them. In this paper, we describe a methodology that maps motions to sounds, and provides a more intimate relationship between the visual and aural domains. In particular, we demonstrate the technique by creating musical soundtracks both for animations and virtual environments, which closely relate the motions of objects with the musical accompaniment composed.

In generating a soundtrack for an animation, an animator wishes to reinforce some visual event or motion through the use of the aural medium. With realistic sound effects,

this may take the form of discrete, synchronized sounds linked to events in a simulation (such as collisions), or continuous sounds affected as they move through an environment (such as Doppler shifting of a siren as a moving source passes by). Such effects are meant to provide information to the viewer outside the visual medium. Musical accompaniments are more often to emphasize or invoke certain moods or emotions underlying an animation. In this sense, the sounds produced are more loosely defined, but can be more powerful than realistic sound effects or visuals alone in conveying a sense of time, place, or mood [Gorb87]. Much of the theory of soundtrack production has come from the related visual fields of film, television, and theater. While many techniques are common to all, computer creation has its own idiosyncrasies, as well as many conveniences that have not been sufficiently exploited thus far.

Traditional approaches to soundtrack production have concentrated on pre- or post-processing methods [Zaza91] with reference to motion events. In the pre-processing technique, a soundtrack is composed early in development, before motions have been finalized. Using timing information from the soundtrack, motion events are then synchronized to sounds. A major drawback of this approach is that it forces animation visuals to be driven by the soundtrack. Animators in general would prefer control to remain outside the aural domain. 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 then done manually in a process known as 'Foleying.' A deficiency to this approach is that changes to the visuals force a regeneration of the soundtrack, which can be time-consuming and expensive. This 'Foleying' process has recently been automated to some extent by several soundtrack systems. The Background Music Generator (BGM) [Naka93] allows animator control over musical generation for animations. This is similar in approach to our system. Several deficiencies exist with the BGM system, that we seek to address. First, the synchronization is manually specified, rather than dynamically bound to the motion control system. Second, the music generated does not depend on the motion control program, making links between motion and sound arbitrary. Instead, high-level emotional control constructs are provided by the user, and music is then created from a database of possible melodies. Simple, sample-based sound effects are also possible. The technique is limited due to its reliance on a database of pre-programmed musical motifs, and purely sample-based realistic effects.

In addition to the animation soundtrack systems described above, much of the research done has been in the field of computer music. Although this research concentrates mainly on sound synthesis and specifically musical applications, much of the methodology is general enough for application to computer animation requirements. The Music-N family of synthesis systems [Road85] developed the concept of parameterizable sound structures, and hierarchical modeling of soundtracks. Most later systems use a similar methodology, including CSOUND [Verc86], Kyma/Platypus [Scal91], and Fugue [Dann91]. This parameterization allowed internal control of a sound by the designer, as well as abstracting

the conceptual *instrument* (sound) from the *score*, which scripts how and when instruments should play. The problem lies in the creation of such parameterized sound structures, and in mapping from the parameter space of the motion domain to that of the sound space. As computer music researchers are concerned only with music and sound synthesis, they are unaware of the available motion control parameters for a given simulation. Similarly, animators are unfamiliar with the technical aspects of sound design. This gulf of understanding is a major problem in combining these related fields.

We have developed a system [Taka92] that aims to address some of the problems, and attempts to also provide real-time performance for use in VEs [Hahn95b]. In this paper, we present the *mkmusic* system which extends this to the domain of music production. The system emphasizes the motion to sound mapping, dynamically binding synchronization-to-motion as a concurrent processing step.

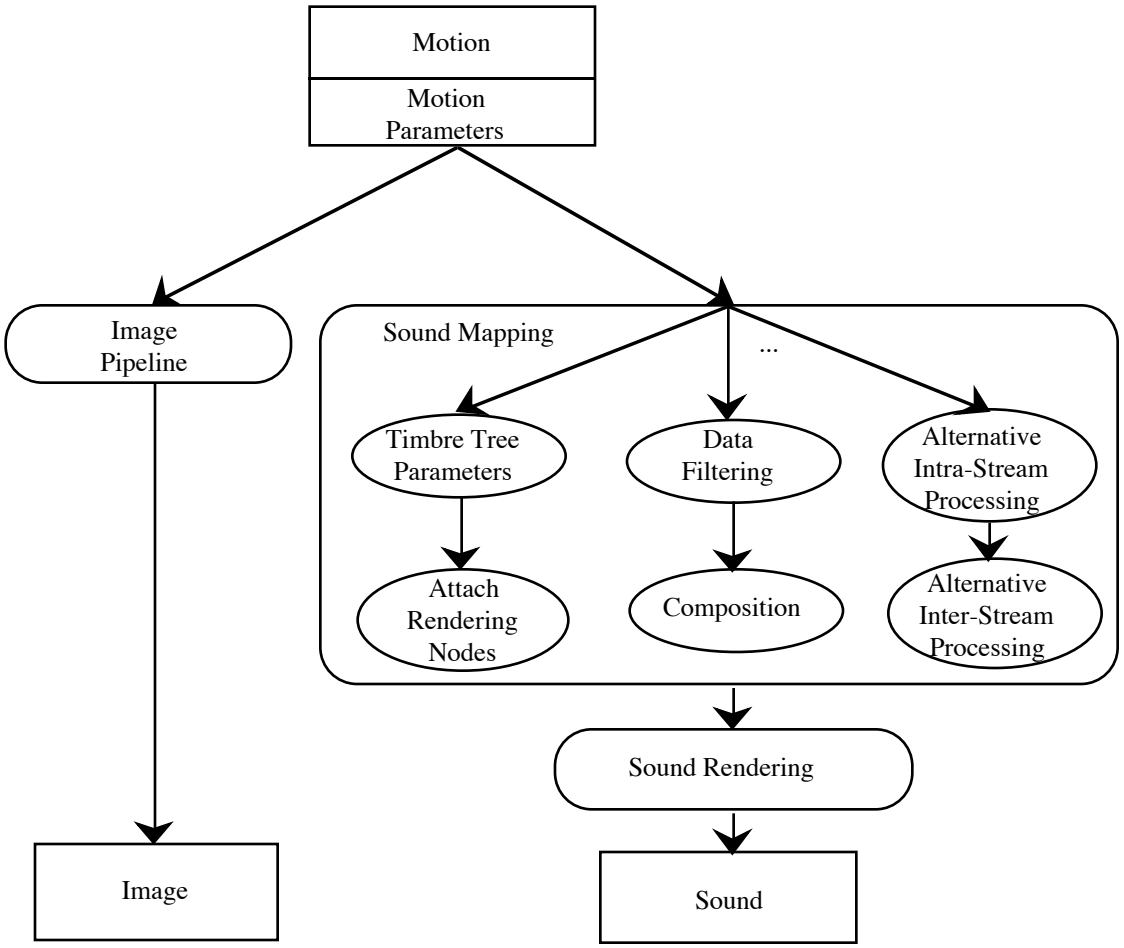

Figure 1.1: Integrating sound and motion.

Figure 1.1 shows the binding between image and sound production pipelines and the motion world. In each case common motion data is used to generate synchronized and dynamically bound visual and aural output. The sound pipeline consists of a mapping process in which motion parameters are tied to sound parameters. The exact method used depends on the sound structure being used (for example *timbre-trees* [Hahn95a], or musical constructs). Essentially two sets of mappings are applied: local, intra-stream mappings which act internally to the structure representing a sound, and inter-stream mappings that attach global constructs to bind streams together. The independence of these processes from rendering allows a variety of representations and mapping schemas to be used, without affecting the final performance of a soundtrack or musical score.

Our system aims to be simple to use, as its target user-group are non-specialized in the auditory field, but still powerful enough to meet specific requirements in aural generation.

2 System Overview

Although the *mkmusic* soundtrack production system is designed to produce musical soundtracks for computer animations, the methodology used is general in nature. This allows application to realistic (non-musical) sound effects, and real-time performance for use in virtual environments.

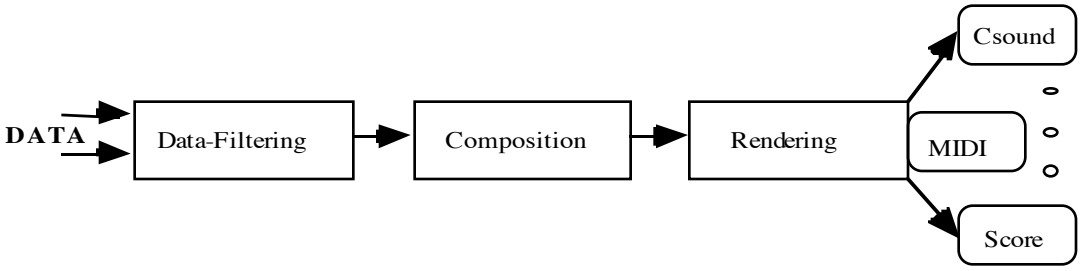

Figure 2.1: *Mkmusic* Production Pipeline.

Figure 2.1 shows the production pipeline for soundtrack generation using the *mkmusic* system. Data is supplied from an external source (such as a motion control program), to the system and this data is then filtered, composed and sent for rendering. At the output stage of this pipeline, external renderers are supported, although direct MIDI rendering is also available internal to the system, both as a production and development utility.

The system links motions to sounds by the use of common data for both visual and auditory display. Synchronization between visuals and sounds is thus inherently maintained, and dynamically bound, so that any changes in motion result not only in corresponding adjustments to visuals, but to the soundtrack as well. The *mkmusic* system operates by first taking the input data stream and filtering it. This *data-filtering process* provides *application-context* to the data, as the system assumes no such context on entry. This filtering process is most important in binding the motions to sounds effectively, and is expected to be user-specified. The filtering methodology is described in more detail in **Section 3**. The filtered data values are then passed to the *composition process*. This allows several concurrent output sound streams to be constrained to provide *auditory-context*. On input to the *composition* module, each sound stream, or musical part is independent of all others. Since each part shares a common producer, there should be some commonalities in the output produced. This may take the form of attaching global environmental effects, such as reverberation due to an enclosed space within which the sonic scene is defined, or musical compositional rules. At this stage, such effects are merely encoded in a suitable format for the renderer chosen. The composition process is described in more detail in **Section 4**. Finally, the composed auditory streams are sent to the *rendering process* for output. Here any global constructs such as environmental effects, as well as the actual sound events are realized. How these effects are processed (or if at all) is dependent on the renderer used. Several renderers are supported, both for musical, and for sound-effects production, including MIDI, CSOUND, and our own *timbre-tree* based renderer. The rendering process will be described in **Section 5**.

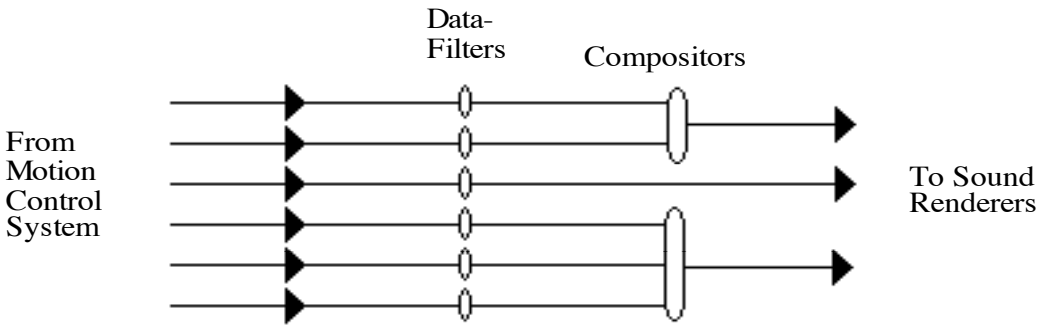

Figure 2.2: Stream Processing.

Figure 2.2 shows the data-filters and stream compositors and their relationship to the data streams through the *mkmusic* system. The data-filtering can be seen to be an *intra-stream* process, where each stream is internally modified. The *composition* module applies *inter-stream* mappings to bind them into a cohesive soundtrack on output. These two processes therefore provide local and global control over the production of the soundtrack.

3 Data-Filtering

The data input stream to *mkmusic* is processed by the *data-filtering module*. It is the responsibility of this module to define the binding between motion and sound parameters. As the input is simply a data-stream, the specific filters applied must be selected (and preferably designed) by the user. The exact nature of the filters and their effect on the data stream are often not straightforward. This mapping problem is merely a more formal, implementational version of the one facing all soundtrack designers: how should the sound correspond to the motion events at the current time. The filtering process is the bridge between the motion world and sound world. How particular motion parameters correspond to parameters of a sound object is often difficult to ascertain. This is the case both for animators, and for sound designers, because of the gulf of understanding between the two fields. However, the flexibility in design of filters, and the interactive rate of production allow this filter-design process to proceed incrementally, until a satisfactory result is obtained. The filters are specified by C-language function calls, which act upon the data inputs, and output as the user chooses. One final system mapping converts all filtered values into the format expected by the composition process (namely into range-limited integer values). This allows users to define libraries of intermediate filters which they can compose together to create the effect they desire.

There are three factors in 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.

3.1 *Motion Parameters*

Parameters available from the motion control system depend greatly on the sophistication and abstraction available in the motion control schema used. Kinematic techniques (e.g. keyframing) 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. However, it is completely the animator's decision which data, 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 color 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. We have found that for musical score generation, at least three output parts are desirable if the music is to create and maintain an interesting sound. If less 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.

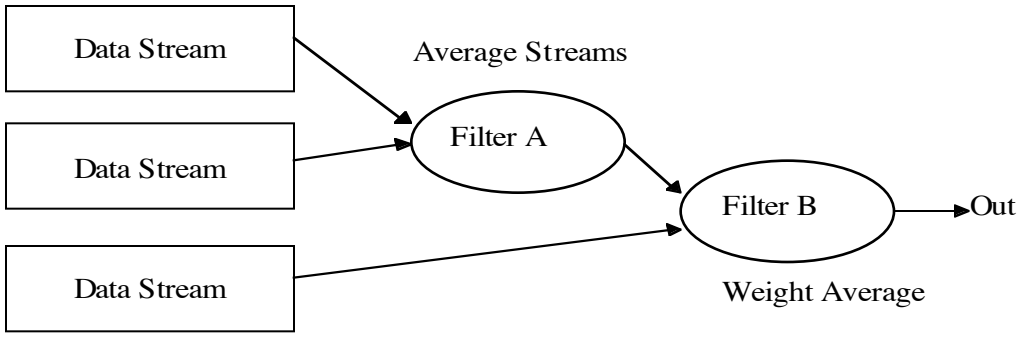

Figure 3.1: Merging data streams to produce reduced filtered outputs.

Figure 3.1 shows such a merging of streams at disjoint intervals. Filter A combines the values in the first two data streams by averaging their added values (A 2D-vector magnitude operation). This averaged stream is then weighted by values from the third, unfiltered stream.

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. We have found two strategies that 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 datasets.

3.2 *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 our 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 is part of the anticipated future work on the system.

3.3 *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 have succeeded in creating aesthetically pleasing soundtracks. Even simple linear mappings 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 3.2a shows frames from the animation; Figure 3.2b, example coding for the filter functions and Figure 3.2c, the five-part score generated.

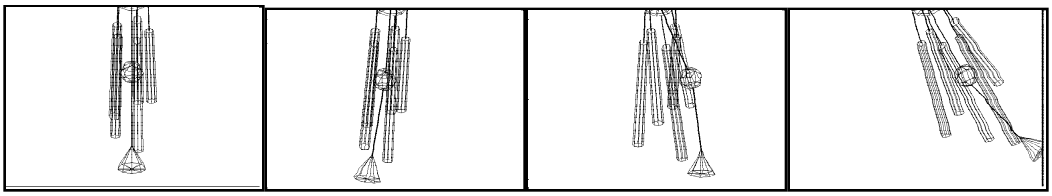

Figure 3.2a: Frames from windchimes animation.

```
musical_note = ChimeMap ( xpos, ypos, zpos, c_len, c_long );
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 3.2b: Filter code extract for windchimes.

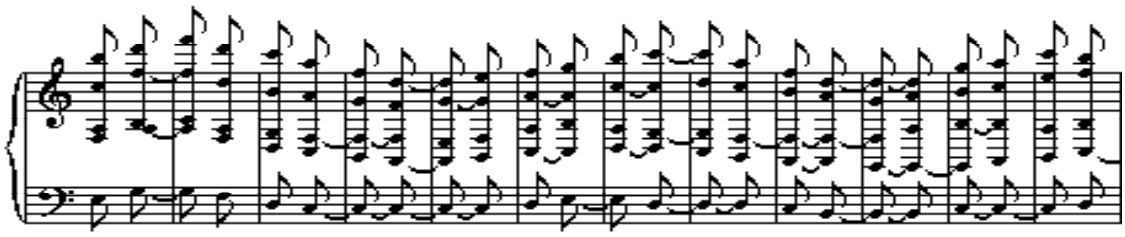

Figure 3.2c: Musical score generated from windchimes animation.

4 Composition Process

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 the 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.

Figure 4.1 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 color 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.

Emotion	Scale
Happy	Major
Sad	Minor
Evil	No Constraint

Figure 4.1a: Emotion to Scale mapping.

Emotion	Note Length
Bright	Short; Staccato
Peaceful	Longer; Flowing.

Figure 4.1b: Emotion to Note Length mapping.

The score in Figure 4.2 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.

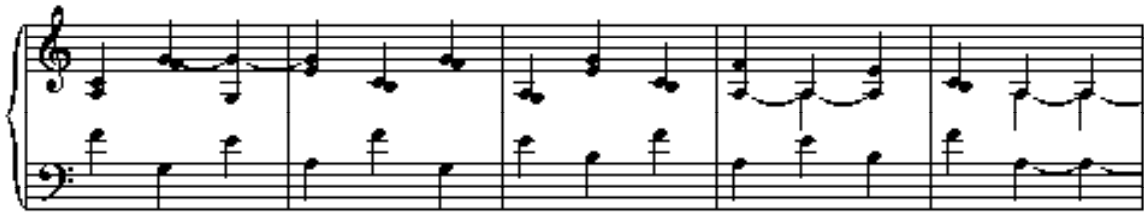

Figure 4.2: Score extract from an animation of waltzing figures.

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 lended to the rhythmic tempo of the underlying bass part (the lowest score part in the extract). 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 center 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.

5 Rendering Process

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 being interfaced to, including MIDI and Csound renderers, which together encompass much of the sound research community's rendering capabilities. Additionally, notated musical scores can be produced using third party applications. In an ideal world, such composed scores would be "rendered" by a live orchestra, with these notated scores provided to the musicians. This is beyond the scope of most animation productions, however, so renderers of a more accessible nature remain the mainstay 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 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 renderer formats rather than develop its own independent one. This should be left to sound experts rather than computer graphics and animation researchers.

6 Real-time Performance and Virtual Environments

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.

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 rendering. 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. Figure 6.1 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 timestamp, duration, amplitude, and frequency (note-value) for the instrument. Each line thus represents a single note in the performed ‘score.’

```
i1      0.000     0.500      5727      123.472
i1      0.500     0.500     15506     1046.528
i1      1.000     1.000     15991      880.000
i1      2.000     2.000     14627      440.000
e
```

Figure 6.1: CSOUND score extract.

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.

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.

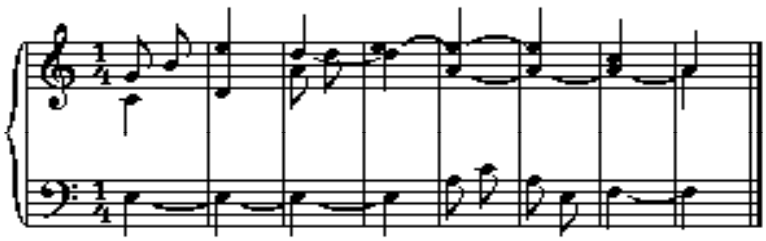

Figure 6.2: VE composed score.

Figure 6.2 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 attentuation through MIDI velocity controls to left and right channels on the synthesizer was also performed.

7 Conclusion

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

soundtrack production needs. The system is versatile enough in design to be used as a data sonifier, or 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 objects within them. A correspondence between the motion and sound domains exists naturally in the real world. Sounds are physically generated due to the motions of objects within an environment. Our system exploits this 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. With minimal user expense, we have created aesthetically pleasing musical compositions, appropriate and unique to the motions they reflect.

8 Acknowledgements

Our thanks to the members of the Computer Graphics Group at the George Washington University, in particular to the Sound Research Group: Hesham Fouad, Joe Geigel, Jong Won Lee, and to Robert Lindeman and Larry Gritz for providing example animation data.

9 References

[Dann91] Dannenberg, R., C.Fraley, and P.Velikonj (1991). "Fugue: A Functional Language for Sound Synthesis", IEEE Computer, Vol.24, No.7, pp36-42.

[Gorb87] Gorbman, C. (1987), "Unheard Melodies". Indiana University Press: Bloomington, Indiana.

[Hahn88] Hahn, J. "Realistic Animation of Rigid Bodies," *Proc. SIGGRAPH'88*, *ACM Computer Graphics*, Vol. 22, No. 3, pp299-308.

[Hahn95a] Hahn, J., J.Geigel, L.Gritz, J.Lee, and S.Mishra."An Integrated Approach to Motion and Sound." To appear in the Journal of Visualization and Computer Applications, 1995.

[Hahn95b] Hahn, J., H.Fouad, L.Gritz, J.Lee. (1995). "Integrating Sounds and Motions in Virtual Environments." To appear in Presence, 1995.

[Naka93] Nakamura, J., T.Kaku, T.Noma, and S.Yoshida. "Automatic Background Music Generation Based on Actors' Emotion and Motions". Proceedings of Pacific Graphics'93, Vol.1, pp147-161.

[Road85] Roads, C. and J.Strawn (1985). "Foundations of Computer Music". MIT Press: Cambridge, Mass.

[Scal91] Scaletti, C. “The Kyma/Platypus Computer Music Workstation” in “The Well-Tempered Object: Musical Applications of Object Oriented Software Technology.” S.T.Pope, editor. MIT Press (1991).

[Taka92] Takala, T. and J.K.Hahn. (1992). “Sound Rendering”, Proc. SIGGRAPH’92, ACM Computer Graphics, Vol.26,No.2, pp211-220.

[Taka93] Takala, T., J.Hahn, L.Gritz, J.Geigel, and J.Lee. “Using Physically-Based Models and Genetic Algorithms for Functional Composition of Sound Signals Synchronized to Animated Motion.” Int. Computer Music Conference’93, Tokyo, Japan.

[Verc86] Vercoe, B. “Csound: A Manual for the Audio Processing System and Supporting Programs.” M.I.T. Media Lab, M.I.T., Cambridge, Ma. (1986)

[Zaza91] Zaza, T. (1991) “Audio Design: Sound Recording Techniques for Film and Video.” Prentice-Hall.