Toward a Framework for Interactive Systems to Conduct Digital Audio and Video Streams

Eric Lee, Thorsten Karrer, and Jan Borchers
Media Computing Group
RWTH Aachen University
52056 Aachen, Germany
{eric, karrer, borchers}@cs.rwth-aachen.de

Following advances in commodity hardware and computing power, interactive conducting systems have grown in complexity and capability. Modern conducting systems incorporate research from a variety of disciplines, from motion tracking, to gesture recognition and interpretation, to digital signal processing. Frameworks have emerged in recent years to enable rapid development of such complex systems, including Max/MSP for manipulating and processing synthesized and sampled audio (Puckette 2002), and EyesWeb for gesture tracking (Camurri et al. 2003). Today's computers are, moreover, able to handle large chains of complex filters and other operations on digitally sampled audio and video streams in real time.

In contrast, modern computer music systems often do not take full advantage of these capabilities by continuing to use synthesized music, usually MIDI-based; even less incorporate video. The advantage of using synthesized music over digitally sampled audio streams is a higher level of semantic access to the data, such as beats, notes and voicings. However, digital audio and video recordings can offer a higher level of fidelity and realism: today's synthesizing technology is still unable to reproduce, for example, the unique character of the Vienna Philharmonic playing in their Golden Hall of Vienna's Musikverein. Part of the problem can be attributed to the difficulty of working with time-based effects in current multimedia frameworks such as Apple's QuickTime (http://apple.com), Microsoft's DirectShow/DirectSound (http://microsoft.com), or Max/MSP.

We have encountered some of these difficulties in our own work, which includes a series of interactive conducting systems that incorporate audio and video recordings. These systems have been well-received as museum exhibits around the world. *Personal Orchestra*, which was coordinated by Max Mühlhäuser, now at Darmstadt University, was installed in the HOUSE OF MUSIC in Vienna in 2000 (Borchers et al. 2004), and is the first system to use audio and video recordings in an interactive conducting system. It switched between a set of preprocessed audio tracks to allow variable speed audio playback without pitch-shifting artifacts. *You're the Conductor*, our follow-up system for the Boston Children's Museum in 2003 (Lee et al. 2004) in collaboration with Teresa Marrin Nakra at Immersion Music, featured a better audio/video rendering engine capable of time-stretching audio in real time. Most recently, we created a “hybrid” Personal Orchestra which incorporates the gesture recognition of Personal Orchestra with the audio/video rendering engine of You're the Conductor.

Our conducting systems allow the user to control various aspects of the musical recording using simple conducting gestures, including tempo, dynamics, and instrument emphasis. We have found that users unfamiliar with our systems, as is usually the case in a museum, most readily grasp the temporal interaction of changing the tempo; children, in particular, enjoy

pushing the limits of how quickly or slowly they can make the orchestra play. In an evaluation session where we silently observed users interacting with Personal Orchestra and then interviewed them, 93% of the users realized that they could control tempo by moving the baton faster or slower, 77% realized that they could control volume by making larger or smaller gestures with the baton, and 37% realized that they could control the instrument emphasis by conducting to different sections of the orchestra shown on the large display (Borchers et al. 2004).

Allowing the user to freely manipulate the tempo of recorded audio and video poses a number of interesting research challenges. In this article, we focus on the specific challenges of this temporal interaction, especially when video accompanies the audio, and our solutions. We begin by describing our current system design and its individual components; this design, which more generally describes our three conducting systems, incorporates our recent research on interpreting conducting gestures and audio time-stretching. We then discuss how this design has become unnecessarily complex as improvements over previous designs were made. This complexity motivated us to design and prototype a new software framework for digitally sampled audio and video streams; this framework adopts a more general and intuitive model of time that we believe is applicable to a wider class of computer music systems. We show how this framework helped simplify the design of our conducting systems. The two contributions of this article are the framework with its proposed model of time, and our improved audio time stretching algorithm; the discussion surrounding our conducting systems provides a context for motivating and describing these contributions.

Terminology

The use and scope of terms such as *synthesized audio* and *digitally sampled audio streams* has evolved with technology over the years. We will clarify their usage in this article in an attempt to minimize confusion with other uses of the same term.

Synthesized audio exploits the quasi-periodic nature of musical audio signals to reduce the amount of information required to represent a musical piece digitally. MIDI is often employed to store and process these music semantics, which are in turn passed to a synthesizer that renders the audio output. Modern synthesizers often use a number of short snippets of recorded audio, *sound samples*, from real instruments to recreate more realistic sounds. The general class of synthesizers which employ this technique are often referred to as *wavetable synthesizers*.

Digitally sampled audio streams, also known as pulse code modulation (PCM) audio, are sequences of numerical values obtained by sampling an analog audio signal at a constant sampling rate; the sampling rate is usually determined based on the Nyquist theorem. Each numerical value in these sequences is also called a *sample*, although this should not be confused with the "sound samples" used in wavetable synthesis; in this article, the term "sample" will always refer to a numerical value in a digital audio stream. Working with digitally sampled audio streams typically requires significantly more processing power and storage space than synthesized audio, although audio streams can offer higher realism, as they are able to capture arbitrary real-world sounds.

Related Work

Mathews' Radio Baton (1991) was one of the first systems that allowed a user to control computer music using a baton and conducting gestures; conducting gestures were determined through the movement of one or more batons emitting radio frequency signals above a flat receiver panel. These gestures were used to control the playback of a MIDI file. In personal communication with the authors in May 2005, Mathews related that a version of the Radio Baton system was turned into an exhibit at the Children's Discovery Museum in San Jose, USA in 1995.

Since Mathews' work on the Radio Baton, a number of conducting systems have been developed, including Morita et. al's conducting system (1991), Marrin's Conductor's Jacket (2000), Realtime Music Solutions' Sinfonia (http://rms.biz), and Usa and Mochida's Multimodal Conducting Simulator (1998); all of these systems featured improvements in gesture recognition and/or output quality. Kolesnik (2004) has compiled an extensive list in his Master's thesis, and we refer the reader to this work for a more complete and detailed discussion than is possible here. We will instead briefly outline only those systems that incorporate digital video and/or sampled audio streams, as these are the most relevant for this discussion.

Ilmonen and Takala's (1999) conducting system used artificial neural networks, and was perhaps the first conducting system to include video in addition to audio. However, audio was synthesized using MIDI, and the video consisted of artificially rendered 3D avatars.

Murphy et al. (2003) created a system that incorporated recorded audio time-stretched in real time using a variant of the phase vocoder algorithm. They did not include video. Audio was processed using Mixxx (Andersen 2003), an open source digital DJ system.

Most recently, Kolesnik's (2004) Master's thesis work on a conducting recognition, analysis and performance system incorporated digital audio that was also time-stretched using a phase vocoder variant. There was an option to provide accompanying video output; however, the video playback was adjusted independently of the audio, and thus, synchronous audio and video playback was not guaranteed. The audio and video rendering modules were implemented in Max/MSP, and in his thesis he also described some workarounds to the challenges he encountered while trying to incorporate a real time phase vocoder module in Max/MSP.

In contrast to the above research, which focus primarily on new methods for recognizing conducting gestures, our work considers conducting as a natural metaphor for improving users' interaction with computer music. While a majority of our users may have musical experience, they are not necessarily professional conductors. Thus, our systems incorporate techniques for offering high quality, time-stretched audio synchronized with video of an orchestra to increase immersiveness.

Design of an Interactive Conducting System

Figure 1 shows a general block diagram for all of our conducting systems; each module will be detailed in the following sections.

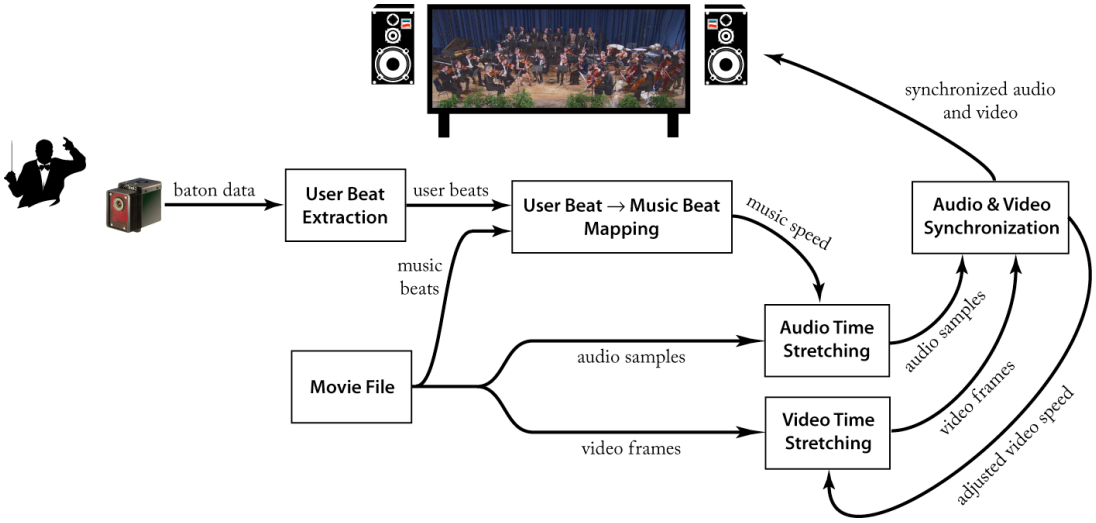

Figure 1: Block diagram of our interactive conducting systems.

User Beat Extraction

The first task after receiving positional information from the baton, such as (x,y) coordinates, is to determine where the user has marked her beats.

The problem of extracting beats from user gestures has been discussed in literature previously. Numerous approaches have been proposed, ranging from simple minima/maxima analysis of the baton's path (Borchers et al. 2004; Kolesnik 2004), to more complex systems utilizing artificial neural networks (Lee et al. 1992; Ilmonen and Takala 1999), or Hidden Markov Models (Usa and Mochida 1998).

Personal Orchestra recognized only simple up and down gestures, and a simple turning point analysis of the gestures was sufficient to determine the user's beats. You're the Conductor, which was primarily targeted towards children and their parents in a children's museum, utilized a simpler gesture recognition scheme which allowed users control over the music speed without the need for absolute beat-level precision. More details on the exact algorithm can be found in (Lee et al. 2004).

User to Music Beat Mapping

The speed of the musical recording must be continuously adjusted as the user marks beats with the baton, so that the orchestra is following the conductor. In Personal Orchestra, this speed calculation also ensures the beats in the musical recording remain aligned with the user's beats. When a user marks a beat, part of the last music beat may not have been played yet, and instantaneously adjusting the music position would result in audible skips in the music. A more realistic approach is to gradually adjust the speed of the music to match the user input (see Figure 2). Our algorithm calculates an adjusted movie speed such that the music beat and the gesture beat will be synchronized after a certain time interval, Δt, based on the following relation:

$$v_u = \frac{T_1 - T_0 + \Delta T}{t_1 - t_0 + \Delta T} \quad (1)$$

The lower case variables denote "real time" and upper case variables denote the movie position, or "movie time". At time t_0, the user increases his conducting speed, v_u, and marks another beat at time t_1; the movie, however, has only advanced to position T_1. The movie speed, $v_m = \frac{\Delta T}{\Delta t}$, is calculated such that the user beats are synchronized with the movie beats at time $t_1+\Delta t$. Substituting $\Delta T = v_m \Delta t$ into (1) and solving for v_m yields:

$$v_m = \frac{v_u + (T_0 - T_1)(t_1 - t_0 + \Delta t)}{\Delta t(t_1 - t_0 + \Delta t)} \quad (2)$$

A more sophisticated approach to mapping user beats to music beats should also take into consideration whether or not the user is a conductor. Our recent work aimed at better understanding the various mental models of conducting by studying the temporal characteristics of conducting gestures, both for conductors and non-conductors (Lee et al. 2005). We found that while conducting styles vary widely from person to person, and even amongst conductors themselves, where and how consistently a person marks her beat relative to the music beat is sufficient to determine whether or not she is a conductor.

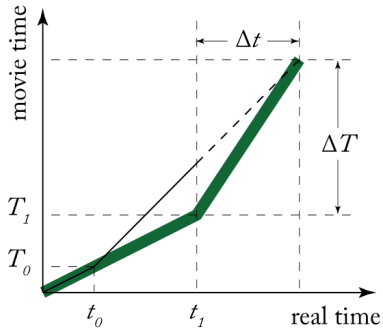

Figure 2: Beat following algorithm. The user increases his conducting speed (thin line) at time t_0, which is recognized by the system at time t_1. At this time, the movie speed (thick line) is increased to $v_m = \frac{\Delta T}{\Delta t}$ to catch up with the user's conducting by time $t_1+\Delta t$.

Our results have interesting implications for the design of interactive conducting systems, particularly for the user beat to music beat mapping. Conductors, unsurprisingly, place their beats precisely and consistently ahead of the music beat. Non-conductors, on the other hand, place their beats much less precisely: sometimes they will lead the music, and sometimes they will follow it. Furthermore, we found that some non-conductors will sometimes mistakenly conduct to the rhythm of the music rather than to the beat. Thus, while conductors would favor a more responsive system, or a small value of Δt in (1), non-conductors would encounter usability issues with the same system; one such issue that we frequently observed is the "spiral of death" where users, in response to a slowdown of the orchestra, slow down their conducting, which causes a further slowdown of the music tempo, and so on.

During the development of You're the Conductor, our collaborator Teresa Nakra found, through informal user tests, that having no beat synchronization did not adversely affect most non-conductors' experience with the system; in fact, most users did not notice that the system did not explicitly synchronize the beat of the music to their beating movements. Instead, they often naturally synchronized their own movements to the beat of the music.

Audio Time-Stretching

Simply changing the speed of an audio recording by resampling results in the well-known pitch-shifting effect, and algorithms for time-stretching audio without changing the pitch have been studied since Gabor's (1946) work on granular synthesis. Time-stretching orchestral music, however, is significantly more challenging than time-stretching other audio, such as speech. Orchestral music is polyphonic, while speech is monophonic. Degradations to time-stretched speech quality are normally tolerated as long as the resulting audio is still intelligible; subtle degradations to time-stretched music quality, in contrast, are less tolerated due to its impact on an otherwise enjoyable experience. Finally, a world-famous orchestra, such as the Vienna Philharmonic, will not tolerate a system that presents its performances with significant audio artifacts. The more complex algorithms used to time-stretch orchestral music have thus not been able to run in real time, until recently.

When Personal Orchestra was developed in 1999, computers were still unable to perform time-stretching in real time with sufficient quality. Thus, we chose to use Prosoniq's Minimum Perceived Loss Time Compression/Expansion (MPEX) algorithm (http://mpex.prosoniq.com). It produced high quality results, but required 32 seconds of processing time per second of audio on an Intel 400 MHz Pentium III computer. To adjust the playback speed of the movie to user input in real time, we prepared a set of audio tracks offline, and then automatically switch to the one that best matches the desired speed while the user is conducting. Unfortunately, this scheme creates problems for audio and video synchronization, and our solutions will be discussed in a later section.

Of course, this scheme is undesirable because the range of playback speeds is limited. Since changes to the speed are discrete, the playback engine is unable to precisely match the desired conducting speed, and audible pops and clicks can be heard when switching between tracks; a short cross-fade when switching between audio tracks helped to minimize these glitches. A final disadvantage is the large amount of overhead required to prepare new movies for this system.

For You're the Conductor, which began development 3 years later, we developed an algorithm to perform the time-stretching in real time; the resulting audio quality was comparable to MPEX for orchestral music. Our algorithm, a variant of the phase vocoder algorithm (Flanagan and Golden 1966), is based on previous work by Laroche and Dolson (1999). The phase vocoder, despite its higher processing requirements and complexity over other algorithms such as granular synthesis, has the advantage that it is capable of producing high quality output over a wide range of stretching factors for polyphonic audio such as orchestral music.

Basic Phase Vocoder

The basic phase vocoder algorithm divides audio data into a series of overlapping win-

dows of *N* samples. Audio duration is expanded and compressed by varying the amount these windows overlap between the input and output (see Figure 3).

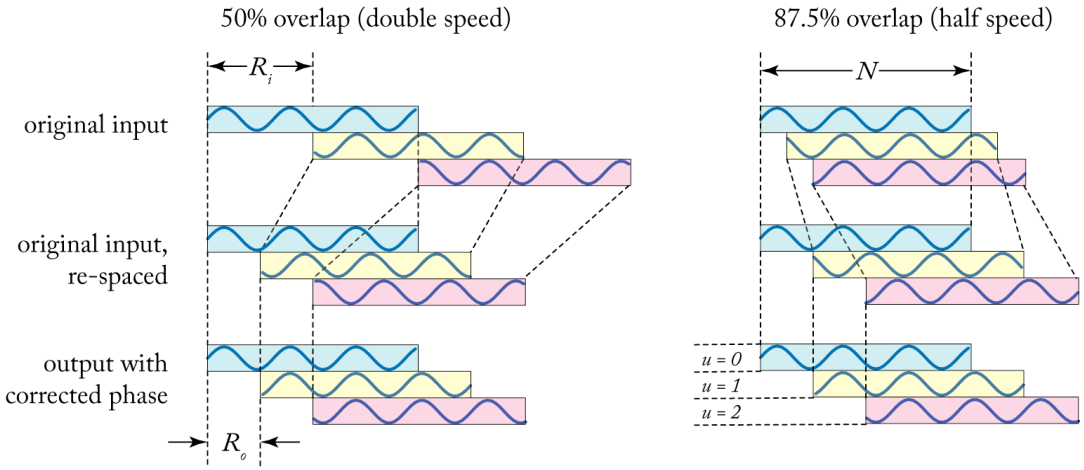

Figure 3: Phase vocoder algorithm. The left and right figures show the playback speed, $\frac{R_i}{R_o}$, doubled and halved, respectively. Respacing the buffers results in phase mismatches, which are corrected using the phase estimation calculation.

To maintain coherency between the re-spaced windows, the phase of the signal's *short time Fourier transform* (STFT) must be adjusted. Let us denote t_i^u and t_o^u to be the times of the u^{th} window of the input and output, respectively, and $\Omega_k = \frac{2\pi k}{N}$ to be the center frequency of the k^{th} frequency bin of the STFT of a window. Then the phase of the output signal, $\angle Y\left(t_o^u, \Omega_k\right)$, can be calculated using the following formulae:

$$\angle Y\left(t_o^u, \Omega_k\right) = \angle Y\left(t_o^{u-1}, \Omega_k\right) + R_o \hat{\omega}\left(t_i^u\right) \tag{3}$$

$$\hat{\omega}_k\left(t_i^u\right) = \Omega_k + \frac{1}{R_i}\Delta_p \Phi_k^u \tag{4}$$

$$\Delta\Phi_k^u = \angle X\left(t_i^u, \Omega_k\right) - \angle X\left(t_i^{u-1}, \Omega_k\right) - R_i \Omega_k \tag{5}$$

$\Delta\Phi_k^u$ is the heterodyned phase difference between two consecutive input windows; its principal determination between -π and π, $\Delta_p \Phi_k^u$ is then used to estimate the instantaneous frequency $\hat{\omega}_k\left(t_i^u\right)$ of a sinusoid in the k^{th} frequency bin. This frequency value is finally used to calculate the phase, $\angle Y\left(t_o^u, \Omega_k\right)$. A more detailed explanation of the phase vocoder algorithm is given in (Flanagan and Golden 1966; Laroche and Dolson 1999).

Unfortunately, the basic phase vocoder algorithm exhibits two types of artifacts: transient smearing, which occurs because transient signals such as drums are incorrectly treated as sinusoids; and a reverberation-like effect, a result of errors in the above phase estimation calculation, that in turn result in a loss of phase coherence between the different frequency bins of a single window. We will focus our discussion on the latter, and refer the interested reader to existing literature on the problem of transient smearing (Bonada 2000; Röbel 2003).

Rigid Phase Locked Phase Vocoder

The reverberation artifacts exhibited by the basic phase vocoder algorithm are caused by applying the same computations for phase estimation to each frequency bin of the STFT, irrespective of whether or not a sinusoid from the original audio signal exists in that bin. To address this problem, Laroche and Dolson (1999) proposed a *rigid phase locking* technique, which aims at preserving the original signal structure. The algorithm, which builds upon the basic phase vocoder, works roughly as follows:

1. Find the frequency bins that contain amplitude peaks in the STFT, which correspond to sinusoids in the original audio.

2. For each of these peak frequency bins, estimate the phase using the basic phase vocoder algorithm (3).

3. For the remaining frequency bins, "lock" their phase to the phase of the nearest peak frequency bin using the relation $\angle Y\left(t_o^u,\Omega_k\right) = \angle Y\left(t_o^u,\Omega_p\right) + \angle X\left(t_i^u,\Omega_k\right) - \angle X\left(t_i^u,\Omega_p\right)$. Ω_p is the center frequency of the peak frequency bin closest to Ω_k.

Laroche and Dolson use a simple local maxima search for the peak-picking algorithm in the first step: a peak is assumed to exist if the STFT amplitude for a frequency bin is larger than its two neighboring bins. Using even this simple technique results in a dramatic improvement over the basic phase vocoder; however, a closer examination of the resulting time-stretched audio still reveals audible artifacts. In particular, the bass sounds less full, and exhibits musical overtones compared to the original signal - artifacts coincidentally similar to those observed in audio poorly encoded using the popular MPEG Audio Layer 3 (MP3) compression algorithm.

Multiresolution Peak-Picking Algorithm

The shallow bass and musical overtone artifacts in the rigid phase locked phase vocoder can be attributed to Laroche and Dolson's constant-resolution peak-picking algorithm (Laroche and Dolson 1999). The human ear interprets frequencies non-uniformly, but the peak-picking algorithm they proposed is applied to frequencies in a linear scale.

A phase vocoder technique that attempts to compensate for non-uniform characteristics of the human ear has been proposed previously by Garas (1998), although the validity of that particular technique has also been questioned (Bernsee 2003).

Instead of proposing a wholly new approach to phase vocoding, we refine Laroche and Dolson's peak-picking algorithm to consider this non-uniformity by approximating the hu-

man ear's frequency response logarithmically. We divide the spectrum into appropriately sized regions and become more selective in choosing which bins contain a peak for higher frequencies (see Figure 4). For an STFT window size of 4096 samples, we assumed that the lowest 16 frequency bins contain a peak corresponding to a sinusoid audible to the human ear. For the next 16 bins, a bin is considered to contain a peak if its amplitude is larger than both of its neighboring bins. For the next 32 bins, the amplitude must be larger than its two neighboring bins, and so on.

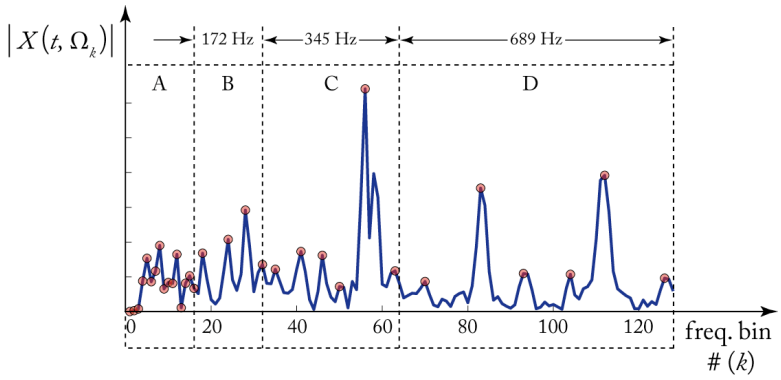

Figure 4: Multiresolution peak-picking algorithm. An amplitude plot of the lowest 128 frequency bins is shown, divided into regions that become exponentially larger for higher frequencies. Peak selection is stricter for higher frequencies.

In preliminary listening tests with users, we found that this *multiresolution peak-picking* algorithm produces an audible improvement in the bass and reduces musical overtones in the time-stretched signal.

Implementation

Our phase vocoder was implemented in C, and optimized for the vector processing unit on PowerPC G4 and higher processors. An 800 MHz G4 processor is sufficient to time-stretch stereo audio sampled at 44.1 kHz in real time.

In our current implementation, the input audio is sampled at 44.1 kHz and an STFT window size of 4096 samples (93 msec) with a constant output overlap of 75% is used. Thus, each output audio block is 1024 samples (23 msec), and the playback speed can be adjusted 43 times per second. To minimize latency but still grant the system the necessary processing time, we chose to start the computationally intensive calculations for time-stretching one audio block in advance. Since an audio block is scheduled for playback while the previous audio block is still playing (see Figure 5), the maximum possible time that can elapse between a speed change request and when it takes effect is 69 msec; this latency is less than a $\frac{1}{64}$ note in a 4/4 piece at 100 beats per minute. The scheme maximizes use of the processor, at the cost of slightly increased latency: a computer barely able to perform the time-stretching calculations in less than 23 msec will be given the opportunity to do so. In any case, some latency in the speed changes is unavoidable, which has consequences for audio and video synchronization, discussed below.

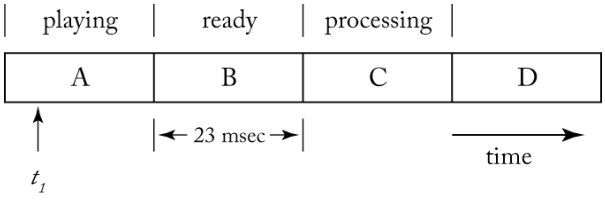

Figure 5: Temporal ordering of audio processing and playback. At time t_1, sample block B is scheduled for playback and audio time-stretching for block C begins. If a speed change event arrives immediately after t_1, it does not take effect until block D.

Video Time-Stretching

Video can be time-stretched by altering the playback speed; while this simple algorithm creates artifacts for certain types of video - for example, free-falling objects no longer behave naturally - this is not a problem with our relatively static footage of an orchestra.

Audio and Video Synchronization

The last step in our design is to synchronize the audio stream with the video stream. Synchronization is an area that has been studied extensively. Because of the previously discussed latency between when a speed change is requested and when it takes effect, independently setting the speeds of the audio and video results in drift over time. Two-way communication between the components being synchronized is required to ensure they remain synchronous (Lamport 1978), and these algorithms are implemented in all modern multimedia frameworks such as QuickTime and DirectShow/DirectSound.

Unfortunately, these same multimedia frameworks are incapable of handling media where the timebase changes continuously, as is the case with dynamically time-stretched audio and video. In Personal Orchestra, we were able to circumvent the synchronization problem by using pre-processed audio tracks that were pitch-shifted rather than time-stretched. Since the mathematical inverse of time-stretching is pitch-shifting, one can decompose a time-stretching operation into a pitch-shift followed by resampling. A pitch-shifted signal stays in the same timebase, so the movie we prepared for playback consists of a series of tracks pitch-shifted offline in half-tone steps. During playback, we switched to the appropriate track and delegated the resampling to QuickTime, which negated the pitch-shift. Thus, for Personal Orchestra, the audio time-stretching was partially completed offline, with the resampling, video time-stretching, and synchronization handled by QuickTime (see Figure 6).

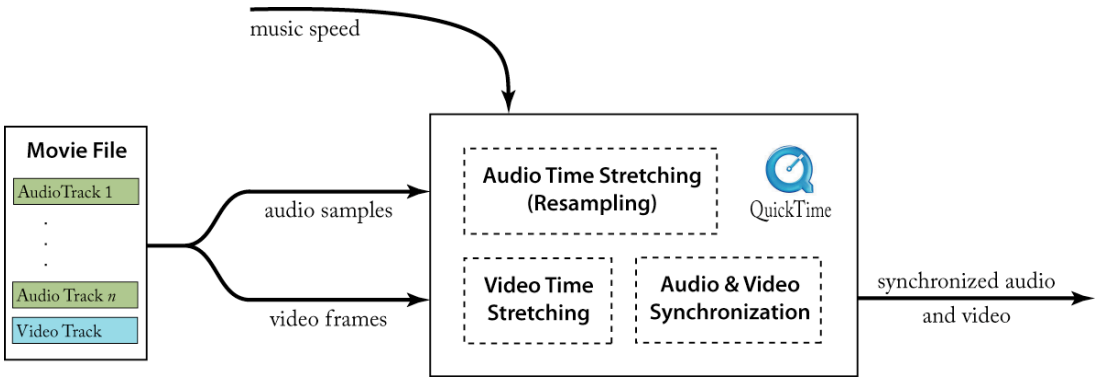

Figure 6: Audio/video rendering in Personal Orchestra. The source movie has *n* audio tracks pitch-shifted at various offsets. During playback, QuickTime selects an audio track and re-samples it to negate its pitch-shift and achieve the desired playback speed. Audio and video synchronization is handled internally by QuickTime.

It should be noted that this inverse relationship between pitch-shifting and time-stretching is purely mathematical. Practically, the algorithms cannot produce perfect results due to discretization and quantization, and thus rather diverse approaches for addressing the artifacts specific to each operation have been proposed. Formant correction, for example, is important for pitch-shifting (Lent 1989), while compensating for transient smearing is more applicable to time-stretching.

To achieve the highest quality time-stretch possible, it is thus undesirable to employ the Personal Orchestra approach of pitch-shifting followed by resampling. In You're the Conductor, our time-stretching algorithm directly outputs the resampled audio data; however this resampling operation in the time-stretching changes the timebase of the data. Although current multimedia frameworks ensure synchronous audio and video playback, they are incapable of handling media with continuously changing timebases, and thus the responsibility of synchronizing time-stretched audio with video falls on the developer of the application itself.

We chose to synchronize the video to the audio in You're the Conductor, and calculate an adjusted video playback speed to ensure the video catches up with the audio by the end of the next audio block:

$$f_{adj} = \frac{T_o - T_c + \Delta T}{t_o - t_c + \Delta t} \qquad (6)$$

Here, t denotes "real time" (the timebase of the stretched audio) and T denotes "movie time" (the timebase of the video and the unstretched audio, see Figure 7). t_c is the current time and t_o is the time at which the next audio block will begin playing with a new speed. Δt and ΔT are the lengths of the audio block in real time and movie time, respectively. T_c is the current movie time in the audio and video stream, and T_o is the movie time of the audio stream when the current audio block has finished playing. The adjusted playback speed, f_{adj}, is computed such that the video will be synchronized with the audio when the next block finishes playing at time $t_o + \Delta t$. The video playback speed is adjusted again at time t_c'.

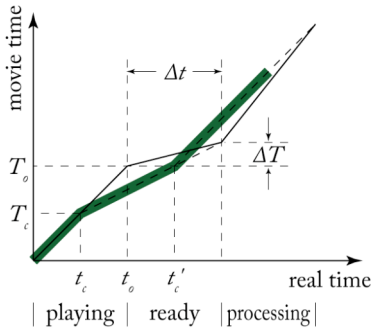

Figure 7: Audio and video synchronization algorithm. The thin and thick lines show the audio and video progression, respectively. At time t_c, the next audio output block, which starts at t_o, is scheduled for playback. Their corresponding values in movie time, T_c and T_o, are required to compute the adjusted video playback speed. This speed is adjusted again at time t_c'.

Design Analysis

A closer analysis of the various modules in our conducting system design reveals a number of interesting observations:

The equation (1) describing the algorithm for mapping user beats to music beat mapping is identical to the equation (6) used for synchronizing audio and video. The difference in the two modules lies in their individual implementations: which parameter to solve for, and the units of time used to calculate the adjusted speed: beats for one, and seconds for the other. Since the tempo of a piece varies during a performance, the beats in the resulting digital audio data will not be evenly spaced, and thus a conversion from beats to seconds is non-trivial.

The design presented in Figure 1 consists of multiple stages of synchronization. First, the user beats are synchronized with the audio, and then the audio is synchronized with the video. This "daisy chaining" is necessary because the time units implicitly change from beats to seconds between the first and second synchronization stages.

The parameters required to perform the audio and video synchronization cannot be obtained from the time-stretched audio and video alone, due to the timebase change from the implicit resampling operation in the audio time-stretching; this is the main reason for having to implement a custom audio and video synchronization module instead of delegating this responsibility to the multimedia framework. Figure 8 is a more detailed diagram of the audio/video rendering modules that shows where the parameters required for audio and video synchronization originate. To compute the adjusted video speed, the synchronization module must obtain the length of the unstretched audio block, ΔT in equation (7), from the original audio signal, indicated by the thick dotted line in Figure 8. This additional connection is normally unnecessary, as in the case of user to music beat mapping, and is furthermore inelegant because it is a connection further up the data flow chain, breaking the modular nature of the intended design.

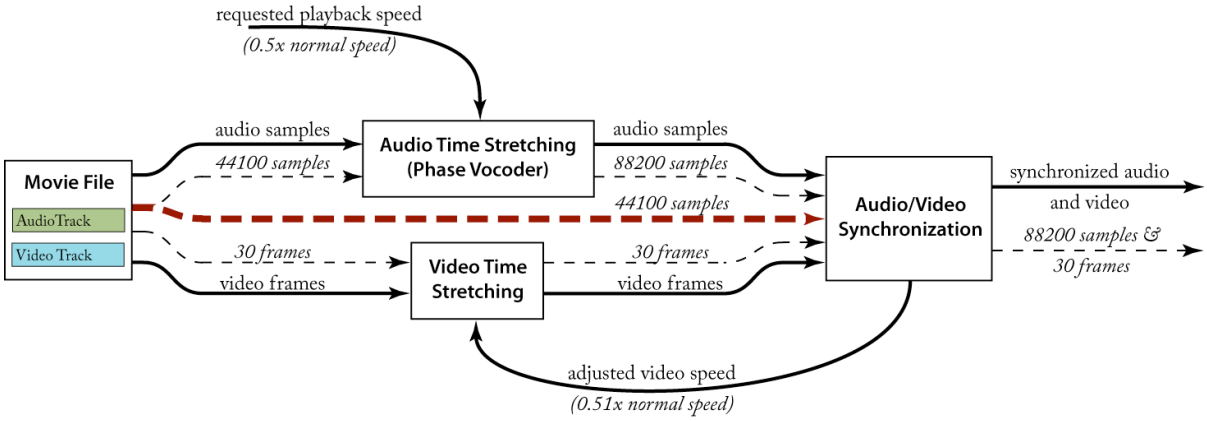

Figure 8: Audio/video rendering in You're the Conductor. The dotted lines show the flow of temporal information; example values are given for one second of input data for clarity. The audio and video synchronization module must retrieve temporal information from the original audio data, indicated by the thick dotted line, to compute the adjusted stretch factor for the video.

These above problems could be solved by using synthesized audio and video, since the semantics of the data are explicitly known (see Figure 9). This observation points us to a possible solution for sampled audio and video streams as well: the semantic nature of the data must be retained throughout the processing pipeline.

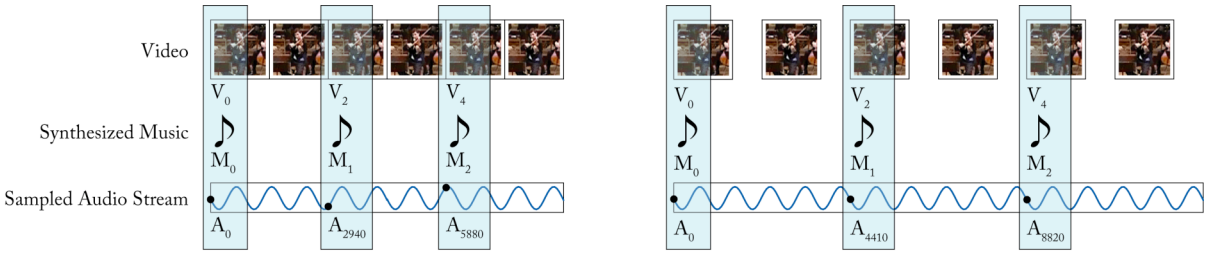

Figure 9: Audio and video synchronization with synthesized music versus a digitally sampled audio stream. On the left is the original movie where 6 frames of video are synchronized with 3 MIDI note events and an audio stream with 6 samples. On the right, the movie has been time-expanded by 50% (recall that video time-stretching does not change the total number of frames, only the speed at which they are played). The mapping between synthesized music and video remains the same; however, the mapping of video frames to audio samples has changed, because of the increased number of samples. More information is required to resynchronize the sampled audio with the video.

A Conceptual Model Problem

The conflict in time models between synthesized music and digitally sampled audio streams can be summarized as a *conceptual model* problem. A conceptual model represents how people are likely to think about and respond to a system (Liddle 1996). Our conceptual model of time is similar to beats and notes in a musical piece, the "movie time". Current multimedia frameworks, on the other hand, expose a time model based on clock ticks in real time, the "clock time". This clock drives, for example, audio sampled at 44.1 kHz and video at 30 frames per second. For applications that do not manipulate time, these two models are compatible, since movie time can be described as a linear function of real time. Consider, however, a more complex example of an audio clip divided into three segments, where the first segment is time expanded, the second is time compressed, and the third remains unchanged (see Figure 10). The relationship between movie time and clock time, which is now non-linear, can no longer be described as easily. This incompatibility of time models in current multimedia frameworks motivated us to propose a new *semantic time framework*.

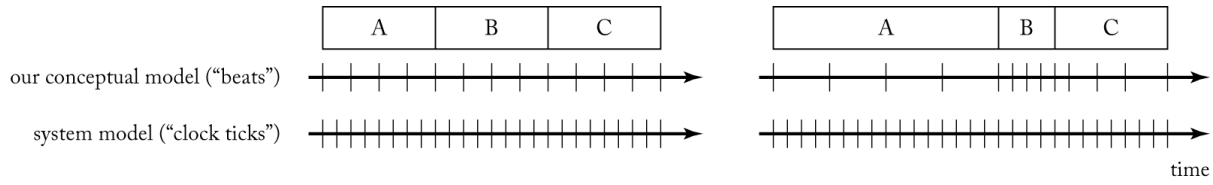

Figure 10: Difference between our conceptual model of time and the system model of time. On the left, an audio clip is divided into three segments. The two time models, beats and clock ticks, are also shown. On the right, segment A is time-expanded and segment B is time-compressed, creating a non-linear relationship between the two models.

The Semantic Time Framework

The above problems apply more generally to applications which incorporate time-based manipulation of multiple, synchronized digital audio and video streams. While our interactive conducting systems are one example of this type of application, other examples include: disc jockey (DJ) software, where two or more independent music tracks must be synchronized; personal video recorders (PVRs), which allow users to view recorded programs at any arbitrary playback speed; and video conferencing systems, which must resynchronize the audio and video streams as they arrive from the other speaker. Thus, to address the problem more generally, we chose to introduce the "semantic time framework", which borrows some concepts taken for granted in synthesized music, and generalizes them for digitally sampled audio and video streams. The generalized and reusable software components that form our framework can then be directly reused to assist in the development of these other applications.

Semantic Time

The main contribution of our new framework over current ones such as QuickTime, DirectShow/DirectSound or Max/MSP is the adoption of a temporal model that is linked to the semantics of the underlying data.

Distinction between semantic time and real time has been proposed before (Bordwell and Thompson 2003; Davis 1993); several theoretical constructs (Jaffe 1985; Honing 2001) have been created to more easily understand and manipulate the mapping from "score time" to "real time" in a musical performance. Jaffe's time maps, in particular, are supported as a protocol for synchronizing to the MIDI time code in MusicKit, a software system for building music, sound, signal processing, and MIDI applications (http://musickit.sourceforge.net). To date, however, there seem to be no frameworks for sampled, time-based data such as audio and video where this distinction between semantic time and real time is integral to the system.

One approach to address the conflict in time models is to preserve the original movie timecode throughout the processing pipeline, especially after time-based operations such as time-stretching; there are no well-known frameworks today that do even this. Alternatively, one could attempt to apply the MIDI model of time, based on beats and notes, to the sampled audio stream; ignoring for the moment how one could go about extracting the note information from an arbitrary sampled musical recording, this scheme would still not be satisfactory as digitally sampled audio streams are clearly not limited to music. Our goal is a more general approach, which we will show has additional benefits for more complex applications such as an interactive conducting system.

Our framework adopts a temporal model of *semantic time*. The current model of a clock ticking at regular intervals in real time is an artificial construct created by engineers to more conveniently work with digitally sampled data streams. Semantic time, on the other hand, is tied to the semantics of the media, “movie time”, and is thus more intuitive for users of the framework.

To illustrate the concept of semantic time, consider again the difference in mental models when time expanding one second of audio sampled at 44.1 kHz by a factor of two. For regular users, this model is “one second of audio at half speed”. Audio frameworks, however, represent the time-stretched audio only as 88200 samples - the semantic information about it being “one second” is lost. Now, let us arbitrarily divide this 1 second audio clip into 1000 *semantic time units* (stu) originally spaced 1 msec of clock time apart. After time expansion, the audio still consists of 1000 stu, but now spaced 2 msec of clock time apart, which is synonymous with the user model.

Where our framework becomes particularly helpful is when one chooses semantic time units that are not spaced at regular intervals. Particularly for music, a more appropriate scale is “beats of the music” (1 stu = 1 beat), and an audio recording of a musical performance will undoubtedly contain variations in the tempo. The idea is the same, however, since the number of beats does not change after time-stretching - only the spacing between them has. What our framework does is to abstract the tedious bookkeeping of which audio sample corresponds to a musical beat from the user, which, as we have shown, changes when the audio is

time-stretched.

Implementation

Our prototype implementation for this framework was written in C++ running on Mac OS X. Video data is encapsulated as a QuickTime movie with extra metadata for storing semantic time. Audio is stored as 32-bit floating point samples and uses Apple's Core Audio library (http://developer.apple.com/audio); a semantic time value is stored for each sample as metadata. The framework allows one to programmatically link modules such as the synchronization and time-stretching together. It also provides services for working with semantic time units to assist in the creation of new modules.

Modules that change the timebase of the media must ensure the mapping from semantic time to sample number is maintained. In the case of the audio time-stretching module using the phase vocoder, this bookkeeping is somewhat tedious due to the fact that the semantic time units can be non-evenly spaced, and due to the nature of the phase vocoder algorithm computing the final output using an overlap-add method (see Figure 11). Our algorithm for computing the semantic time values, $t(s)$, for the output buffer samples is:

$$t(s) = interpolate\left(t'(\lfloor s' \rfloor), t'(\lceil s' \rceil)\right) \tag{7}$$

$$s' = \frac{s}{\Delta s}\left(s'_1 - s'_0\right) \tag{8}$$

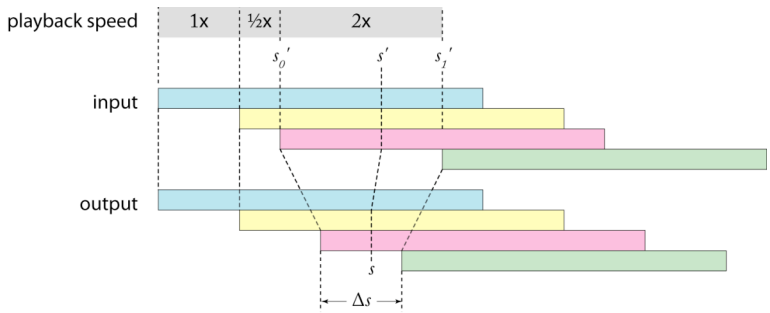

Figure 11: The semantic time value stored with each audio sample, s, must be recomputed when the audio is time-stretched, using the corresponding sample, s', in the original audio.

Here, s is a sample number in the output buffer with a corresponding sample number s' in the input buffer. Since s' will normally lie in between two sample numbers, $t(s)$ must be interpolated from the neighboring semantic time values, $t'()$, in the input buffer; our implementation currently uses linear interpolation. Δs is the distance between two windows in the output buffer.

The semantic time values for an output audio block are computed just before processing of the next block begins, since the starting sample number of this next block in the input buffer must be known to perform the calculation.

Note that once this module has been implemented once, it can be reused in other applications that utilize audio time-stretching without any further modifications, and thus abstracting the above details from the user.

Improved Conducting System Design

This framework, with a more general and intuitive time model for digitally sampled multimedia streams, has allowed us to simplify the design of our interactive conducting systems.

We begin by redesigning the audio/video rendering modules in the context of our framework (see Figure 12). Unlike the original design (Figure 8), this revised design is independent of implementation. The input and output parameters, in units of semantic time, neatly abstract the implementation details. Our framework, for example, accommodates both the pitch-shifting followed by resampling and the phase vocoder approaches to time-stretching without requiring any changes to the synchronization module. The temporal information that flows between the modules, in units of semantic time, is also more intuitive for users of the framework. Moreover, one does not need to worry about the details of remapping audio samples to video frames because of the timebase change.

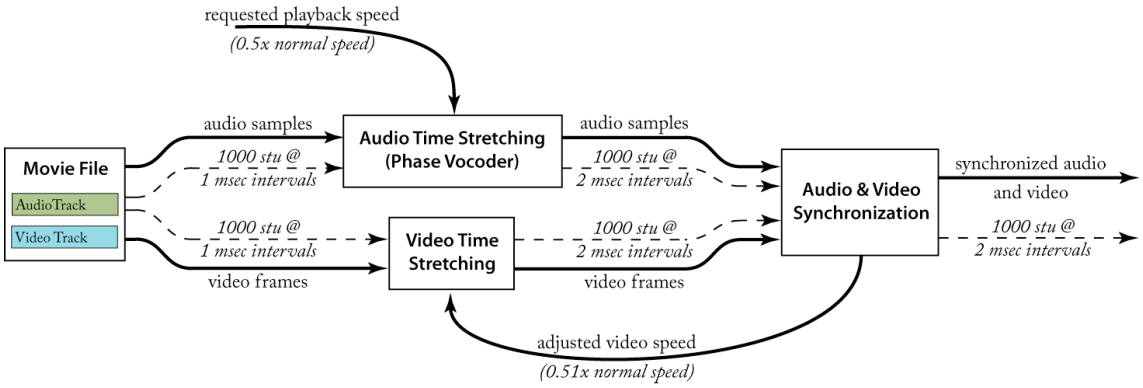

Figure 12: Revised audio/video rendering engine in You're the Conductor using the semantic time framework (compare with Figure 8). Temporal information is now in a more intuitive scale of semantic time units (stu); the design is also more elegant in that it does not expose any implementation-specific details.

With this modular, implementation-independent synchronize module, we can further simplify our general conducting system design, since the audio and video synchronization algorithm is identical to the mapping of user beats to music beats. Using a common semantic time unit of musical beats throughout the entire system, we can combine the two operations into a single one where the audio and video are synchronized directly to the musical beats (see Figure 13).

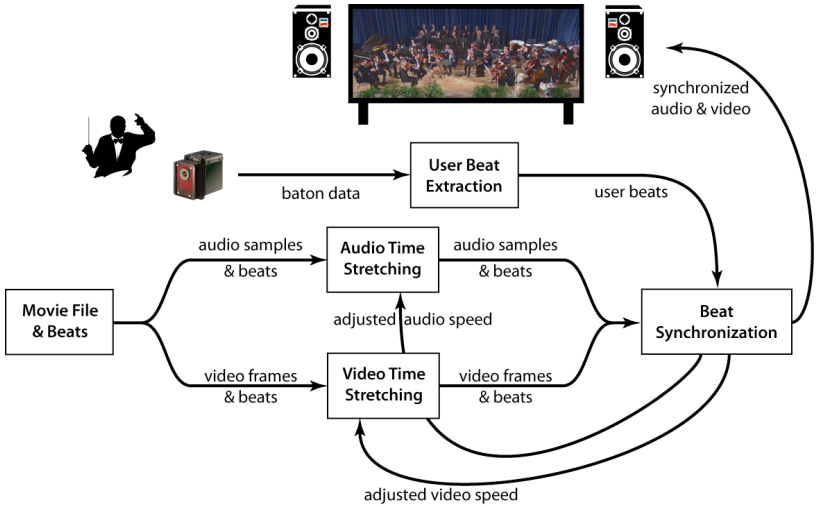

Figure 13: Revised conducting system design (compare with Figure 1). The two synchronization modules have been merged, and beats are used as the time units throughout the system.

In our current implementation, the beats of the music were manually determined offline using a simple software tool we wrote; this tool allows one to tap out the music beats along to the music, and then fine-align them using a view of the audio waveform. One could also use a beat detection algorithm presented in current literature (Jensen and Andersen 2003; Goto 2001; Dixon 2001) to do this in real time. Unfortunately, there currently does not appear to be an algorithm that is able to reliably detect the beat of orchestral music, such as the Blue Danube Waltz by Johann Strauss. In a recording we have of this piece, performed by the Vienna Philharmonic, the tempo varies between 15 and 80 beats per minute, with an average of 50 beats per minute; we have observed that even people often have trouble tracking the beat of this piece.

Beyond Interactive Conducting Systems

Our discussion so far has focused on how our semantic time framework can help one more easily build interactive conducting systems which use time-stretching. One could argue, however, that time-stretching is a rather semantically simple operation on the temporal axis, similar to scaling the brightness of an image. As our framework matures, we hope to expand this framework to include more semantically rich time-based operations, such as: a module to remove "dead" time where the audio is silent or the video is empty; a module to equalize the time axis such that semantic time units become uniformly spaced; a module which shifts the micro-timing within a beat to create a "swing" effect in music.

Another area we are currently exploring is other possibilities for semantic time units. Units of music beats, which can be non-uniformly spaced in time, were used to demonstrate the flexibility of our framework. What other units exist for music or other types of audio such as speech?

Conclusions

We discussed the challenges of designing interactive conducting systems that incorporate digitally sampled audio and video streams. We presented our implementation of the phase vocoder that features a multiresolution peak peaking algorithm; the algorithm compensates for the human ear's non-uniform response to frequency, and produces audibly better results over previous algorithms. We also discussed how the use of time-stretching in sampled audio and video, which modify the timebase of the media, led to an increasingly complex and un-modular design. Moreover, unnecessary effort was required to design and implement an audio and video synchronization module, functionality that is already available in modern multimedia frameworks, but was not suitable for use in our application. Our analysis of these problems motivated us to design and implement a new multimedia framework using semantic time. This framework, unlike current multimedia frameworks, uses "semantic time", which is more general and allows non-uniform spacing between the time units. Semantic time allows us to support operations which change the timebase of the data, such as time-stretching, without breaking the modularity of the intended design or require users of the framework to think about the tedious bookkeeping details associated with these timebase changes. Finally, we showed how applying our framework to interactive conducting system has resulted in a simpler and more elegant design.

As we continue our work in designing interactive conducting exhibits for public spaces, we will further improve our time-stretching algorithms, and refine our semantic time framework for time-based effects with multimedia. Although discussion of our semantic time framework was limited to the context of our interactive conducting systems, we are continuing to explore how this framework enables more sophisticated time-based interaction with music; the semantic time framework is the foundation for not only our latest conducting system under development for the Betty Brinn Museum in Milwaukee, but also a system for computer-assisted, co-operative improvisation system with medieval music. Time-based interaction with music and multimedia is becoming increasingly popular, and our aim is to build a toolkit of time-based effects that will assist others in constructing increasingly innovative, interactive multimedia systems.

Acknowledgements

The authors would like to thank Max Mühlhäuser, Wolfgang Samminger, the HOUSE OF MUSIC in Vienna, and the Vienna Philharmonic for their contributions to Personal Orchestra; Teresa Marrin Nakra, Immersion Music, Ron Yeh, the Boston Children's Museum, and the Boston Pops for their contributions to You're the Conductor; Ingo Grüll for his contributions to the hybrid Personal Orchestra; and Julius Smith and Rafael Ballagas for their valuable feedback in an early draft of this article.

References

Andersen, T. H. 2003. "Towards novel DJ interfaces." *Proceedings of the NIME 2003 Conference on New Interfaces for Musical Expression*. pp. 30–35.

Bernsee, S. M. 2003. "Time Stretching and Pitch Shifting of Audio Signals." *The DSP Dimension*. http://dspdimension.com/html/timepitch.html.

Bonada, J. 2000. "Automatic technique in frequency domain for near-lossless time-scale modification of audio." *Proceedings of the ICMC 2000 International Computer Music Conference*. Berlin: ICMA, pp. 396–399.

Borchers, J., E. Lee, W. Samminger, and M. Mühlhäuser. 2004. "Personal Orchestra: A real-time audio/video system for interactive conducting." *ACM Multimedia Systems Journal Special Issue on Multimedia Software Engineering*, 9(5): 458–465. Errata published in next issue.

Bordwell, D., and K. Thompson. 2003. *Film Art: An Introduction*. New York: McGraw-Hill.

Camurri, A., B. Mazzarino, and G. Volpe. 2003. "Analysis of Expressive Gesture: The EyesWeb Expressive Gesture Processing Library." *Gesture Workshop 2003* Lecture Notes in Computer Science, vol. 2915. Genova: Springer, pp. 460–467.

Davis, M. 1993. "Media Streams: An Iconic Visual Language for Video Annotation." *Telektronikk*, 4(93): 59–71.

Dixon, S. 2001. "An Interactive Beat Tracking and Visualisation System." *Proceedings of the ICMC 2001 International Computer Music Conference*. Havana: ICMA, pp. 215–218.

Flanagan, J. L., and R. M. Golden. 1966. "Phase Vocoder." *Bell Systems Technical Journal*, 45(Nov): 1493–1509.

Gabor, D. 1946. "Theory of Communication." *Journal of Institution of Electrical Engineers*, pp. 429–457.

Garas, J., and P. C. Sommen. 1998. "Time/Pitch Scaling Using the Constant-Q Phase Vocoder." *Proceedings of the 1st STW Workshop on Semiconductor Advances for Future Electronics (SAFE 98).* pp. 173–176.

Goto, M. 2001. "An Audio-based Real-time Beat Tracking System for Music With or Without Drum-sounds." *Journal of New Music Research*, 30(2): 159–171.

Honing, H. 2001. "From time to time: The representation of timing and tempo." *Computer Music Journal*, 35(3), 50–61.

Ilmonen, T., and T. Takala. 1999. "Conductor Following With Artificial Neural Networks." *Proceedings of the ICMC 1999 International Computer Music Conference*. Beijing: ICMA, pp. 367–370.

Jaffe, D. 1985. "Ensemble Timing in Computer Music." *Computer Music Journal*, 9(4): 38–48.

Jensen, K., and T. H. Andersen. 2003. "Beat estimation on the beat." Proceedings of the IEEE Workshop on Applications of Signal Processing to Audio and Acoustics. New York: IEEE.

Kolesnik, P. 2004. Conducting Gesture Recognition, Analysis and Performance System. M.S. thesis, McGill University.

Lamport, L. 1978. "Time, clocks, and the ordering of events in a distributed system." *Communications of the ACM*, 21(7): 558–565.

Laroche, J., and M. Dolson. 1999. "Improved Phase Vocoder Time-Scale Modification of Audio." *IEEE Transactions on Speech and Audio Processing*, 7(3): 323–332.

Lee, E., T. Marrin Nakra, and J. Borchers. 2004. "You're the Conductor: A Realistic Interactive Conducting System for Children." *Proceedings of the NIME 2004 Conference on New Interfaces for Musical Expression*. pp. 68–73.

Lee, E., M. Wolf, and J. Borchers. 2005. "Improving Orchestral Conducting Systems in Public Spaces: Examining the Temporal Characteristics and Conceptual Models of Conducting Gestures." *Proceedings of the CHI 2005 Conference on Human Factors in Computing Systems*. pp. 731–740.

Lee, M., G. Garnett, and D. Wessel. 1992. "An Adaptive Conductor Follower." *Proceedings of the ICMC 1992 International Computer Music Conference*. San Jose: ICMA, pp. 454–455.

Lent, K. 1989. "An Efficient Method for Pitch Shifting Digitally Sampled Sounds." *Computer Music Journal*, 13(4): 65–71.

Liddle, D. 1996. Design of the Conceptual Model in *Bringing Design to Software*, edited by T. Winograd, 17–31. Addison-Wesley.

Marrin Nakra, T. 2000. *Inside the Conductor's Jacket: Analysis, interpretation and musical synthesis of expressive gesture*. Ph.D. thesis, Massachusetts Institute of Technology.

Mathews, M. V. 1991. *Current Directions in Computer Music Research*. Cambridge: MIT Press. Chap. The Conductor Program and Mechanical Baton, pages 263–282.

Morita, H., S. Hashimoto, and S. Ohteru. 1991. "A Computer Music System that Follows a Human Conductor." *IEEE Computer*, 24(7): 44–53.

Murphy, D., T. H. Andersen, and K. Jensen. 2003. "Conducting Audio Files via Computer Vision." *Gesture Workshop 2003* Lecture Notes in Computer Science, vol. 2915. Genova: Springer, pp. 529–540.

Puckette, M. 2002. "Max at seventeen." *Computer Music Journal*, 26(4): 31–43.

Röbel, A. 2003. "Transient detection and preservation in the phase vocoder." *Proceedings of the ICMC 2003 International Computer Music Conference*. Singapore: ICMA, pp. 247–250.

Usa, S., and Y. Mochida. 1998. "A Multi-modal Conducting Simulator." *Proceedings of the ICMC 1998 International Computer Music Conference*. Ann Arbor: ICMA, pp. 25–32.