DiMaß: A Technique for Audio Scrubbing and Skimming using Direct Manipulation

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

ABSTRACT

Scrubbing or skimming through an audio-only recording remains a challenge with today's audio interfaces. We present DiMaß, a technique for direct manipulation of an audio timeline with continuous, high-fidelity audio feedback. Building upon prior work in interactive conducting systems, DiMaß uses improved algorithms to (1) estimate the input position and velocity of low sampling rate, relative input devices such as a mouse or iPod scroll wheel; (2) adjust audio play rate to precisely track user input; and (3) interactively time-stretch the audio without changing the pitch at arbitrary forwards and backwards play rates. Early feedback showed that users are willing to tolerate slightly reduced responsiveness in exchange for smoother sounding audio. We are currently exploring how DiMaß enables users to more quickly and efficiently scrub and skim through audio.

Categories and Subject Descriptors

H.5.1 [**Information interfaces and presentation (e.g., HCI)**]: Multimedia Information Systems—*Audio input/output*; H.5.2 [**Information interfaces and presentation (e.g., HCI)**]: User Interfaces—*Interaction styles*; H.5.5 [**Information interfaces and presentation (e.g., HCI)**]: Sound and Music Computing—*Systems*

General Terms

Algorithms, Human Factors

Keywords

audio scrubbing, audio skimming, direct manipulation, audio interfaces, synchronization, time-stretching

1. INTRODUCTION

While interfaces for skimming and searching through text, still images, and, to a certain extent, video exist today, the same is not true for audio. Even when searching through a movie, one must rely solely on visual feedback to accurately pinpoint a desired location in the movie timeline. Part of the challenge is that audio is an inherently time-based medium; unlike video, where a single time-instant can be interpreted as an image, no such equivalent exists for audio. To interpret audio, then, it must be perceived over time; however, arbitrary play rate adjustment of audio to support scrubbing and skimming is also challenging – the naïve method of time-stretching audio using resampling creates disturbing pitch-shifting artifacts.

Constant pitch time-stretching algorithms, where the original pitch of the audio is preserved, have been studied extensively in recent years [6, 16, 18, 24]; while these algorithms have traditionally been very computationally expensive, the processing capabilities of modern hardware continue to increase exponentially. In the near future, even small, portable electronic devices such as Apple's iPod[1] portable digital media player will be capable of time-stretching audio in real time.

It remains difficult, however, with today's software frameworks to design and implement time-based interactions with multimedia. We began to address this problem in previous work on interactive orchestral conducting systems [20]; DiMaß[2] continues this work, providing a technique for **Di**rect **M**anipulation **A**udio **S**crubbing and **S**kimming.

Shneiderman defines a *direct manipulation* interface as one with "visible objects and actions of interest, with rapid, reversible, incremental actions and feedback" [25]. DiMaß allows users to interact with an audio timeline by "grabbing" on to it and sliding it around (see Figure 1); continuous audio feedback is provided using a high-fidelity, constant pitch time-stretching algorithm, so that users are always aware of where they are in the audio.

There exists an abundance of previous research demonstrating how directly manipulating position is superior to rate-based control for navigation tasks (e.g., manipulating a cursor on a screen) [7, 14]. However, there does not appear to be any existing systems that allow the user to directly manipulate the position of an audio timeline whilst receiving continuous, high-fidelity audio feedback.

[1]http://www.apple.com/ipod/
[2]DiMaß is pronounced *dee-MAHS*; the "ß" is the German sharp S, an abbreviation for two S's, not the Greek letter beta. The word "DiMaß" is a pun on the German word *das Maß*, which means "quantity of measure". *die Maß*, which is pronounced the same way as DiMaß, is a colloquial term used in southern Germany for "a liter (34 ounces) of beer" (think Oktoberfest).

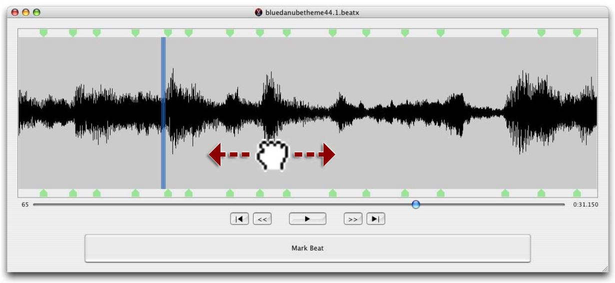

Figure 1: DiMaß interaction: the user "grabs" onto the waveform and slides it to the left or right. The audio waveform follows the cursor, and time-stretched audio is played synchronously with the movements.

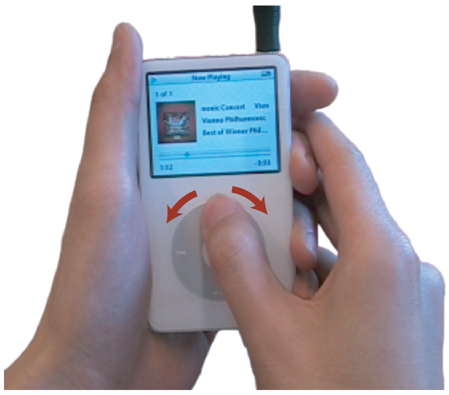

Figure 2: Navigating an audio timeline using the iPod scroll wheel. Clockwise gestures advance the audio position forwards, and counter-clockwise gestures backwards. No audio feedback is given whilst scrolling on a standard iPod.

Consider the following two usage scenarios:

- Yvonne is listening to a podcast of a lecture while studying for an examination. She wants to navigate to a particular part of the 90-minute lecture. Even though an iPod offers position-based control of the audio timeline via its scroll wheel, there is no audio feedback whilst scrolling (see Figure 2).

- Marius has recorded a 30-minute interview with the mayor, and wants to extract excerpts to include with his weekly five-minute podcast. This task requires navigating to specific points in the audio and placing trim markers. Many current audio editing applications, such as Adobe's *Audition*[3] do not offer audio feedback whilst trimming audio sequences, and thus the editor must resort to a tedious, iterative trial-and-error process whereby a marker is placed on the audio timeline, the audio is played back starting at the marked point at normal speed, the marker is adjusted, and so forth.

In this paper, we discuss the algorithms used in DiMaß, and one implementation of these algorithms in a tool for tagging music recordings with beat metadata. We also describe the results of informally testing DiMaß with users, using this tool.

[3]http://www.adobe.com/products/audition/

2. RELATED WORK

The use of a constant pitch time-stretching algorithm such as the phase vocoder to support audio scrubbing has been proposed previously [26]. To date, however, there appears to be no concrete implementation demonstrating this interaction.

Certain audio editing applications offer audio scrubbing, where audio feedback is provided in response to cursor movement on the audio timeline. These applications either track user input precisely, but render the audio with low-fidelity, or vice-versa. The waveform editor included in Apple's *Final Cut Pro*[4], for example, simply skips to the desired audio position and plays a short snippet at that position while scrubbing. Sagan Technology's *Metro*[5], an audio/video/MIDI sequencing software, provides variable pitch audio feedback during scrubbing, but the audio cannot be scrubbed above or below a certain rate. DiMaß offers an improved interaction where the audio precisely tracks user input, and the time-stretched audio is both high-fidelity and constant pitch.

Hürst et al.'s *elastic audio slider* [13] and Arons' *SpeechSkimmer* [4] support constant pitch audio skimming for speech. In addition, Arons' *SpeechSkimmer* offers multiple semantic levels of speech skimming by, for example, automatically detecting and eliminating pauses in the speech. These systems, however, offer rate-based control rather than position-based control of the audio timeline. DiMaß, in contrast, allows the user to directly manipulate the timeline of audio, and can time-stretch complex, polyphonic audio, such as music, at high quality, in addition to monophonic audio signals such as speech.

Direct manipulation of an audio timeline has also been explored in disc jockey (DJ) systems [5] and interactive orchestral conducting systems [20, 23]. However, these applications use specialized and expensive input devices that accurately report both position and velocity at frequent and regular intervals. Moreover, the range of play rate changes in response to user input is limited: in conducting, for example, the tempo seldom falls below 20 bpm (beats per minute), or rises above 240 bpm (0.25 and 3 times normal speed for an 80 bpm piece, respectively). In contrast, audio skimming can easily reach speeds close to zero and above 20 times normal speed. DiMaß supports both arbitrary scrub rates, and input from low-cost, low-power devices such as a mouse or iPod scroll wheel.

3. DIRECT MANIPULATION OF AN AUDIO TIMELINE

DiMaß can be described as users imposing their own sense of time, "user time", onto the audio. We further distinguish between "audio time", the timeline that is embedded in the original audio file, from "real time", as obtained, for example, from a clock. The relationship between audio time and real time dictates how fast the audio is playing at a given moment, for example, and this relationship is often used to study and analyze timing patterns in music [11, 15].

[4]http://www.apple.com/finalcutpro/
[5]http://www.sagantech.biz

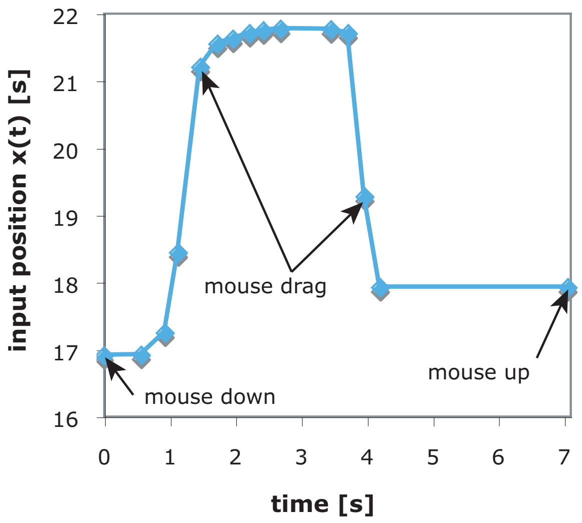

Figure 4: Example plot of mouse input position events. Only changes to mouse position are reported, resulting in irregularly spaced events. In this example, no new events are received after $t = 4$ sec because the mouse has not been moved, until the mouse button is released at $t = 7$ sec.

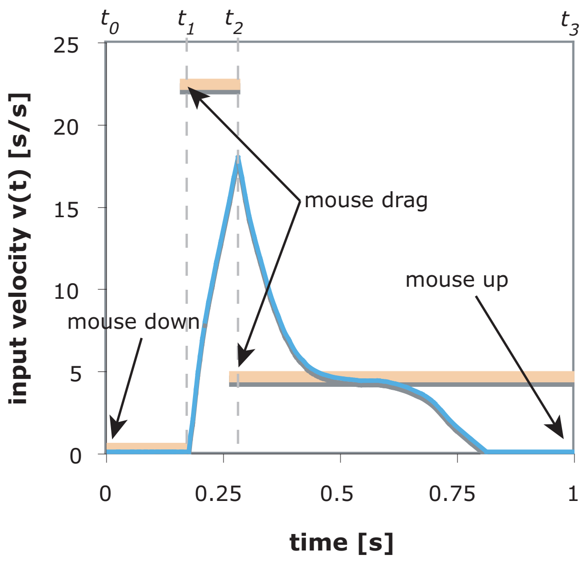

Figure 5: Estimated mouse velocity. Units are "audio seconds" per "real time second". There are two drag events, at t_1 and t_2. Note that since $t_2 - t_1$ is only 0.1 s, $v(t)$ has not had a chance to reach v_{t_1}.

We divide DiMaß into three parts: motion estimation, input tracking, and audio time-stretching (see Figure 3). The motion estimator receives position events, $p(t)$, and estimates the user's instantaneous position, $x(t)$, and velocity, $v(t)$. Note that since $x(t)$ has units of "audio seconds", $v(t)$ is measuring "audio seconds" per (real time) "second". The input tracker synchronizes the audio play rate, $r(t)$, to $x(t)$ and $v(t)$. Finally, the audio is time-stretched at this adjusted rate, and updates the input tracker with the current position in the unstretched audio, $a(t)$. $a'(t)$ is the timeline of the time-stretched audio.

3.1 Motion Estimation

Many devices, such as a mouse, report only *changes* to position [8]; the windowing system may also choose to discard input events under high system load. Thus, while the device itself may have a temporal resolution above 100 Hz, the actual time interval between input events will vary (e.g., if the mouse is not moved, no events will be received, see Figure 4). However, the input tracker requires *instantaneous* position and velocity, which we must calculate based on this irregular, and relatively infrequent, position information. For example, the user in Figure 4 has dragged the mouse pointer to audio time 18 sec at $t = 4$ sec. Until the mouse button is released at $t = 7$ sec, we are not aware that the user has not moved the mouse, and we can only estimate the instantaneous position and velocity.

When an input position event is received at time t_j, it is mapped to an absolute position in the audio waveform, $x(t_j)$. To achieve low-latency position tracking, $x(t)$ is not filtered before it is passed to the input tracker. The mouse velocity at time t_j can be estimated using:

$$v_{t_j} = \frac{x(t_j) - x(t_{j-1})}{t_j - t_{j-1}}. \tag{1}$$

To minimize discontinuities in $v(t)$, we let it rise exponentially from its current value over the next 250 ms until it reaches v_{t_j}, and then fall exponentially over the next 250 ms to zero – unless another input event is received in the meantime (see Figure 5). While it may appear that this scheme introduces a minimum input-to-response latency of 250-500 ms, we will show in the next section how the input tracker is able to combine this velocity estimate with the position information to achieve almost zero latency.

3.2 Input Tracking

The input tracker computes an adjusted audio play rate such that the audio position $a(t)$ tracks the input position $x(t)$ and velocity $v(t)$; that is, the "audio time" is synchronized to the "user time". Existing work examines ways to synchronize audio to video (or vice versa) [3, 17, 19, 21], or audio and video to conducting gestures [20]. These algorithms, however, assume the following:

- the drift between the independent (user) timebase and the dependent (audio) timebase is small
- there are no sudden changes to the speed and/or position of the user input
- the speed and position of the user input is accurate
- time is always moving forwards

For DiMaß, we addressed these limitations. Figure 6a illustrates the synchronization algorithm presented in [19]. The adjusted play rate, $r_g(t)$, is calculated using:

$$r_g(t_i) = r_g(t_{i-1}) \frac{x(t_i) + v(t_i)\Delta t - a(t_i)}{r_g(t_{i-1})\Delta t} \tag{2}$$

where t_i is the current time, t_{i-1} is the time of the last correction, and Δt is the "catch-up interval". Intuitively, this formula scales the audio play rate by the ratio of how

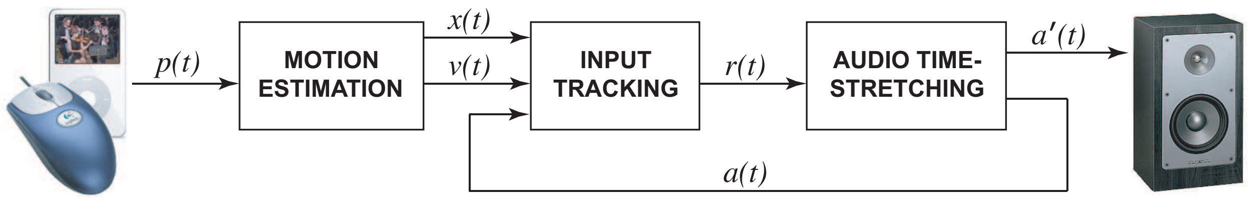

Figure 3: DiMaß block diagram. The motion estimator takes the input device position, $p(t)$, and calculates the desired audio position and velocity ($x(t)$ and $v(t)$, respectively). These are in turn used by the input tracker to compute an adjusted audio play rate, $r(t)$. After the audio is time-stretched ($a'(t)$), the current position in the unstretched audio, $a(t)$, is fed back to the input tracker to ensure that the audio remains synchronous to the user input.

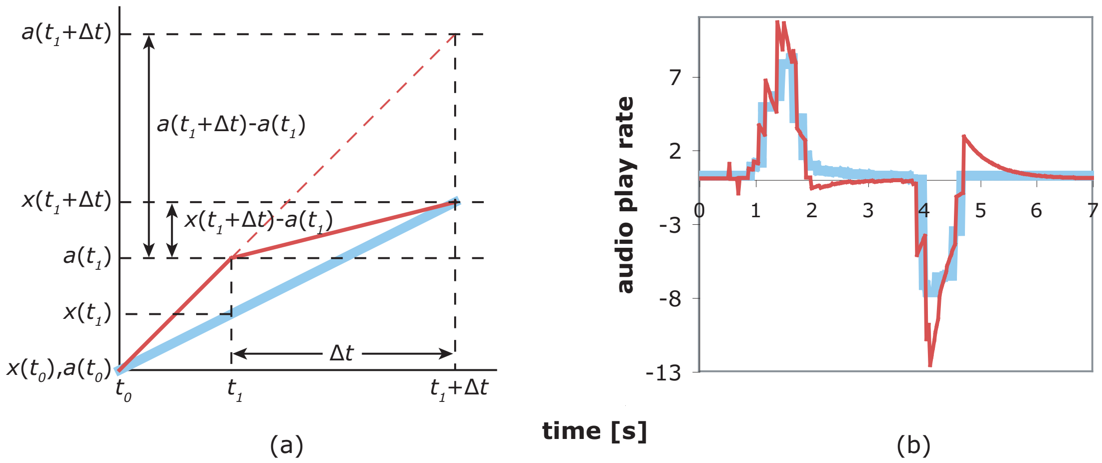

Figure 6: (a) Synchronization algorithm from [19]. (b) Effect of applying this algorithm on the input data shown in Figure 4. The thick blue line is the input velocity $v(t)$, and the thin red line is the adjusted play rate $r(t)$. The synchronized play rate overshoots the desired target at $t = 4.75$ sec, resulting in undesirable oscillations.

much we *want* the audio to advance and how much we *expect* it to advance over the interval Δt. Figure 6b shows the result of applying this algorithm to the user input shown in Figure 4, and we observe that:

- At $t = 4$ sec, the input device has stopped moving, but the latency in the velocity estimation results in the audio over-shooting the target position. Such an effect is disconcerting for the user, who expects the audio to stop at precisely the position specified, and not oscillate back and forth.
- The adjusted play rate $r(t)$ contains many spikes, due to the sudden changes in the input position and velocity.

To address the first problem, we first impose a condition that if the adjusted play rate does not occur in the direction of movement, then the adjusted play rate is set to zero. However, this condition does not guarantee synchronization. We observe that an instantaneous rate adjustment is given by $r_n(t_i) = \frac{x(t_i)-a(t_i)}{\epsilon}$, and is not affected by inaccuracies in $v(t)$; however, it is only valid for small values of ϵ, which usually results in even larger jumps to the play rate when the input position changes. Thus, we use this instantaneous rate adjustment only when its magnitude is smaller than $r_g(t)$.

To remove the spikes in the play rate, we introduce a "viscosity" parameter μ, which is used to adjust the catch-up interval Δt, and to smoothen the play rate adjustment. (2) becomes:

$$r_g(t_i) = \mu r_g(t_{i-1}) + (1-\mu) r_g(t_{i-1}) \frac{x(t_i) + v(t_i)\mu\Delta t - a(t_i)}{r_g(t_{i-1})\mu\Delta t} \quad (3)$$

Figure 7 shows the effect of two different values of μ on $r(t)$. Note that a higher viscosity setting increases the interval from when the user stops moving, to when the audio catches up; we use this interval as a measure of response time, and Figure 8 shows the effect of increasing μ on the response time. Early user feedback suggests that an appropriate value for μ depends on the precision of the input device (e.g., mouse vs. DJ turntable), the audio type (music vs. speech), and the application (editing vs. searching).

3.3 Audio Time-Stretching

The last step is to time-stretch the audio using the adjusted play rate, $r(t)$, and determine the current position in the audio timeline, $a(t)$, to feed back to the input tracker; this feedback is necessary for the tracker to function correctly [19, 20].

Constant pitch time-stretching algorithms can be roughly classified in two categories: time domain and frequency do-

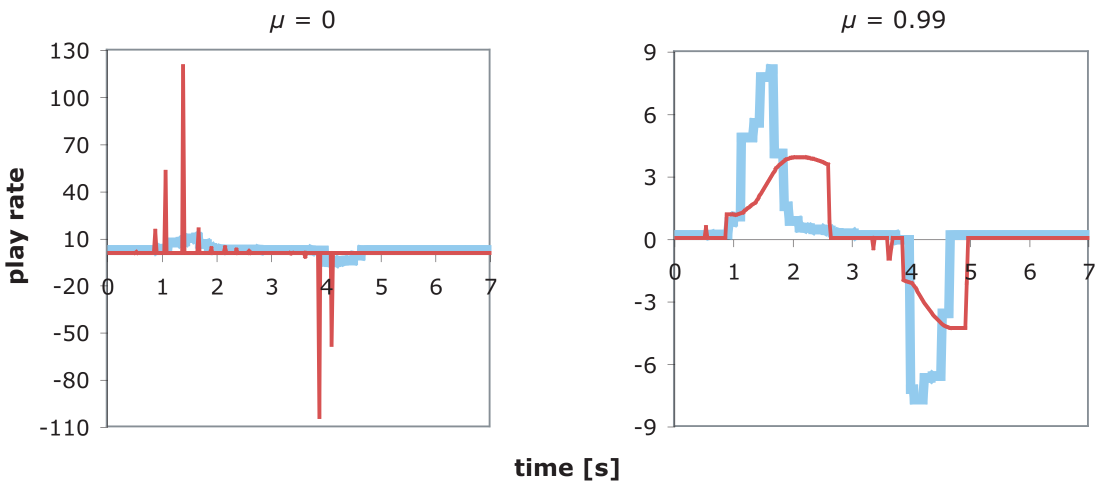

Figure 7: Effect of viscosity μ on the adjusted play rate $r(t)$ (thin red line): the play rate is smoother as μ increases. The thick blue line is the input velocity $v(t)$.

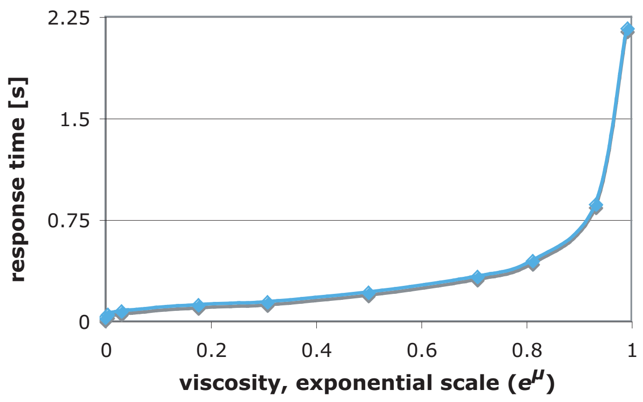

Figure 8: Effect of μ on response time.

main. Time domain techniques such as time domain harmonic scaling (TDHS) [22] and waveform similarly overlap-add (WSOLA) [27] are relatively computationally inexpensive (certain models of digital answering machines use a variant of these algorithms to support time-stretched playback of recorded messages), and adequate for structurally simple, monophonic audio signals such as speech. However, such algorithms are inadequate for polyphonic audio signals such as music. Moreover, time domain algorithms are generally limited to stretch factors of ±20% before the resulting audio artifacts become audibly disturbing.

Frequency domain algorithms, usually based on the phase vocoder [9], on the other hand, are capable of time-stretching at a wider range of stretch factors. Recent research in this area has also addressed many of the reverberation and transient-smearing artifacts typical of phase vocoder algorithms [6, 16, 18, 24], making it an attractive choice for arbitrary audio signals, despite the increased computational cost (usually a factor of ten or more, compared to time domain algorithms).

An increasing number of multimedia software frameworks support these frequency domain constant pitch audio time-stretching in real time: Apple's *TimePitch* (part of the Core Audio[6] framework), and zplane.development's *élastique*[7] (a stand-alone time-stretching library) are two such examples. Unfortunately, neither of these time-stretchers preserve the mapping between the input and time-stretched audio; that is, it is not possible to determine $a(t)$ from the time-stretched audio timeline, $a'(t)$, information which is required to synchronize the audio to the user input. Moreover, this information cannot be retroactively determined by observing the input/output behavior of the time-stretching "black boxes" for two reasons:

1. The algorithms used for constant pitch time-stretching quantize the input play rate; while small (usually less than 0.1%), these errors will accumulate, resulting in as much as 60 ms of error per minute of audio.

2. The algorithms require a certain amount of buffering of input samples to perform processing. Unfortunately, this buffer creates a bounded but indeterministic delay from input to output that depends on, amongst other parameters, the current play rate. Thus, the input audio samples that were requested when rendering the time-stretched audio samples may not actually be used to render audio until some time later in the future.

TimePitch and *élastique* are, furthermore, limited in their play rates: *TimePitch* supports play rates between 0.25 and 8 times normal speed, and *élastique* rates between 0.1 and 10 times normal speed. In DiMaß, however, the audio play rate frequently drops below 0.1 and rises above 20 times normal speed, or more. Existing studies have shown that audio such as speech time-stretched beyond two or three times normal speed is beyond intelligibility [2], and so the need to support such extreme play rates may be questioned. However, a text document zoomed out beyond legibility still provides strong visual cues, such as paragraph length or color. Similarly, audio time-compressed beyond intelligibility can still provide identifiable speaker and tone changes for speech, or genre cues for music. For searching tasks where the location of the desired target is approximately known (e.g., the

[6]http://developer.apple.com/audio/
[7]http://www.zplane.de

latter half of a lecture), supporting such extreme play rates allows the user to quickly skim past the unneeded sections, but still have the audio feedback to maintain context.

The Semantic Time Framework (STF) is a software library we created for developing multimedia applications with time-based interaction. It preserves the input to output audio time mapping using *semantic time*; semantic time defines time in more meaningful units such as music beats, as opposed to audio samples and video frames (analogous to using "blocks" to refer to distance in North American cities, instead of meters). Our first implementation of STF was used to simplify the design of interactive orchestral conducting systems [20]. For DiMaß, we enhanced STF to support arbitrary forwards and backwards stretch factors. Backwards time-stretching is achieved by simply reversing the order of the audio samples. While more sophisticated backwards time-stretching approaches have been proposed for speech [4, 12], it is unclear how such schemes would apply to other types of audio, such as music, and a further analysis of backwards time-stretching of audio would have been beyond the scope of this work. Constant pitch time-stretching is performed using PhaVoRIT, our Phase Vocoder for Real-Time Interactive Time-Stretching [16].

4. IMPLEMENTATION

We incorporated DiMaß into *Beat Tapper*, a tool we developed for tagging musical recordings with beat metadata[8]. Using *Beat Tapper*, users mark beats in an audio file by "tapping along" while the audio is playing, and manually fine-aligning them afterward using a visual representation of the audio waveform (see Figure 1). *Beat Tapper* can also play these beats back synchronously with the music as audible "taps". The audio play rate can be adjusted dynamically, and using DiMaß, users can quickly skim through the audio, or scrub around a specific area of the waveform to help them fine-align a beat. We also included variable pitch time-stretching using resampling, as an alternative to PhaVoRIT.

It is important to note that *Beat Tapper* is just one possible implementation of the DiMaß technique. *Beat Tapper* demonstrates how DiMaß simplifies a fine-grained searching task (looking for beats). We are currently examining other applications that would benefit from an implementation of the DiMaß algorithms presented in this paper. For example, imagine an "enhanced iPod" where the scroll wheel can be used for audio skimming (see Figure 2), but with the addition of time-stretched audio feedback. Such an implementation would assist with searching and skimming through podcasts, or finding songs in certain concert music recordings that, unlike traditional CDs, have not been split into tracks by song.

5. EARLY USER FEEDBACK

We tested DiMaß with seven users (six students and one professional) using *Beat Tapper*, with the aim of obtaining qualitative feedback on DiMaß. Users used either a mouse or a SMART Actalyst[9] touch screen to interact with *Beat Tapper*. We tested DiMaß with both music, using a Vienna Philharmonic recording of *Blue Danube Waltz* by Johann Strauss, and speech, using an excerpt from an audio book of *Ein Wintermärchen* by Heinrich Heine. Users were encouraged to experiment with the viscosity setting, and with constant and variable pitch time-stretching.

Users readily grasped the directness of grabbing on to the waveform and shuffling it around – two users even attempted to manipulate the waveform with two hands on the touch screen (but failed due to sensor limitations of the Actalyst device, which does not support two-handed interaction). One user commented on how he expected the waveform to move with some "inertia", and thus continue to move even after he released the waveform. As expected, all users vastly preferred constant pitch time-stretching to variable pitch, where the audio resembles "a broken record"; such audio also becomes disturbing with rapid movements due to the increased pitch. Most users preferred a higher viscosity setting, between 0.5 and 0.9, as it produced smoother and more pleasant-sounding audio.

Two of the users thought the technique would be useful when going through audio podcasts of lectures, for example, when studying for an exam. One of the users, who works for the local student radio station, observed how a similar tool would help simplify his task of cutting audio recordings for radio shows.

We are currently preparing for a more formal user study aimed at obtaining quantitative results comparing DiMaß to existing audio scrubbing and skimming techniques for the tasks outlined above, and in the introduction of this paper.

6. FUTURE WORK

We are currently pursuing the following directions in our work for DiMaß:

DiMaß implementations: We have, thus far, implemented DiMaß as part of the *Beat Tapper* application, which uses the mouse for audio scrubbing. We are examining how DiMaß can be used in other applications and input devices, such as the scroll wheel in the iPod portable digital media player for searching through podcasts.

Enhanced interaction techniques: We are currently examining how DiMaß can be improved by, for example, combining it with existing speech skimming techniques proposed by Arons [4]. There is also an emerging trend in recent human-computer interaction research of building "organic" user interfaces, where the system behavior is inspired by the natural laws of physics, biology, and human cognition [1, 10]. One could imagine, for example, applying physical kinetics to DiMaß and allow users to "toss" the audio waveform in *Beat Tapper* in either direction with some inertia. Similarly, we could model the relationship between the cursor position and waveform position as a mass and spring system, replacing the current "viscosity" setting with an "elasticity" setting.

Formal user evaluation: While preliminary user feedback on DiMaß has been positive, we intend to conduct more formal user studies to obtain quantitative results comparing DiMaß to existing techniques for audio scrubbing and skimming.

7. CONCLUSIONS

We presented DiMaß, a direct manipulation technique for interacting with the timeline of digital audio. DiMaß consists of (1) motion estimation to compute an instantaneous input position and velocity for relative input at irregular

[8] *Beat Tapper* is available for download at: `http://media.informatik.rwth-aachen.de/dimass.html`
[9] http://www.smarttech.com/actalyst/

and infrequent intervals; (2) input tracking to adjust audio play rate based on the estimated input position and velocity; and (3) audio time-stretching to arbitrarily adjust the play rate of digital audio, both forwards and backwards. We introduced an improved algorithm to synchronize audio to user input; this algorithm includes a “viscosity” setting that smoothens the adjusted play rate, but with a small impact on responsiveness. Informal user tests showed that even though our algorithm is capable of almost zero-latency synchronization at a low viscosity setting, users preferred a higher setting as it produced more pleasant-sounding audio. We hope our work on DiMaß will make scrubbing and searching through audio for both content producers and consumers a more efficient and pleasant experience.

8. ACKNOWLEDGEMENTS

The authors would like to thank Ken Greenebaum, David Holman and Rafael Ballagas for their feedback on an early paper draft, and the users who participated in the user tests.

9. REFERENCES

[1] A. Agarawala and R. Balakrishnan. Keepin’ it real: Pushing the desktop metaphor with physics, piles and the pen. In *Proceedings of the CHI 2006 Conference on Human Factors in Computing Systems*, pages 1283–1292, Montréal, Canada, April 2006. ACM Press.

[2] A. Amir, D. Ponceleon, B. Blanchard, D. Petkovic, S. Srinivasan, and G. Cohen. Using audio time scale modification for video browsing. In *Proceedings of the 33rd Annual Hawaii International Conference on System Sciences*. IEEE, 2000.

[3] D. P. Anderson and G. Homsy. A continuous media I/O server and its synchronization mechanism. *IEEE Computer*, 24(10):51–57, 1991.

[4] B. Arons. SpeechSkimmer: a system for interactively skimming recorded speech. *ACM Transactions on Computer-Human Interaction (TOCHI)*, 4(1):3–38, 1997.

[5] T. Beamish, K. Maclean, and S. Fels. Manipulating music: multimodal interaction for DJs. In *Proceedings of the CHI 2004 Conference on Human Factors in Computing Systems*, pages 327–334, Vienna, Austria, April 2004.

[6] J. Bonada. Automatic technique in frequency domain for near-lossless time-scale modification of audio. In *Proceedings of the ICMC 2000 International Computer Music Conference*, Berlin, 2000. ICMA.

[7] S. K. Card, W. K. English, and B. J. Burr. Evaluation of mouse, rate-controlled isometric joystick, step keys, and text keys for text on a CRT. *Ergonomics*, 21:601–613, 1978.

[8] S. K. Card, J. D. Mackinlay, and G. G. Robertson. A morphological analysis of the design space of input devices. *ACM Transactions on Information Systems*, 9(2):99–122, 1991.

[9] J. L. Flanagan and R. M. Golden. Phase vocoder. In *Bell Systems Technical Journal*, volume 45, pages 1493–1509, November 1966.

[10] D. Holman, P. Stojadinović, T. Karrer, and J. Borchers. Fly: an organic presentation tool. In *Extended Abstracts of the CHI 2006 Conference on Human Factors in Computing Systems*, pages 863–868, Montréal, Canada, April 2006. ACM Press.

[11] H. Honing. From time to time: The representation of timing and tempo. *Computer Music Journal*, 25(3):50–61, 2001.

[12] W. Hürst, T. Lauer, and C. Bürfent. Playing speech backwards for classification tasks. In *Proceedings of the ICME 2005 International Conference on Multimedia and Expo*. IEEE, July 2005.

[13] W. Hürst, T. Lauer, C. Bürfent, and G. Götz. Forward and backward speech skimming with the elastic audio slider. In *Proceedings of the 19th British HCI Group Annual Conference*, Edinburgh, Scotland, 2005.

[14] E. L. Hutchins, J. D. Hollan, and D. A. Norman. Direct manipulation interfaces. *Human-Computer Interaction*, 1:311–338, 1985.

[15] D. Jaffe. Ensemble timing in computer music. *Computer Music Journal*, 9(4):38–48, 1985.

[16] T. Karrer, E. Lee, and J. Borchers. PhaVoRIT: A phase vocoder for real-time interactive time-stretching. In *Proceedings of the ICMC 2006 International Computer Music Conference*, New Orleans, USA, November 2006. ICMA, In Print.

[17] H. Kitamura. New algorithms and techniques for well-synchronized audio and video streams communications. In *Proceedings of the 6th International Conference on Computer Communications and Networks*, pages 214–219. IEEE, 1997.

[18] J. Laroche and M. Dolson. Improved phase vocoder time-scale modification of audio. *IEEE Transactions on Speech and Audio Processing*, 7(3):323–332, 1999.

[19] E. Lee. *Audio Anecdotes III: Tools, Tips, and Techniques for Digital Audio (Ken Greenebaum and Ronen Barzel, Eds.)*, chapter Dynamic Synchronization: Drifting Into Sync. AK Peters, 2006. In Print.

[20] E. Lee, T. Karrer, and J. Borchers. Toward a framework for interactive systems to conduct digital audio and video streams. *Computer Music Journal*, 30(1):21–36, 2006.

[21] T. D. C. Little and F. Kao. An intermedia skew control system for multimedia data presentation. In *Proceedings of the 3rd International Workshop on Network and Operating System Support for Digital Audio and Video*, 1993.

[22] D. Malah. Time-domain algorithms for harmonic bandwidth reduction and time scaling of speech signals. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 27(2):121–133, 1979.

[23] D. Murphy, T. H. Andersen, and K. Jensen. Conducting audio files via computer vision. In *Gesture Workshop 2003*, volume 2915 of *Lecture Notes in Computer Science*, pages 529–540, Genova, 2003. Springer.

[24] A. Röbel. Transient detection and preservation in the phase vocoder. In *Proceedings of the ICMC 2003 International Computer Music Conference*, pages 247–250, Singapore, 2003. ICMA.

[25] B. Shneiderman. *Designing the User Interface*. Addison Wesley, 3rd edition, 1997.

[26] R. Sussman and J. Laroche. Application of the phase vocoder to pitch-preserving synchronization of an

audio stream to an external clock. In *Proceedings of the 1999 IEEE Workshop on Applications of Signal Processing to Audio and Acoustics*, pages 75–78, New York, October 1999. IEEE.

[27] W. Verhelst and M. Roelands. An overlap-add technique based on waveform similarity (WSOLA) for high quality time-scale modification of speech. In *Proceedings of the ICASSP 1993 International Conference on Acoustics, Speech, and Signal Processing*, volume II, pages 554–557. IEEE, 1993.