Effective Browsing of Long Audio Recordings

Camille Goudeseune
Beckman Institute
University of Illinois at Urbana-Champaign
Urbana, Illinois, 61801 USA
cog@illinois.edu

ABSTRACT

Timeliner is a browser for long audio recordings and features that it derives from such recordings. Features can be either signal-based, like spectrograms, or model-based, like categorical classifiers.

Unlike conventional audio editors, Timeliner pans and zooms smoothly across many orders of magnitude, from days-long overviews to millisecond-scale details, with zero latency, zero flicker, and low CPU load. Also, to suggest which details are worth zooming in to examine, Timeliner's agglomerative hierarchical caches propagate feature-specific details up to wider zoom levels. Because these details are not averaged away, "big data" can be browsed rapidly and effectively. Several studies demonstrate this.

Categories and Subject Descriptors

H.5.1 [**Information Interfaces and Presentation**]: Multimedia Information Systems—*audio input/output*; H.4.3 [**Information Interfaces and Presentation**]: Communications Applications—*information browsers*; H.5.5 [**Information Interfaces and Presentation**]: Sound and Music Computing—*signal analysis, synthesis, and processing*

General Terms

Algorithms, Human Factors

Keywords

Deep Zoom, Big Data, Mipmap

1. INTRODUCTION

Human audition of acoustic events often outperforms automatic detection, but it fails for long recordings. High-speed playback helps only slightly: even continuous speech can rarely be understood faster than twice normal speed [1]. Long recordings are fatiguing, because uninterrupted alertness is needed to notice brief transient events.

IMMPD'12, November 2, 2012, Nara, Japan.

However, human vision helps. Visualizations at a range of temporal scales can efficiently eliminate long uninteresting intervals. Synchronizing such visualizations with audio playback then lets a user search visually (that is, rapidly), only occasionally taking time to briefly listen to intervals deemed visually interesting. This multiscale, multimedia approach is used by an audio browser, Timeliner, whose design and implementation this paper describes. Timeliner itself is open-source and available for download [8].

Timeliner's source data is an audio recording, hours or even days long. It displays features derived from this recording as stacked images, horizontally synchronized with the waveform and a time axis (Fig. 1). Features include spectrograms [5], Mel-frequency cepstral coefficients (MFCC's) [16], spectrograms transformed to reduce the visual salience of non-anomalous events [14], and output log likelihoods from event classifiers [24, 25].

The features themselves are rendered purely visually, without audio, because aural search is comparatively slow. In this multimedia context, the most effective role for audio presentation is brief testing of hypotheses that were formed while searching visually. The source recording itself is of course the ground truth for such tests, but audio-presented features could help as well (although this is not currently implemented). For example, in speaker diarization, the visual presentation of a categorical classifier for a particular person could be augmented with audio resynthesis tailored to extract just that person's voice.

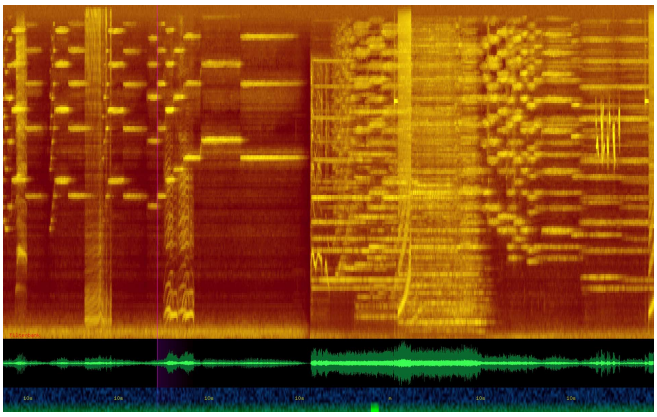

Figure 1: Timeliner's display: features (here, a single spectrogram), waveform (green), and time axis (blue).

Initially, Timeliner presents the whole recording to the user, who can then zoom in to interesting areas to rapidly find even very brief anomalous segments. Commodity mobile computers running Timeliner smoothly zoom across six orders of temporal magnitude.

Conventional audio editors compute the appearance of each horizontal extent of a pixel by undersampling data from the corresponding time interval. But in Timeliner, such intervals might be minutes long. During a fast pan or zoom, such extreme undersampling would produce flickering strong enough to entirely obscure the data. This would negate the benefit of such continuous pan-and-zoom gestures, which are natural to handheld devices and to be contrasted with formulating database queries at a keyboard.

To restore smoothness, Timeliner builds a multiscale agglomerative cache for the recording and for each derived feature. Such a cache efficiently reports the minimum, mean, and maximum data values found during a temporal subinterval, either for scalar data such as the recording itself, or for vector data in HTK format [23]. Final rendering assigns a color to each texel, using a color map particular to that kind of data (see Sec. 3.3).

The cached vector-format data is itself summarized pyramidally and moved to graphics memory. This yet again improves performance by freeing main memory to store longer recordings, and by exploiting the GPU. Even at full-screen resolution, the relatively slow CPU and main memory bus of mobile computers running OpenGL ES or its derivative WebGL are only lightly loaded.

Timeliner runs natively on Ubuntu Linux. It requires two external software packages, HTK 3.4.1 and, to display features derived from pre-trained neural networks, the software suite QuickNet [7].

File parsing and interfacing to external utilities are both implemented in the scripting language Ruby. Numerical computation uses C++, to both increase speed and conserve memory. Graphics are rendered with OpenGL and its utility toolkit GLUT.

1.1 Long recordings on mobile devices

Handheld audio recorders that store recordings on inexpensive memory cards originally marketed for cameras have become common, but they need some persuading to produce useful multi-hour recordings. Recording may need to be restarted when the data file reaches 2 GB or 4 GB on a FAT16- or FAT32-formatted card. Automatic power-down may need overriding, and short battery life may even demand an external power supply.

Finally, devices not dedicated to audio or lacking an advanced operating system often suffer from an automatic gain control that cannot be disabled. But this is rarely a problem when the mobile device doing the recording is the same one that will be used for browsing with Timeliner.

2. PREPROCESSING

A Ruby/C++ preprocessor prepares data for the C++ browser. This lets the latter start very quickly, a convenience for consecutive or even simultaneous browsing sessions. If the preprocessing happens immediately after the unavoidably slow act of recording, its own duration is negligible.

The preprocessor computes features from the recording with the help of external packages. These features, and the recording itself, are then read as memory-mapped files and then rewritten as serialized data, that is, pre-parsed data structures that the browser will load directly from disk. The preprocessor is given the following:

- a source recording in a format understood by the audio utility `sox`, such as .mp3 or .wav
- the directory that will contain the serialized files
- how many channels to use when computing spectrogram and MFCC features
- optionally, a directory of short easter-egg sound files, named such because the preprocessor "hides" them in the source recording, for tasks that measure Timeliner's effectiveness (see Sec. 6).

For uniform graphical presentation, the preprocessor also normalizes each feature's data to the unit interval.

Because HTK incorrectly rounds sampling duration to the nearest exact multiple of 100 ns, common sampling rates other than 8 kHz or 16 kHz (exactly 1250 or 625 multiples of 100 ns, respectively) result in a drift in reported time. This may be tolerable for recordings lasting a few seconds, but not for a few days. The preprocessor therefore simply resamples the source recording at 16 kHz, a surprisingly common sample rate for speech processing [13, 15].

2.1 Vector-format features

Features are computed from the source recording with a sliding Hamming window. The window's size and skip can be tuned for particular kinds of recorded material. Each feature is written to disk in HTK format [23].

Spectrogram features are computed with HTK's filterbank feature. Saliency-maximized spectrograms convolve a standard spectrogram with a saliency-optimized filter computed by an external Matlab script [14].

Daubechies wavelet features use the Gnu Scientic Library (GSL) implementation. At the specified sampling rate, 32-sample intervals from the recording are convolved with a Hamming window and passed to GSL. The result from GSL is then stored in HTK format.

Features that are based on neural networks, such as categorical classifiers, are computed with tools from the software package SPRACHcore [6], in particular the utilities `feacat` and `qnsfwd` from its neural net code QuickNet [7]. First, `feacat` builds a 'pad' file from the source recording. Then, using a file containing weights from a pretrained neural network, `qnsfwd` converts the pad file to an 'act' file. From each line of this file, the floating-point weights of features are extracted and stored in HTK format.

One specialized feature is an unusual classifier, for use with easter eggs (see Sec. 6). If easter eggs are specified, they are combined with the source recording. When an egg is placed, its audio signal comes from a uniformly randomly chosen source egg, and its amplitude is also scaled randomly. The number of eggs placed is chosen to fit a specified density of eggs per unit time, usually no more than a few eggs per minute. Eggs are distributed randomly and uniformly without overlap, subject to the constraint that at least a specified minimum duration separates consecutive eggs.

For convenience when testing, an oracular "perfect" classifier indicates the locations of easter eggs. At each sample period this feature's value is 1.0 or 0.0, as that period does or does not contain an easter egg.

3. AGGLOMERATIVE CACHE

Recall that Timeliner's agglomerative caches are used to efficiently extract summary values, such as the maximum or the mean, from a subinterval of values from either the recording or features derived from it.

3.1 Construction

The cache is built as a rooted binary tree, starting from the leaves. A leaf node corresponds to one sample of data, and stores that sample's value. In the next layer of the tree, each node's payload stores the minimum, mean, and maximum of the two values of its children. The payload also stores the number of samples that its children represent (in this case, two). This pattern propagates up to the root node.

Elementary arithmetic computes a parent node's payload from its children's: the parent's minimum is the minimum of its children's minima, *etc.* It is only worth noting that the mean can be computed only if the payload includes the number of samples. (To avoid roundoff error, the mean is calculated in double precision. All other calculation and storage uses single precision to conserve memory.) When a layer in the tree has an odd number of nodes, the final node in the next layer up of course gets only one child, from whom its payload is copied verbatim.

Memory is significantly conserved by giving the leaves a payload with only one value, instead of the four used by non-leaf nodes. This single value is vacuously its own minimum, mean, and maximum. Similarly, the number of samples is vacuously one and thus need not be stored.

To conserve even more memory, but at the cost of coarser temporal resolution, leaves may store more than one sample. Storing n consecutive samples in each leaf reduces the number of nodes (and hence the cache's memory footprint) n-fold, but limits zoom-in to a resolution n times coarser. This trade-off is particularly useful for the cache of the displayed waveform, if visible submillisecond detail would not assist a browsing task.

For vector-valued data such as a spectrogram's coefficients during a given sampling period, or even just stereo source recording, each node can store not just one payload but a whole set of payloads, one for *each* element of the vector.

3.2 Querying

Given any time interval, that is, a contiguous subset of samples (a duration), the agglomerative cache returns that interval's payload, as generalized from the definition of a node's payload. Unsurprisingly, the cache does so by combining payloads from nodes. The number of nodes visited is proportional to the depth of the binary tree. This is of course logarithmic with respect to the recording's total duration, and independent of the interval's own duration, so queries are fast.

Combining payloads starts at the root node. At each node visited, if the node's time interval is *disjoint* with the query interval, it is discarded. If the node's interval is *contained* within the query, it is kept. Finally, if the node's interval overlaps *partially* with the query, testing continues with the node's children, down to the leaves. (Leaves are detected from their uniform depth in the tree, because this has no space penalty and negligible time penalty.) As this recursion unwinds back up the tree, payloads are combined with the same elementary arithmetic that was used to construct the cache in the first place.

3.3 Color maps

To reduce calling overhead, the functions that query the cache return not just individual payloads but entire arrays of them. These payloads may also be immediately convolved with a color map, thereby returning an array of colors ready for rendering with an OpenGL texture map.

A general color map converts an individual min-mean-max payload to a hue, saturation, and value (HSV), which in turn is converted to a red-green-blue triple. The extra step is warranted because HSV color space better matches human visual perception. (Hue means "rainbow color," saturation means lack of grayness, and value means brightness.)

In practice, Timeliner constrains a color map $f : \mathbb{R}^3 \to \mathbb{R}^3$ by decomposing it into two parts, $f : \mathbb{R}^3 \to \mathbb{R}^1 \to \mathbb{R}^3$. The first part computes a weighted sum of the payload's components. The second part uses that sum to interpolate between two endpoints of a color gradient. HSV variations between the endpoints then determine those variations in f as a whole. Because this decomposition constrains the image of f to a straight line, these three variations are not truly independent. But the decomposition also quadruples how much data can be browsed, enabling longer duration, finer temporal resolution, finer frequency resolution, and combinations thereof (see Sec. 4.1.3).

For example, when defining a color map for a spectrogram fine enough to resolves individual sinusoidal components, a weighted sum that uses only the maximum best reveals these sinusoids. At coarser frequency resolutions, some of the maximum's weight should transfer to the mean, to avoid confusing actual sinusoids with narrowband noise.

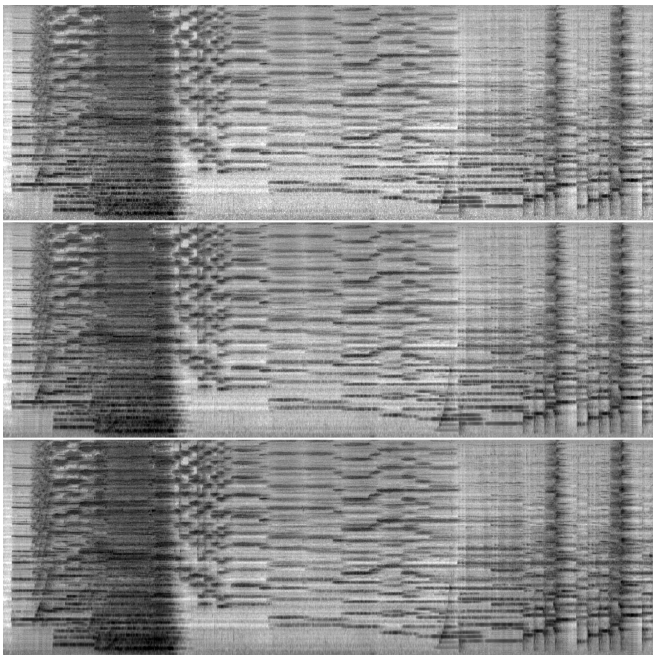

Figure 2: Monochrome spectrogram of one minute of orchestral music (darker areas are louder). Payload weights, from top to bottom: 100% minimum, 100% mean, and 100% maximum.

Conventional pure-mean spectrograms can be assisted by slightly weighting the maximum. This is because everyday spectra are asymmetric, with more peaks than notches. Fig. 2 demonstrates this: the pure-maximum weighted color map has more prominent dark amplitude peaks than the pure-mean one, while the pure-minimum one suffers from both reduced visual contrast and white-dot clutter. Conversely, notch-dominated absorption spectra such as those produced by time-domain spectroscopic optical coherence tomography [22] demand a mean-with-*minimum* weighting.

Similarly, classifiers of anomalous events that represent presence as rare high values and absence as common low values, such as the easter egg oracle, should ideally emphasize only maxima (Fig. 3, top). This is because a pure maximum weighting preserves the visibility of even the briefest events at the widest zooms. If the classifier suffers from false positives, the maximum can be diluted with the mean, which averages away such noise (but also detail: Fig. 3, middle and bottom).

Traditional printed speech spectrograms or 'sonograms' use endpoints of pure black and pure white (again, Fig. 2). Endpoints whose hues differ let the resulting image allude to heat and cold (Fig. 1), highlights and neutrality (Fig. 3), or other artistic metaphors. Endpoints with similar hues are useful when many features are displayed simultaneously, as with a bank of categorical classifiers: this reserves hue for distinguishing between categories, a task it is particularly good at.

4. GRAPHICAL RENDERING

Timeliner's unusual attributes force all of its displays—the recording's waveform, the features derived from the recording, and even the time axis itself—to be rendered in unusual ways. These methods are elaborated here.

4.1 Mipmaps of derived features

Instead of expensively computing a texture map from the feature data many times per second, Timeliner precomputes texture maps just once. Because it cannot compute an infinite continuum of these, it computes them at only a finite number of resolutions. At any given zoom level, the two texture maps closest to that resolution are interpolated to produce the final display. Each texture map is one "level" of a mipmap, which is a venerable anti-aliasing technique for texture maps applied to three-dimensional scenes. Timeliner abuses this technique in a merely two-dimensional situation. (Like Timeliner's agglomerative cache, the mipmap was invented to eliminate flicker induced by subsampling [20].) As a performance bonus, the inter- and intra-level interpolation of textures is calculated by the GPU instead of the CPU.

Because OpenGL's two-dimensional mipmaps cannot scale in only one of two dimensions, Timeliner resorts to more seldom used one-dimensional mipmaps. (A possible alternative is the anisotropic mipmap called a ripmap [11, pp. 61–62], but in this specialized application it uses memory inefficiently [12]. At any rate, neither OpenGL nor commodity GPUs implement ripmaps.) These mipmaps tile to cover the width of the full data. Each tile is as wide as possible, to reduce overhead in graphics memory.

4.1.1 *Non-optical mipmap levels*

Conventionally, each level of a mipmap is derived from its next higher-resolution level, often by just averaging texels.

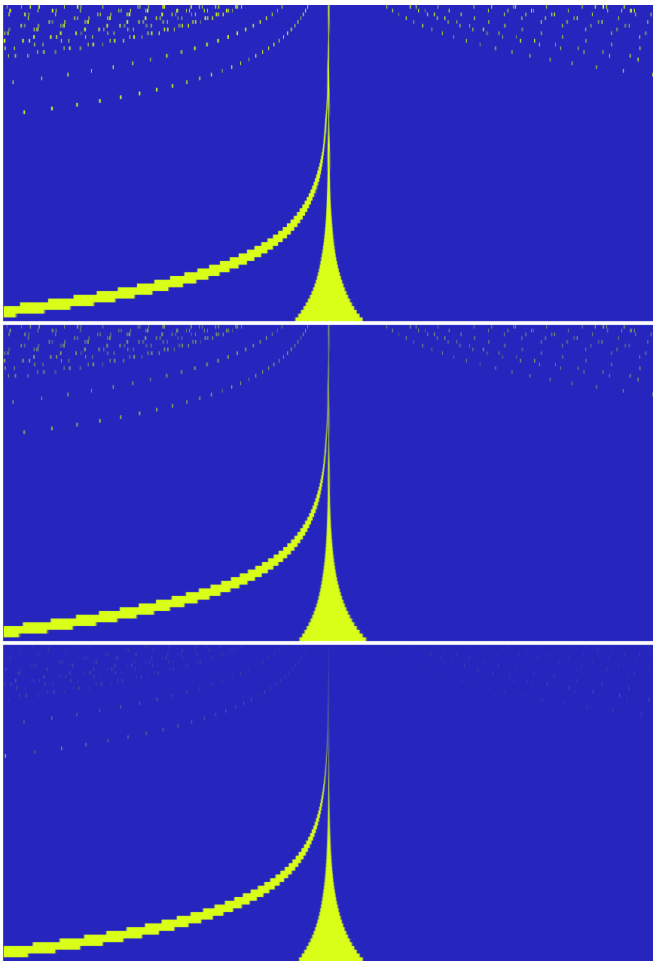

Figure 3: Thin slices of screenshots of an event classifier zoomed from 8 hours to 10 seconds. The 48 (yellow) events have duration 1 to 4 seconds. From top to bottom: 100% maximum; 50% maximum and 50% mean; 100% mean.

Advanced filters and windows sometimes elaborate this [3, 4], but the basic mechanism of computing one level from the previous one remains. Such a mechanism fails here because the source data is not optical. In a deliberate violation of optical principles, we wish to preserve the visibility of interesting details across all resolutions (see Sec. 3.3). Thus, each level of the mipmap must be computed directly from the original data. Each texel, at each level of the mipmap, spans a particular interval of data. That data, efficiently summarized into that interval's payload by the agglomerative cache, then yields that texel's color.

4.1.2 *Information rate*

Because a mipmap interpolates between a finite number of zoom levels, it merely approximates the output of the agglomerative cache. But this subtle quantization noise is practically invisible, especially when actively panning and zooming. The corresponding advantage over non-mipmap rendering is typically a tripled information rate, measured in texels per second (on commodity mobile computers with so-called dedicated graphics, as opposed to less expensive

"integrated graphics" subsystems that steal main memory for the GPU).

4.1.3 *Conserving graphics memory*

To quadruple how much data can be browsed, each texel is stored in only 8 bits, instead of in the more conventional 32-bit red-green-blue format (GPUs internally pad 24 bits of RGB to 32). More precisely measuring information rate in bits per second rather than texels per second, reducing each texel's size from 24 visible bits of color to only 8 would seem to reduce the information rate threefold.

In practice, however, the information rate is reduced less severely. First, the payload's weighted sum often has one or more zero weights: the whole payload is not exploited. Second, because the payload's three elements are somewhat correlated in everyday circumstances, squeezing the payload into 8 bits rather than 24 does not hide entirely two thirds of the raw information about its underlying statistical distribution. Third, smaller texels generally increase the GPU's performance. One quarter as many bytes need to be fetched from the GPU's memory, so its internal caches are better exploited while rendering mipmaps. This improved performance reveals itself as higher resolution and frame rate. In other words, fewer *bits* per *texel* is countered by more *texels* per *frame* and more *frames* per *second*. Their net product, bits per second, may even increase rather than decrease.

4.2 Audio waveform

Even though a displayed waveform reveals little more than peak amplitude, its sheer familiarity justifies devoting part of the screen to it. To extract slightly more information from this part of the screen, then, the waveform's amplitude auto-scales to match what is currently on screen. The unscaled display is bright, but possibly of near-zero amplitude. Behind that, the scaled display is dimmer (Fig. 4). Dimness varies with scaling, to prevent unusually large blobs from becoming excessively salient. Also, to reduce flicker artifacts when quickly zooming or panning, the auto-scaling is slew-rate limited. (The rates for increasing and for decreasing the scaling differ, like an audio dynamic range compressor's attack and release controls.)

Recall that with recording-derived features, the agglomerative cache is used only to construct mipmaps. For waveform display, however, the browser uses the cache without any such indirection because, again, two-dimensional mipmaps cannot scale in only the horizontal dimension.

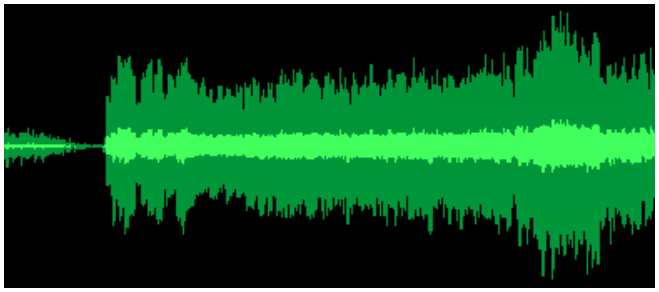

Figure 4: Detail of waveform display. The bright waveform's amplitude remains nominal, while the dim waveform behind it auto-scales to fill the allocated on-screen height.

Figure 5: Detail of time axis display.

4.3 Time axis and warnings

The bottom of the screen shows a temporal axis, but without textual and numerical clutter (Fig. 5) that merely distracts while actively browsing. Abbreviated units such as `d` for days or `10s` for ten seconds hint at the display's zoom level. When zoom level changes, some units fade out while others fade in.

Behind these abbreviations, a faint white-noise texture provides optic flow during pan and zoom, even if on-screen flow is momentarily absent elsewhere as when displaying an interval of silence. This "infinite" texture is implemented as multiple layers of a single texture, which crossfade in opacity like the components of a Shepard-Risset glissando crossfade in amplitude [19]. (With a Perlin noise source, such infinite noise textures work in higher dimensions [2]; even arbitrary images can produce infinitely zoomable textures [9].)

To represent the fraction of the whole recording that is on-screen, the noise texture is overlaid with an indicator like the thumb of a scrollbar (the short bright-green patch in Fig. 5; also barely visible at bottom middle in Fig. 1).

If a pan or zoom-out tries to go past the start or end of the recording, a red flash at the start or end (or both) alerts the user of the limit being hit. Similarly, an attempt to zoom in beyond the data's resolution produces a yellow warning flash at the left and right edges of the screen.

5. OPERATION

Pan and zoom can be done with either the mouse or keyboard, whichever a user finds most familiar. However, the keyboard is faster for two reasons. First, panning by "pawing" the mouse involves wasted backstroke motion. Second, left-hand pan and zoom frees the right-hand mouse to have the sole task of aiming at particular positions to listen to sounds. The user's gaze too is freed from hunting for keys, because all the keys lie under the left hand without needing to aim the fingers, in the WASD layout that became dominant for mouse-keyboard real-time games in the mid-1990s. Users familiar with a mouse's scroll wheel for zooming in and out can use that to zoom, too. On a mobile device's touchscreen, pan and zoom fit the familiar flick and pinch gestures respectively.

To clarify optic flow despite noisy input gestures, both pan and zoom are slightly smoothed with a simple IIR filter, whose only sophistication is that it adapts to Timeliner's frame rate for identical temporal behavior independent of the hardware's capabilities.

5.1 Audio playback

After positioning a cursor (the purple vertical hairline in Fig. 1) with a mouse click, tapping the spacebar starts playback from that cursor. Touchscreens need no separate positioning gesture: one tap both positions and commands playback.

A subsequent tap usually stops playback. But if the user repositions the cursor during playback, that subsequent tap immediately skips playback to that position, without an intervening pause. On a touchscreen, the equivalent "skip

ahead" shortcut demands a different gesture such as a two-finger tap. Not having to explicitly stop before each start halves the number of taps, when briefly listening at many places as Timeliner's search strategy encourages.

During playback, a specialized playback cursor appears and progresses rightwards. A mutex synchronizes its movement with audio playback.

Audio is loaded as a memory-mapped file, for similar benefits as when the preprocessor uses this technique.

5.2 Annotations

To mark the position of an interesting sound, the user moves the mouse to the sound's position and hits a key. Touchscreens use either a double-tap or a tap-and-hold. The annotation is confirmed by a brief flash fading to a vertical hairline at that screen position. Another keystroke or tap-and-hold lets the user undo the last annotation.

Upon exit, Timeliner logs the annotations to a text file.

6. EVALUATION

A central premise of Timeliner is that visual search for interesting sounds is faster and more accurate than aural search. Two scenarios have measurably demonstrated that Timeliner's smooth, deep zooming guided by helpful features results in effective browsing and annotation of long recordings.

The first scenario was at an uncontrolled, untutored laboratory open house. Many young children used Timeliner with a traditional spectrogram in the context of an "easter egg hunt" game, racing the clock to find and mark brief amusing sound effects (mooing, cuckoo clocks, motorcycles, *etc.*) hidden in orchestral music [10]. This exhibit aimed to teach the novel concept of time-frequency representations to children aged about 6 to 13.

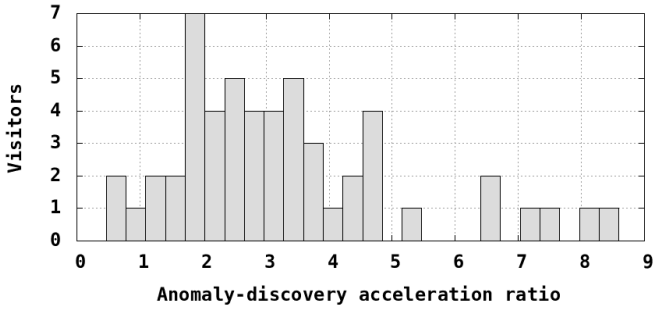

Figure 6: Rate of discovering anomalous sounds by children using Timeliner, as a multiple of real time (reproduced from Hasegawa-Johnson *et al.* [10]).

Had the children ignored or misunderstood the spectrogram, they would have found about as many sound effects as they would have from real-time listening. In fact, most found about three times as many, some upwards of seven times as many (Fig. 6). The bimodal distribution of performance was explained by informal feedback from participants: the lower mode corresponds to children who used an audio-visual search strategy, the higher mode to purely visual search. (An even higher mode, far outside the plotted figure, corresponds to friendly competition among the graduate students staffing the exhibit.)

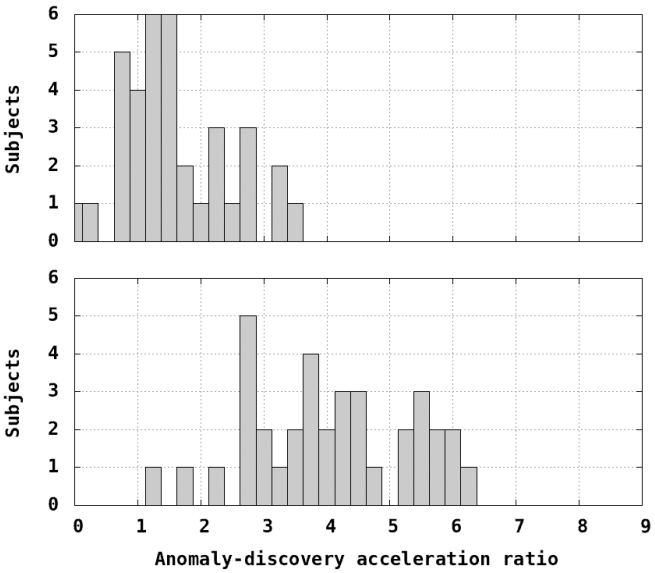

Figure 7: Rate of discovering anomalous sounds by adult subjects using Timeliner, as a multiple of real time. Timeliner displayed spectrograms that were either conventional (top) or saliency-maximized (bottom).

The second scenario was a controlled study using adult subjects given a task of finding and annotating rare, quiet, anomalous sounds that had been injected into long recordings of business meetings. Compared to conventional spectrograms, subjects' F-score (a combination of precision and accuracy) doubled when viewing saliency-maximized spectrograms [14]. In this more difficult task, subjects viewing conventional spectrograms found 1.5 to 3 times as many anomalies as would be expected from real-time search. Subjects viewing saliency-maximized spectrograms found considerably more (Fig. 7). Because this study aimed to measure only the effectiveness of saliency maximization on spectrograms, other helpful features such as event classifiers were not shown.

7. CONCLUSIONS AND FUTURE WORK

Timeliner demonstrates fast, effective browsing of recordings long enough to be otherwise intractable. Its flicker-free deep zoom with zero latency on commodity mobile hardware is made possible by both agglomerative caching and one-dimensional mipmaps. Its user interface is straightforward, efficient, quickly learned, and well suited to touchscreen and mouse-and-keyboard.

By generalizing the agglomerative cache's binary tree to a quadtree or an octree, and then constructing mipmaps in two or three dimensions, similar smooth browsing of two- or three-dimensional data is possible. (Others have suggested this for the special case of optical two-dimensional data: Silverlight's DeepZoomImageTileSource class implements this in the context of serving images to a web browser plugin [17]. This has been adapted to large databases of scientific imagery [18, 21].) Fast browsing of large areal or volumetric data will surely be as compelling as it is of time series.

8. ACKNOWLEDGMENTS

This work is funded by National Science Foundation grant 0807329.

The author thanks Mark Hasegawa-Johnson, Sarah King, Kai-Hsiang Lin, and Xiaodan Zhuang for their technical observations and insights, Audrey Fisher for meticulous proofreading, and the reviewers for their helpful suggestions.

9. REFERENCES

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

[2] P. Bénard, A. Bousseau, and J. Thollot. Dynamic solid textures for real-time coherent stylization. In *Symposium on Interactive 3D Graphics and Games (I3D)*, pages 121–127. ACM, 2009.

[3] J. Blow. Mipmapping, part 1. *Game Developer Magazine*, 8(12):13–17, Dec. 2001.

[4] J. Blow. Mipmapping, part 2. *Game Developer Magazine*, 9(1):16–19, Jan. 2002.

[5] D. Cohen, C. Goudeseune, and M. Hasegawa-Johnson. Efficient simultaneous multi-scale computation of FFTs. Technical Report FODAVA-09-01, NSF/DHS FODAVA-Lead: Foundations of Data and Visual Analytics, 2009.

[6] D. Ellis. The SPRACH project. www.icsi.berkeley.edu/~dpwe/projects/sprach, 1999.

[7] D. Ellis, C. Oei, C. Wooters, and P. Faerber. Quicknet. www.icsi.berkeley.edu/Speech/qn.html, 2012.

[8] C. Goudeseune. Timeliner. http://mickey.ifp.illinois.edu/speechWiki/index.php/Software, 2012.

[9] C. Han, E. Risser, R. Ramamoorthi, and E. Grinspun. Multiscale texture synthesis. *ACM Trans. Graphics*, 27(3), Aug. 2008.

[10] M. Hasegawa-Johnson, C. Goudeseune, J. Cole, H. Kaczmarski, H. Kim, S. King, T. Mahrt, J.-T. Huang, X. Zhuang, K.-H. Lin, H. V. Sharma, Z. Li, and T. S. Huang. Multimodal speech and audio user interfaces for K-12 outreach. In *Proc. Asia-Pacific Signal and Information Processing Assn.*, 2011.

[11] P. S. Heckbert. Fundamentals of texture mapping and image warping. Master's thesis, University of California, Berkeley, June 1989.

[12] C.-F. Hollemeersch, B. Pieters, P. Lambert, and R. Van de Walle. A new approach to combine texture compression and filtering. *The Visual Computer*, 28(4):371–385, 2012.

[13] M. Huijbregts. *Segmentation, Diarization and Speech Transcription: Surprise Data Unraveled*. PhD thesis, University of Twente, 2008.

[14] K.-H. Lin, X. Zhuang, C. Goudeseune, S. King, M. Hasegawa-Johnson, and T. S. Huang. Improving faster-than-real-time human acoustic event detection by saliency-maximized audio visualization. In *Proc. ICASSP*, pages 1–4, 2012.

[15] S. Meignier and T. Merlin. LIUM SpkDiarization: an open source toolkit for diarization. In *Carnegie-Mellon University Sphinx Workshop for Users and Developers (CMU-SPUD)*, Mar. 2010.

[16] P. Mermelstein. Distance measures for speech recognition: Psychological and instrumental. In C. H. Chen, editor, *Pattern Recognition and Artificial Intelligence*, pages 374–388. Academic Press, 1976.

[17] Microsoft Corp. Silverlight. www.silverlight.net, 2012.

[18] B. Reitinger, M. Hoefler, A. Lengauer, R. Tomasi, M. Lamperter, and M. Gruber. Dragonfly: interactive visualization of huge aerial image datasets. In *Proc. 21st ISPRS Congress*, volume 37, pages 491–494, 2008.

[19] R. N. Shepard. Circularity in judgements of relative pitch. *J. Acoust. Soc. Am.*, 36(12):2346–2353, 1964.

[20] L. Williams. Pyramidal parametrics. *SIGGRAPH Computer Graphics*, 17(3):1–11, July 1983.

[21] R. Williams, L. Yan, X. Zhou, L. Lu, A. Centeno, L. Kuan, M. Hawrylycz, and G. Rosen. Global exploratory analysis of massive neuroimaging collections using Microsoft Live Labs Pivot and Silverlight. In *Neuroinformatics: INCF Japan Node Session Abstracts*, 2010.

[22] C. Xu and S. A. Boppart. Comparative performance analysis of time-frequency distributions for spectroscopic optical coherence tomography. In *Biomedical Topical Meeting*, page FH9. Optical Society of America, 2004.

[23] S. Young, G. Evermann, T. Hain, D. Kershaw, G. Moore, J. Odell, D. Ollason, D. Povey, V. Valtchev, and P. Woodland. *The HTK Book*. Cambridge University Engineering Dept., Cambridge, UK, 2002.

[24] X. Zhuang, X. Zhou, M. A. Hasegawa-Johnson, and T. S. Huang. Real-world acoustic event detection. *Pattern Recognition Letters*, 31(2):1543–1551, Sept. 2010.

[25] X. Zhuang, X. Zhou, T. S. Huang, and M. Hasegawa-Johnson. Feature analysis and selection for acoustic event detection. In *Proc. ICASSP*, pages 17–20, 2008.