Content-Based Classification, Search, and Retrieval of Audio

Erling Wold, Thom Blum, Douglas Keislar,
and James Wheaton
Muscle Fish

Many audio and multimedia applications would benefit from the ability to classify and search for audio based on its characteristics. The audio analysis, search, and classification engine described here reduces sounds to perceptual and acoustical features. This lets users search or retrieve sounds by any one feature or a combination of them, by specifying previously learned classes based on these features, or by selecting or entering reference sounds and asking the engine to retrieve similar or dissimilar sounds.

The rapid increase in speed and capacity of computers and networks has allowed the inclusion of audio as a data type in many modern computer applications. However, the audio is usually treated as an opaque collection of bytes with only the most primitive fields attached: name, file format, sampling rate, and so on. Users accustomed to searching, scanning, and retrieving text data can be frustrated by the inability to look inside the audio objects.

Multimedia databases or file systems, for example, can easily have thousands of audio recordings. These could be anything from a library of sound effects to the soundtrack portion of a news footage archive. Such libraries are often poorly indexed or named to begin with. Even if a previous user has assigned keywords or indices to the data, these are often highly subjective and may be useless to another person. Searching for a particular sound or class of sound (such as applause, music, or the speech of a particular speaker) can be a daunting task.

How might people want to access sounds? We believe there are several useful methods, all of which we have attempted to incorporate into our system.

■ *Simile*: saying one sound is like another sound or a group of sounds in terms of some characteristics. For example, "like the sound of a herd of elephants." A simpler example would be to say that it belongs to the class of speech sounds or the class of applause sounds, where the system has previously been trained on other sounds in this class.

■ *Acoustical/perceptual features*: describing the sounds in terms of commonly understood physical characteristics such as brightness, pitch, and loudness.

■ *Subjective features*: describing the sounds using personal descriptive language. This requires training the system (in our case, by example) to understand the meaning of these descriptive terms. For example, a user might be looking for a "shimmering" sound.

■ *Onomatopoeia*: making a sound similar in some quality to the sound you are looking for. For example, the user could making a buzzing sound to find bees or electrical hum.

In a retrieval application, all of the above could be used in combination with traditional keyword and text queries.

To accomplish any of the above methods, we first reduce the sound to a small set of parameters using various analysis techniques. Second, we use statistical techniques over the parameter space to accomplish the classification and retrieval.

Previous research

Sounds are traditionally described by their pitch, loudness, duration, and timbre. The first three of these psychological percepts are well understood and can be accurately modeled by measurable acoustic features. Timbre, on the other hand, is an ill-defined attribute that encompasses all the distinctive qualities of a sound other than its pitch, loudness, and duration. The effort to discover the components of timbre underlies much of the previous psychoacoustic research that is relevant to content-based audio retrieval.[1]

Salient components of timbre include the amplitude envelope, harmonicity, and spectral envelope. The attack portions of a tone are often essential for identifying the timbre. Timbres with similar spectral energy distributions (as measured by the centroid of the spectrum) tend to be judged as perceptually similar. However, research has shown that the time-varying spectrum of a single musical instrument tone cannot generally be treated as a "fingerprint" identifying the instrument, because there is too much variation across

the instrument's range of pitches and across its range of dynamic levels.

Various researchers have discussed or proto-typed algorithms capable of extracting audio structure from a sound.[2] The goal was to allow queries such as "find the first occurrence of the note G-sharp." These algorithms were tuned to specific musical constructs and were not appropriate for all sounds.

Other researchers have focused on indexing audio databases using neural nets.[3] Although they have had some success with their method, there are several problems from our point of view. For example, while the neural nets report similarities between sounds, it is very hard to "look inside" a net after it is trained or while it is in operation to determine how well the training worked or what aspects of the sounds are similar to each other. This makes it difficult for the user to specify which features of the sound are important and which to ignore.

Analysis and retrieval engine

Here we present a general paradigm and specific techniques for analyzing audio signals in a way that facilitates content-based retrieval. Content-based retrieval of audio can mean a variety of things. At the lowest level, a user could retrieve a sound by specifying the exact numbers in an excerpt of the sound's sampled data. This is analogous to an exact text search and is just as simple to implement in the audio domain.

At the next higher level of abstraction, the retrieval would match any sound containing the given excerpt, regardless of the data's sample rate, quantization, compression, and so on. This is analogous to a fuzzy text search and can be implemented using correlation techniques. At the next level, the query might involve acoustic features that can be directly measured and perceptual (subjective) properties of the sound.[4,5] Above this, one can ask for speech content or musical content.

It is the "sound" level—acoustic and perceptual properties—with which we are most concerned here. Some of the aural (perceptual) properties of a sound, such as pitch, loudness, and brightness, correspond closely to measurable features of the audio signal, making it logical to provide fields for these properties in the audio database record. However, other aural properties ("scratchiness," for instance) are more indirectly related to easily measured acoustical features of the sound. Some of these properties may even have different meanings for different users.

We first measure a variety of acoustical features of each sound. This set of N features is represented as an N-vector. In text databases, the resolution of queries typically requires matching and comparing strings. In an audio database, we would like to match and compare the aural properties as described above. For example, we would like to ask for all the sounds similar to a given sound or that have more or less of a given property. To guarantee that this is possible, sounds that differ in the aural property should map to different regions of the N-space. If this were not satisfied, the database could not distinguish between sounds with different values for this property. Note that this approach is similar to the "feature-vector" approach currently used in content-based retrieval of images, although the actual features used are very different.[6]

Since we cannot know the complete list of aural properties that users might wish to specify, it is impossible to guarantee that our choice of acoustical features will meet these constraints. However, we can make sure that we meet these constraints for many useful aural properties.

Acoustical features

We can currently analyze the following aspects of sound: loudness, pitch, brightness, bandwidth, and harmonicity.

Loudness is approximated by the signal's root-mean-square (RMS) level in decibels, which is calculated by taking a series of windowed frames of the sound and computing the square root of the sum of the squares of the windowed sample values. (This method does not account for the frequency response of the human ear; if desired, the necessary equalization can be added by applying the Fletcher-Munson equal-loudness contours.) The human ear can hear over a 120-decibel range. Our software produces estimates over a 100-decibel range from 16-bit audio recordings.

Pitch is estimated by taking a series of short-time Fourier spectra. For each of these frames, the frequencies and amplitudes of the peaks are measured and an approximate greatest common divisor algorithm is used to calculate an estimate of the pitch. We store the pitch as a log frequency. The pitch algorithm also returns a pitch confidence value that can be used to weight the pitch in later calculations. A perfect young human ear can hear frequencies in the 20-Hz to 20-kHz range. Our software can measure pitches in the range of 50 Hz to about 10 kHz.

Brightness is computed as the centroid of the

short-time Fourier magnitude spectra, again stored as a log frequency. It is a measure of the higher frequency content of the signal. As an example, putting your hand over your mouth as you speak reduces the brightness of the speech sound as well as the loudness. This feature varies over the same range as the pitch, although it can't be less than the pitch estimate at any given instant.

Bandwidth is computed as the magnitude-weighted average of the differences between the spectral components and the centroid. As examples, a single sine wave has a bandwidth of zero and ideal white noise has an infinite bandwidth.

Harmonicity distinguishes between harmonic spectra (such as vowels and most musical sounds), inharmonic spectra (such as metallic sounds), and noise (spectra that vary randomly in frequency and time). It is computed by measuring the deviation of the sound's line spectrum from a perfectly harmonic spectrum. This is currently an optional feature and is not used in the examples that follow. It is normalized to lie in a range from zero to one.

All of these aspects of sound vary over time. The trajectory in time is computed during the analysis but not stored as such in the database. However, for each of these trajectories, several features are computed and stored. These include the average value, the variance of the value over the trajectory, and the autocorrelation of the trajectory at a small lag. Autocorrelation is a measure of the smoothness of the trajectory. It can distinguish between a pitch glissando and a wildly varying pitch (for example), which the simple variance measure cannot.

The average, variance, and autocorrelation computations are weighted by the amplitude trajectory to emphasize the perceptually important sections of the sound. In addition to the above features, the duration of the sound is stored. The feature vector thus consists of the duration plus the parameters just mentioned (average, variance, and autocorrelation) for each of the aspects of sound given above. Figure 1 shows a plot of the raw trajectories of loudness, brightness, bandwidth, and pitch for a recording of male laughter.

After the statistical analyses, the resulting analysis record (shown in Table 1) contains the computed values. These numbers are the only information used in the content-based classification and retrieval of these sounds. It is possible to

see some of the essential characteristics of the sound. Most notably, we see the rapidly time-varying nature of the laughter.

Training the system

It is possible to specify a sound directly by submitting constraints on the values of the N-vector described above directly to the system. For example, the user can ask for sounds in a certain range of pitch or brightness, However, it is also possible to train the system by example. In this case, the user selects examples of sounds that demonstrate the property the user wishes to train, such as "scratchiness."

For each sound entered into the database, the N-vector, which we represent as *a*, is computed. When the user supplies a set of example sounds for training, the mean vector μ and the covariance matrix R for the *a* vectors in each class are calculated. The mean and covariance are given by

$$\mu = (1/M) \sum_j a[j]$$

$$R = (1/M) \sum_j (a[j] - \mu)(a[j] - \mu)^T$$

where M is the number of sounds in the summation. In practice, one can ignore the off-diagonal elements of R if the feature vector elements are reasonably independent of each other. This simplification can yield significant savings in computation time. The mean and covariance together become the system's model of the perceptual property being trained by the user.

Classifying sounds

When a new sound needs to be classified, a distance measure is calculated from the new sound's *a* vector and the model above. We use a weighted
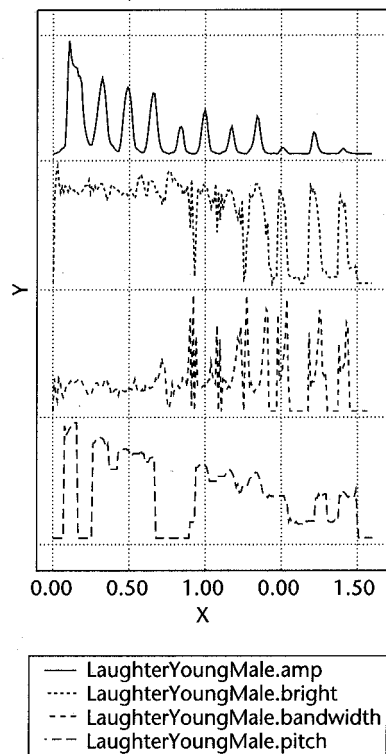
Property	Mean	Variance	Autocorrelation
Loudness	−54.4112	221.451	0.938929
Pitch	4.21221	0.151228	0.524042
Brightness	5.78007	0.0817046	0.690073
Bandwidth	0.272099	0.0169697	0.519198

Table 1. Male laughter. Duration: 2.12571.

——	LaughterYoungMale.amp
·····	LaughterYoungMale.bright
- - -	LaughterYoungMale.bandwidth
-·-	LaughterYoungMale.pitch

Figure 1. Male laughter.

L_2 or Euclidean distance:

$$D = ((a - \mu)^T R^{-1} (a - \mu))^{1/2}$$

Again, the off-diagonal elements of R can be ignored for faster computation. Also, simpler measures such as an L_1 or Manhattan distance can be used. The distance is compared to a threshold to determine whether the sound is "in" or "out" of the class. If there are several mutually exclusive classes, the sound is placed in the class to which it is closest, that is, for which it has the smallest value of D.

If it is known a priori that some acoustic features are unimportant for the class, these can be ignored or given a lower weight in the computation of D. For example, if the class models some timbral aspect of the sounds, the duration and average pitch of the sounds can usually be ignored.

We also define a likelihood value L based on the normal distribution and given by

$$L = exp(-D^2/2)$$

This value can be interpreted as "how much" of the defining property for the class the new sound has.

Retrieving sounds

It is now possible to select, sort, or classify sounds from the database using the distance measure. Some example queries are

■ Retrieve the "scratchy" sounds. That is, retrieve all the sounds that have a high likelihood of being in the "scratchy" class.

■ Retrieve the top 20 "scratchy" sounds.

■ Retrieve all the sounds that are less "scratchy" than a given sound.

■ Sort the given set of sounds by how "scratchy" they are.

■ Classify a given set of sounds into the following set of classes.

For small databases, it is easiest to compute the distance measure(s) for all the sounds in the database and then to choose the sounds that match the desired result. For large databases, this can be too expensive. To speed up the search, we index (sort) the sounds in the database by all the acoustic features. This allows us to quickly retrieve any desired hyper-rectangle of sounds in the database by requesting all sounds whose feature values fall in a set of desired ranges. Requesting such hyper-rectangles allows a much more efficient search. This technique has the advantage that it can be implemented on top of the very efficient index-based search algorithms in existing commercial databases.

As an example, consider a query to retrieve the top M sounds in a class. If the database has M_0 sounds total, we first ask for all the sounds in a hyper-rectangle centered around the mean μ with volume V such that

$$V/V_0 = M/M_0$$

where V_0 is the volume of the hyper-rectangle surrounding the entire database. The extent of the hyper-rectangle in each dimension is proportional to the standard deviation of the class in that dimension.

We then compute the distance measure for all the sounds returned and return the closest M sounds. If we didn't retrieve enough sounds that matched the query from this first attempt, we increase the hyper-rectangle volume by the ratio of the number requested to the number found and try again.

Note that the above discussion is a simplification of our current algorithm, which asks for bigger volumes to begin with to correct for two factors. First, for our distance measure, we really want a hypersphere of volume V, which means we want the hyper-rectangle that circumscribes this sphere. Second, the distribution of sounds in the feature space is not perfectly regular. If we assume some reasonable distribution of the sounds in the database, we can easily compute how much larger V has to be to achieve some desired confidence level that the search will succeed.

Quality measures

The magnitude of the covariance matrix R is a measure of the compactness of the class. This can be reported to the user as a quality measure of the classification. For example, if the dimensions of R are similar to the dimensions of the database, this class would not be useful as a discriminator, since all the sounds would fall into it. Similarly, the system can detect other irregularities in the training set, such as outliers or bimodality.

The size of the covariance matrix in each dimension is a measure of the particular dimen-

sion's importance to the class. From this, the user can see if a particular feature is too important or not important enough. For example, if all the sounds in the training set happen to have a very similar duration, the classification process will rank this feature highly, even though it may be irrelevant. If this is the case, the user can tell the system to ignore duration or weight it differently, or the user can try to improve the training set. Similarly, the system can report to the user the components of the computed distance measure. Again, this is an indication to the user of possible problems in the class description.

Note that all of these measures would be difficult to derive from a non-statistical model such as a neural network.

Segmentation

The discussion above deals with the case where each sound is a single gestalt. Some examples of this would be single short sounds, such as a door slam, or longer sounds of uniform texture, such as a recording of rain on cement. Recordings that contain many different events need to be segmented before using the features above. Segmentation is accomplished by applying the acoustic analyses discussed to the signal and looking for transitions (sudden changes in the measured features). The transitions define segments of the signal, which can then be treated like individual sounds. For example, a recording of a concert could be scanned automatically for applause sounds to determine the boundaries between musical pieces. Similarly, after training the system to recognize a certain speaker, a recording could be segmented and scanned for all the sections where that speaker was talking.

Performance

We have used the above algorithms at Muscle Fish on a test sound database that contains about 400 sound files. These sound files were culled from various sound effects and musical instrument sample libraries. A wide variety of sounds are represented from animals, machines, musical instruments, speech, and nature. The sounds vary in duration from less than a second to about 15 seconds.

A number of classes were made by running the classification algorithm on some perceptually similar sets of sounds. These classes were then used to reorder the sounds in the database by their likelihood of membership in the class. The following discussion shows the results of this process for several sound sets. These examples illustrate the char-

acter of the process and the fuzzy nature of the retrieval. (For more information, and to duplicate these examples, see the "Interactive Web Demo" sidebar.)

Example 1: Laughter. For this example, all the recordings of laughter except two were used in creating the class. Figure 2 shows a plot of the class membership likelihood values (the Y-axis) for all of the sound files in the test database. Each vertical strip along the X-axis is a user-defined category (the directory in which the sound resides). See the "Class Model" sidebar on p. 32 for details on how our system computed this model.

The highest returned likelihoods are for the laughing sounds, including the two that were not included in the original training set, as well as one of the animal recordings. This animal recording is of a chicken coop and has strong similarities in sound to the laughter recordings, consisting of a number of strong sound bursts.

Example 2: Female speech. Our test database contains a number of very short recordings of a

Figure 2. Laughter classification.

Legend:
- Laughter.order not in training set
- Animals
- Bells
- Crowds
- k2000
- Laughter
- Telephone
- Water
- Mcgill/altotrombone
- Mcgill/cellobowed
- Mcgill/oboe
- Mcgill/percussion
- Mcgill/tubularbells
- Mcgill/violinbowed
- Mcgill/violinpizz
- Speech/female
- Speech/male

Table A. Class model for laughter example.

Feature		Mean	Variance	Importance
Duration		2.71982	0.191312	6.21826
Loudness:	Mean	−45.0014	18.9212	10.3455
	Variance	200.109	1334.99	5.47681
	Autocorrelation	0.955071	7.71106e-05	108.762
Brightness:	Mean	6.16071	0.0204748	43.0547
	Variance	0.0288125	0.000113187	2.70821
	Autocorrelation	0.715438	0.0108014	6.88386
Bandwidth:	Mean	0.363269	0.000434929	17.4188
	Variance	0.00759914	3.57604e-05	1.27076
	Autocorrelation	0.664325	0.0122108	6.01186
Pitch:	Mean	4.48992	0.39131	7.17758
	Variance	0.207667	0.0443153	0.986485
	Autocorrelation	0.562178	0.00857394	6.07133

Figure 3. "Tear gas" similarities.

matrix R. The highest likelihoods are for the other female speech recordings, with the male speech recordings following close behind.

Example 3: Touchtones. A set of telephone touchtones was used to generate the class in Figure 4. Again, the touchtone likelihoods are clearly separated from those of other categories. One of the touchtone recordings that was left out of the training set also has a high likelihood, but notice that the other one, as well as one of those included in the training set, returned very low likelihoods. Upon investigation, we found that the two low-likelihood touchtone recordings were of entire seven-digit phone numbers, whereas all the high-likelihood touchtone recordings were of single-digit tones. In this case, the automatic classification detected an aural difference that was not represented in the user-supplied categorization.

Applications

The above technology is relevant to a number of application areas. The examples in this section will show the power this capability can bring to a user working in these areas.

Audio databases and file systems

Any audio database or, equivalently, a file system designed to work with large numbers of audio files, would benefit from content-based capabilities. Both of these require that the audio data be represented or supplemented by a data record or object that points to the sound and adds the necessary analysis data.

When a new sound is added to the database, the analyses presented in the previous section are run on the sound and a new database record or object is formed with this supplemental information. Typically, the database would allow the user to add his or her own information to this record. In a multiuser system, users could have their own copies of the database records that they
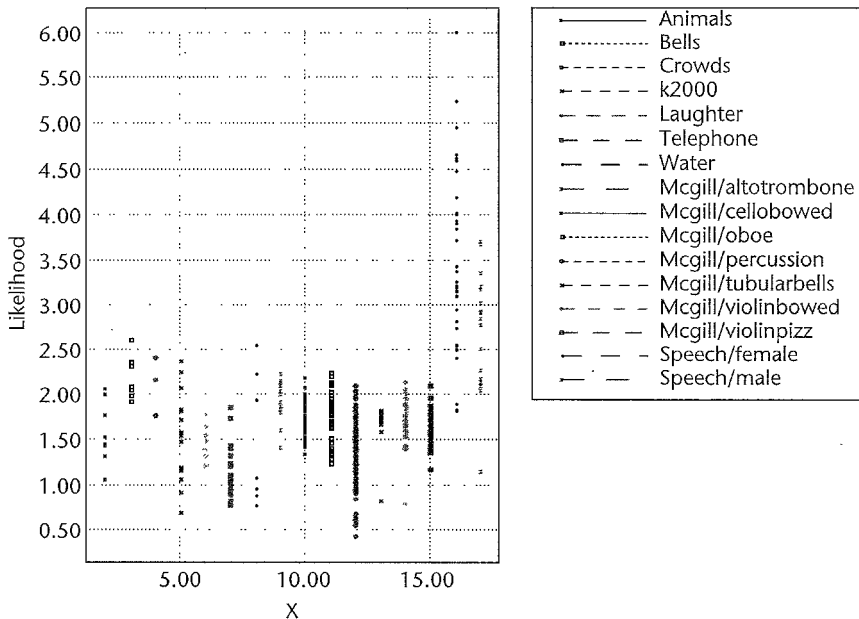
group of female and male speakers. For this example, the female-spoken phrase "tear gas" was used. Figure 3 shows a plot of the similarity (likelihood) of each of the sound files in the test database to this sound using a default value for the covariance

could modify for their particular requirements.

Figure 5 shows the record used in our sound browser, described in the next section. Fields in this record include features such as the sound file's name and properties, the acoustic features as computed by system analysis routines, and user-defined keywords and comments.

Any user of the database can form an audio class by presenting a set of sounds to the classification algorithm of the last section. The object returned by the algorithm contains a list of the sounds and the resulting statistical information. This class can be private to the user or made available to all database users. The kinds of classes that would be useful depend on the application area. For example, a user doing automatic segmentation of sports and news footage might develop classes that allow the recognition of various audience sounds such as applause and cheers, referees' whistles, close-miked speech, and so forth.

The database should support the queries described in the last section as well as more standard queries on the keywords, sampling rate, and so on.

An audio database browser

In this section, we present a front-end database application named SoundFisher that lets the user search for sounds using queries that can be content based. In addition, it permits general maintenance of the database's entries by adding, deleting, and describing sounds.

Figure 6 shows the graphical user interface (GUI) for the application during the formation of a query. The upper window is the Query window. The Search button initiates a search using the query and the results are then displayed in the Current Sounds window. Initially, the Results window shows a listing of all the sounds in the database.

A query is formed using a combination of constraints on the various fields in the database schema and a set of sounds that form a training set for a class. The example in Figure 6 is a query to find recent high-fidelity sounds in the database containing the "animal" or "barn" keywords that are similar to goose sounds, ignoring sound duration and average loudness.

The top portion of the Query window consists of a set of rows, each of which is a component of the total query. Each component includes the name of the field, a constraint operator appropriate for the data type of that field, and the value to which the operator is applied. Pressing one of the
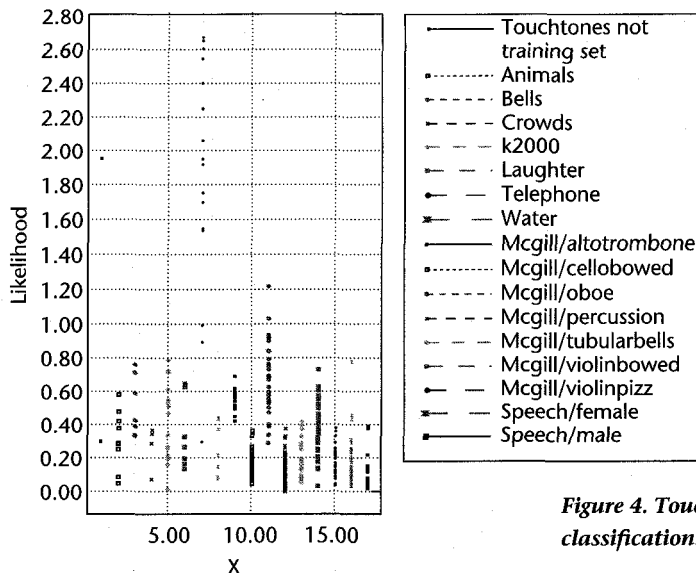
buttons in the row pops up a menu of possibilities or a slider and entry window combination for floating-point values. In Figure 6, there is one component that constrains the date to be recent, one that constrains the keywords, and one that specifies a high sampling rate. The OR subcomponent on the keyword field is added through a menu item. There are also menu items for adding and deleting components. All the components are ANDed together to form the final query.

The bottom portion of the Query window consists of a list of sounds in the training set. In this case, the sounds consist of all the goose recordings. We have brought up sliders for duration and loudness and set them to zero so that these features will be ignored in the likelihood computation.

Although not shown in this figure, some of the query component operators are fuzzy. For example, the user can constrain the pitch to be approximately 100 Hz. This constraint will cause the system to compute a likelihood for each sound equal to the inverse of the distance between that sound's pitch feature and 100 Hz. This likelihood is used as a multiplier against the likelihood computed from the similarity calculation or other parts of the query that yield fuzzy results. Note

Figure 4. Touchtone classification.

Sound file attributes
    File name
    Sample rate
    Sample size
    Sound file format
    Number of channels
    Creation date
    Analysis date
User attributes
    Keywords
    Comments
Analysis feature vector
    Duration
    Pitch   [mean, variance, autocorrelation]
    Amplitude [mean, variance, autocorrelation]
    Brightness [mean, variance, autocorrelation]
    Bandwidth [mean, variance, autocorrelation]
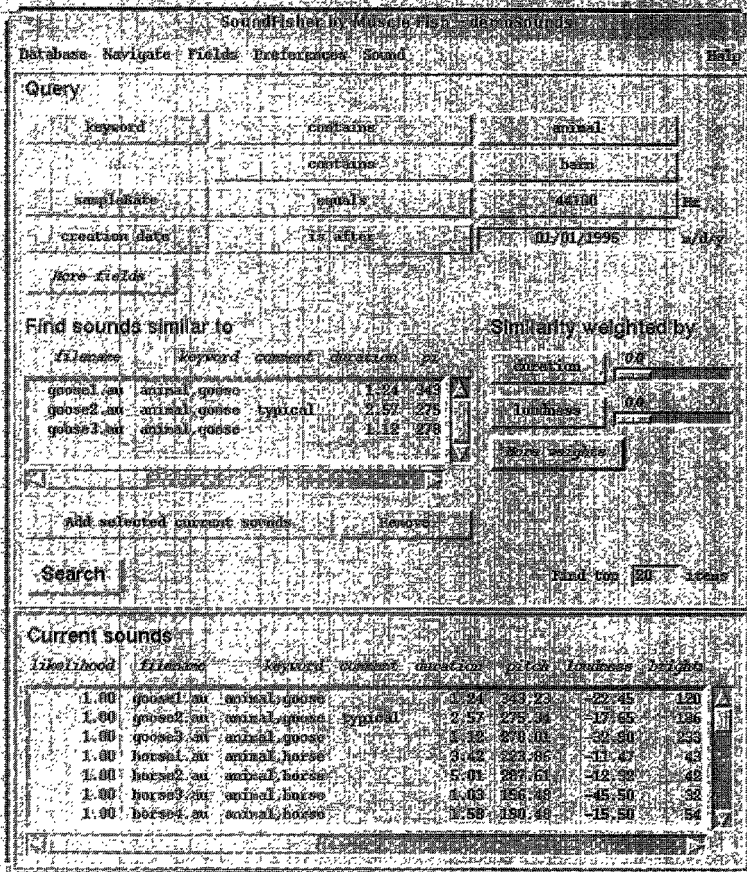
Figure 5. Database record.

Figure 6. SoundFisher browser.

as "scratchiness," save that model under a name, then reuse that concept in future queries.

Audio editors

Current audio editors operate directly on the samples of the audio waveform. The user can specify locations and values numerically or graphically, but the editor has no knowledge of the audio content. The audio content is only accessible by auditioning the sound, which is tedious when editing long recordings.

A more useful editor would include knowledge of the audio content. Using the techniques presented in this article, a variety of sound classes appropriate for the particular application domain could be developed. For example, editing a concert recording would be aided by classes for audience applause, solo instruments, loud and soft ensemble playing, and other typical sound features of concerts. Using the classes, the editor could have the entire concert recording initially segmented into regions and indexed, allowing quick access to each musical piece and subsections thereof. During the editing process, all the types of queries presented in the preceding sections could be used to navigate through the recording. For example, the editor could ask the system to highlight the first C-sharp in the oboe solo section for pitch correction.

A graphical editor with these capabilities would have Search or Find commands that functioned like the query command of the SoundFisher audio browser. Since it would often be necessary to build new classes on the fly, there should be commands for classification and analysis or tight integration with a database application such as the Sound-Fisher audio browser.

Surveillance

The application of content-based retrieval in surveillance is identical to that of the audio editor except that the identification and classification would be done in real time. Many offices are already equipped with computers that have built-in audio input devices. These could be used to listen for the sounds of people, glass breaking, and so on. There are also a number of police jurisdictions using microphones and video cameras to continuously survey areas having a high incidence of criminal activity or a low tolerance of such activity. Again, such surveillance could be made more efficient and easier to monitor with the ability to detect sounds associated with criminal activity.

that ANDing two fuzzy searches is accomplished by multiplying the likelihoods and ORing two fuzzy searches by adding the likelihoods.

There are a number of ways to refine searches through this interface, and all queries can be saved under a name given by the user. These queries can be recalled and modified. The Navigate menu contains these commands as well as a history mechanism that remembers all the queries on the current query path. The Back and Forward commands allow navigation along this path. An entry is made in the path each time the Search button is pressed. It is, of course, possible to start over from scratch. There is also an option to apply the query to the current sounds or to the entire database of sounds.

Any saved query can be used as part of a new query. One of the fields available for constructing query components is "query," meaning "saved query." This lets the user perform complex searches that combine previous queries in Boolean expressions. It also lets the user train the system with a class of sounds embodying a concept such

Automatic segmentation of audio and video

In large archives of raw audio and video, it is useful to have some automatic indexing and segmentation of the raw recordings. There has been quite a bit of work on the video side of the segmentation problem using scene changes and camera movement.[7] The audio soundtrack of video as well as audio-only recordings can be automatically indexed and segmented using the analysis methods discussed previously.

This is accomplished by analyzing the recording and extracting the trajectories for loudness, pitch, brightness, and other features. Some segmentation can be done at this level by looking at transitions and sudden changes in the analysis data. We used this technique to develop the Audio-to-MIDI conversion system that is part of the Studio Vision Pro 3.0 product from Opcode Systems. In this product, the raw trajectories are segmented by amplitude and pitch and converted into musical score information in the form of MIDI data. This is a convenient representation for understanding and manipulating the musical content of the audio recording. This product assumes musical instrument recordings, so pitch is very important. In a more general context, it might be more appropriate to segment the sound by amplitude or spectral changes.

You could treat these segments as individual sounds that can then be analyzed for their statistical features, as we have described above. Alternately, you could arbitrarily look at overlapping windows of the raw analysis data as the individual sounds. Once this is done, each of these sounds can be classified and thus indexed.

Future directions

In our current work, we are focusing on several areas to improve and refine the performance of our search, analysis, and retrieval engine.

Additional analytic features

An analysis engine for content-based audio classification and retrieval works by analyzing the acoustic features of the audio and reducing these to a few statistical values. The analyzed features are fairly straightforward but suffice to describe a relatively large universe of sounds. More analyses could be added to handle specific problem domains.

General phrase-level content-based retrieval

Our current set of acoustic features is targeted toward short or single-gestalt sounds. Matching sets of our features as trajectories in time or matching segmented sequences of single-gestalt sounds would allow phrase-level audio content to be stored and retrieved. For example, the Audio-to-MIDI system referenced above could be used to do matching of musical melodies. As with all media search, a fuzzy match is what is desired.

Source separation

In our current system, simultaneously sounding sources are treated as a single ensemble. We make no attempt to separate them, as source separation is a difficult task. Approaches to separating simultaneous sounds typically involve either Gestalt psychology[8] or non-perceptual signal-processing techniques.[9,10] For musical applications, polyphonic pitch-tracking has been studied for many years, but might well be an intractable problem in the general case.

Sound synthesis

Sound synthesis could assist a user in making content-based queries to an audio database. When the user was unsure what values to use, the synthesis feature would create sound prototypes that matched the current set of values as they were manipulated. The user could then refine the synthesized example until it bore enough similarity to the desired sort of sound.

Our examples show the efficacy and useful fuzzy nature of the search. The results of searches are sometimes surprising in that they cross semantic boundaries, but aurally the results are reasonable. This is work in progress. Further implementation and testing of the system will reveal whether the chosen acoustical features are sufficient or excessive for usefully analyzing and classifying most sounds. We believe, however, that the basic approach presented here works well for a wide variety of audio database applications. **MM**

References

1. R. Plomp, *Aspects of Tone Sensation: A Psychophysical Study*, Academic Press, London, 1976.

2. S. Foster, W. Schloss, and A.J. Rockmore, "Towards an Intelligent Editor of Digital Audio: Signal Processing Methods," *Computer Music J.*, Vol. 6, No. 1, 1982, pp. 42-51.

3. B. Feiten and S. Gunzel, "Automatic Indexing of a Sound Database Using Self-Organizing Neural Nets," *Computer Music J.*, Vol. 18, No. 3, Summer 1994, pp. 53-65.

4. T. Blum et al., "Audio Databases with Content-Based Retrieval," workshop on Intelligent Multimedia Information Retrieval, 1995 Int'l Joint Conf. on Artificial Intelligence, available at http://www.musclefish.com.

5. D. Keislar et al., "Audio Databases with Content-Based Retrieval," *Proc. Int'l Computer Music Conference 1995*, International Computer Music Association, San Francisco, 1995, pp. 199-202.

6. H. Zhang, B. Furht, and S. Smoliar, *Video and Image Processing in Multimedia Systems*, Kluwer Academic Publishers, Boston, 1995.

7. H. Zhang, A. Kankanhalli, and S. Smoliar, "Automatic Partitioning of Full-Motion Video," *Multimedia Systems*, Vol. 1, No. 1, 1993, pp. 10-28.

8. S. McAdams, "Recognition of Sound Sources and Events," *Thinking in Sound: The Cognitive Psychology of Human Audition*, Clarendon Press, Oxford, 1993.

9. J. Moorer, "On the Transcription of Musical Sound by Computer," *Computer Music J.*, Vol. 1, No. 4, 1977, pp. 32-38.

10. E. Wold, *Nonlinear Parameter Estimation of Acoustic Models*, PhD Thesis, University of California at Berkeley, Berkeley, Calif., 1987.
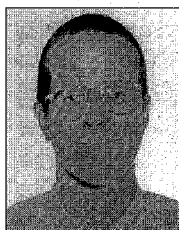
Erling Wold earned a PhD in EECS at the University of California, Berkeley in 1987 where he did research in source separation, FFT computer architectures, and stochastic sampling. Since that time, he has concentrated on signal processing and software architectures for music applications. He is a prolific composer and has written music for a variety of ensembles as well as for several feature films and theatrical works, including two operas. With the co-authors, he founded Muscle Fish, a software engineering firm specializing in audio and music.
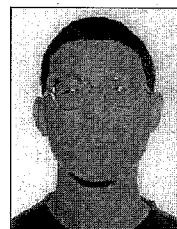
Thomas Blum received a BA in Computer Applications to Music Synthesis from Ohio State in 1978. He creates software to help manage complex data, like sound, and creative tasks, like music composition. In 1978, he cofounded the Computer Music Association and served for roughly 10 years as an associate editor of *Computer Music Journal* (MIT Press).

Douglas Keislar received a PhD in Music from Stanford University, where he conducted psychoacoustical research at the Center for Computer Research in Music and Acoustics (CCRMA). He is an associate editor of *Computer Music Journal* (MIT Press).

James A. Wheaton is president of Harmonic Systems, Inc. He received his BS in Philosophy from MIT in 1980. His research interests include new musical instruments, interactive music, and Internet audio applications.

Readers may contact Erling Wold at Muscle Fish LLC, 2550 Ninth Street, Suite 207B, Berkeley, CA 94710, e-mail erling@musclefish.com.