Mapping data and audio using an event-driven audio server for personal computers

Michael Hamman
University of Illinois Urbana-Champaign
705 W. Nevada # 4
Urbana, IL 61801
m-hamman@uiuc.edu

Camille Goudeseune
University of Illinois Urbana-Champaign
603 W. Nevada #1
Urbana, IL 61801
cog@uiuc.edu

ABSTRACT

Recent research suggests that auditory display offers new means forobserving and differentiating complex data. One standard method forrendering an auditory display is playback of previously generated audiofiles. This method is enhanced through playback modification usingtools such as Intel's RSX. MIDI-based synthesizers provide yet anothermethod for auditory display. These methods have various drawbacksthat are pressed to the limit when confronted with the requirements ofanalogical display systems. What is needed, therefore, is a way ofrendering audio that is to auditory display what a system like OpenGL isto graphical display. Audio Rendering Engine And Library (AREAL)is a real-time audio renderer and sound synthesis software library. It offers the software developer and auditory display designer a set oftools for developing high-quality audio applications for low-cost multimediacomputers using consumer or professional audio hardware. The primarypurpose of AREAL is to enable a "model-based" approach to audio which isbecoming common on high-end (and high-cost) workstations. This papergives a brief description of this software system and its potential foruse within the auditory display community.

INTRODUCTION

Recent research suggests that auditory display offers new means forobserving and differentiating complex data [3][7]. An auditory displaycan be used to signal the occurrence of discrete events or to signal thestate of an evolving multivariate data set [1][3][8]. This leadsto a variety of approaches. At one extreme, auditory events are associatedto data symbolically; that is, their meaning is characterized in termsof the source to which they are attributed [6][7][8]. At theother extreme, auditory events are associated with data analogically; thatis, their meaning is characterized in terms of the interrelations withinand among data [8][10].

One standard method for rendering an auditory display is playback ofpreviously generated audio files. An advantage of this method isthat it allows for precisely controlled studio engineering involving recording,synthesis, and processing techniques that would be computationally prohibitivefor real-time rendering. In addition, tools for editing and playbackof audio files are readily available under virtually any operating system. A disadvantage of this method is that multiple playback of the same audiofile, over and over, can become tiresome to the ear. Moreover, whiledisk storage generally costs little, the management of large numbers ofaudio files is unruly and inevitably error-prone. Finally, no matterhow many different audio files are used, there will always be holes inthe mapping of data features to acoustic features if the data or processto be mapped is even slightly complex. This becomes more and morethe case as we move toward analogical displays.

One way to deal with this situation is to employ already existing (andfor the most part, free) audio tools which allow for real-time manipulationof stored sound files. This can be a viable solution for "symbolic"auditory displays representing discrete events and process states. Through the manipulation of a single sound file, one could represent manydifferent states of a display object such as an icon or mouse cursor. However, this approach still suffers under the requirements called forwith analogic displays. In order to deliver effective analogic representations,an audio rendering system is needed which can be deeply mapped to the modelbeing represented.

At first glance, MIDI-based synthesizers offer an appealing alternativeto the use of sound files. Using MIDI, a synthesizer can be triggeredby mapping data to particular MIDI messages [7]. The problem withMIDI, however, is two-fold. First, one must accept the sounds thatare given by the particular synthesizer technology at ones disposal. These sounds may differ, depending on the particular device being used.Second, MIDI is limited by its narrow control bandwidth and thuscannot be used for the display of highly multivariate and rapidly evolvingdata [9]. This has a secondary consequence which is that while onemight solve the problem of mapping multivariate data to sound through theuse of system exclusive messages (messages that are manufacturer-specific),these messages exacerbate the problem of narrow bandwidth since they usuallyrequire higher bandwidth per event.

What is needed is a way of rendering audio that is to auditory displaywhat a system like OpenGL is to graphical display. Moreover, toolsare needed with which auditory display designers can test hypotheses andexperiment with different methods by which a data model might be renderedacoustically.

AREAL

Audio Rendering Engine And Library (AREAL) is a real-time audio rendererand sound synthesis software library. It offers the software developerand auditory display designer a set of tools for developing high-qualityaudio applications for low-cost multimedia computers using consumer orprofessional audio hardware. The primary purpose of AREAL is to enablea "model-based" approach to audio which is becoming common on high-end(and high-cost) workstations. We understand "model-based" audio tobe that in which auditory events are generated in real-time as a directmapping of a dynamically evolving data set or process.

AREAL incorporates (1) a real-time scheduler for managing uninterruptedplayback of computed audio samples using a standard sound card; (2) anextendible synthesis library, whose API is called directly by an application;and (3) a method by which audio rendering "models" are specified.

SOFTWARE ARCHITECTURE

AREAL consists of three software layers: the Scheduler, the SynthesisEngine, and a C-language "wrapper" (figure 1).

The Scheduler handles the passage of samples to the sound card. It does this by iteratively filling buffers with samples as these are computedwithin the Synthesis Engine and inserting these into the sound card's queue. It maintains as low a latency as possible while providing sufficient safetyfrom audio interruptions due to other processes needing the CPU. When the Scheduler needs more samples, it executes a callback against theSynthesis Engine.

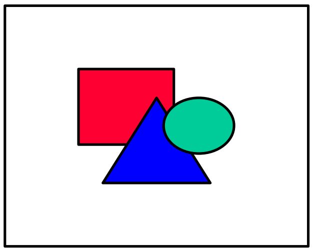

Figure 1

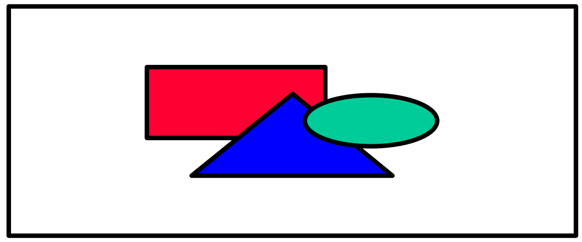

Figure 2

The Synthesis Engine contains a hierarchy of sound synthesis classesand a facility for handling messages from the Wrapper and Scheduling layers. Upon initialization, a list of sound objects is instantiated. Whenthe Scheduler requests samples (through execution of its callback procedure),the Synthesis Engine obtains buffers of samples from each synthesis objectand sums them into the buffer that has been passed from the Scheduler (figure2).

The sound synthesis class library is a collection of C++ classes, eachof which specifies a particular synthesis algorithm and defines a standardclass interface. We envision the sound synthesis class hierarchyas an "open" architecture, enabling addition of customized synthesis algorithmsand extensions of current algorithms based on particular rendering andoptimization requirements. Currently, the library is extendible onlyat the source-code level. In a future version, however, it is anticipatedthat this extendibility will be realized through implementation as an ActiveXcontainer. ActiveX is a Microsoft construct that allows differentapplications to exchange information and for one application to embed itselfinto the other, thus becoming, in effect, a component of it. As ActiveXcontrollers, sound synthesis libraries could literally be "dropped into"AREAL and thus become one of its synthesis libraries. This kind ofextendibility holds great promise for the future of audio applicationsin general and for auditory display in particular.

Sample underflow (which can cause annoying clicks in the audio signal)is handled through a facility called the OctaneMeister. The OctaneMeisteranticipates possible sample underflow by monitoring CPU load. Whenthis load nears 100 per cent, the OctaneMeister tells the list of instantiatedsynthesis objects to lower its computation "octane." Each synthesis objecthas its own method for reducing its octane which is defined within theclass from which it is instantiated. Moreover, synthesis objectsare ranked in order of complexity: those whose performance would be mostseriously compromised through the lowering of its "octane" are placed atthe bottom of the list.

The C-language "wrapper" forms a shell around the Scheduler and theSynthesis Engine. It exports a set of C functions to the client application. This set of functions is referred to as the "API." The API constitutesthe sole interface between the application and AREAL (as depicted by thesingle bar connecting the APPLICATION and the Wrapper in figure 1). The motivation for defining the interface as a C-language API, rather thanallowing the application to interface directly to the C++ library classeswithin AREAL, is to make the interface as simple as possible to use andto allow for applications written in languages other than C++ to be linkedto the library. Functions defined within the API instantiate synthesisobjects, set up of data-to-renderer mappings, and handle control messagesto individual synthesis objects.

MAPPING DATA TO AUDIO FROM WITHIN AN APPLICATION

Finding flexible though coherent methods for mapping data to an audio-renderingagent is a challenging task. Our approach is predicated on the desirethat once such a mapping has been established, an application should nolonger have to think about how its data and processes are being acousticallyrendered. Therefore, each time there is a change made to a variablewhich is mapped to an audio parameter, a call is made to a function withinthe AREAL API, telling AREAL the new value of that variable. Thisvalue is then dispatched to the appropriate synthesis algorithm.

Within the application, each data point that is to be rendered is mappedto a parameter within a sound synthesis algorithm. Such an algorithmis encapsulated in a synthesis class residing within AREAL. A soundsynthesis parameter is used to control some aspect of the synthesis algorithm. In a relatively simple case, such a parameter could control the frequencyof a single sine tone, while another could control its amplitude. In more complicated cases, such a parameter might control some aspect ofa more elaborate synthesis algorithm.

In an effort at a clear explication of how this works, a simple exampleis provided. In this example, we start with a data set that containstwo variables, x and y, which describe a Cartesian graph. We selecta simple additive synthesis algorithm which is defined by 3 parameters:frequency (F), amplitude (A), and number partials (T). Within theapplication (with which the AREAL library has been linked), first a rangeis defined for each synthesis parameter. In our example, F will havethe range [100Hz, 1100Hz], A will have the range [10000, 20000], and Twill have the range [0 partials, 20 partials].

After ranges have been defined for each synthesis parameter, a singleC-language function call is made to set the mapping between these datapoints and synthesis parameters:

```
SetMapping(hSynthObj,"F=(x,10,40);A=.3*(x,10,40)+.7*(y,.03,-.03); T=(y,.1, .9)");
```

SetMapping() has two arguments: a pointer (handle) to the synthesisobject (which has been previously defined within the application), anda string which specifies the mapping between application data points andsynthesis parameters. This string is parsed within the AREAL libraryin order to realize the specified mapping. In the above example,the two data points are mapped to the three synthesis parameters as follows. All synthesis parameters in the mapping are normalized to the range [0,1]relative to the range set earlier in the program. F is a direct linearmapping from data point x: as x increases from 10 to 40, F increasesfrom 0 to 1. A is a weighted sum of data points x and y, 30% and70% respectively, with x traversing the domain (10, 40) and y traversingthe domain (0.03, -0.03). So, for instance, if x=25 and y=0.0, thenA would have a normalized value of (.399=(.3 * .33)+(.7*.5)). MeanwhileT maps directly to data point y with the domain (.1, .9).

Such a mapping is specified once within the application. Afterthis, whenever the state of one of the data points is changed a call ismade to the AREAL API to inform it of this change in state:

```
...
x = getXValue();
SetParmValue(hSynthObj,"x", x);
...
```

Given a sequence of states for x and y, a correlated sequence of parameterstates is generated within the synthesis algorithm. Such a situationis shown in figure 3, in which a sequence of 5 states for x and y is correlatedto a similar sequence for synthesis parameters F, A, and T. Thesesynthesis parameters are in turn mapped against the range with respectto which they have been defined for this application. The valuesfor the synthesis parameters thus realized are shown in figure 4. As can be observed, this sequence defines an acoustical behavior in whichfrequency rises from 100 to 370 Hz, while amplitude drops from 17000 to15400 and the number of partials goes from 5 to 0 partials. The resultantbehavior defines an acoustical "gesture" which, while rising in pitch,descends in loudness and in timbral richness. This "gesture" constitutesa potentially informative and evocative representation of the data it rendersin which x rises continuously while y descends somewhat sharply beforecoming back up again slightly.

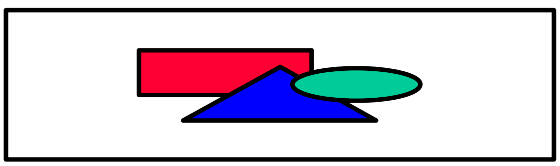

Figure 3

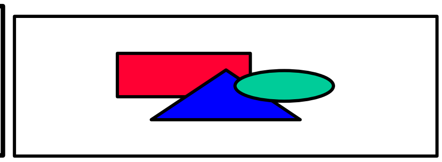

Figure 4

AUDIO DESCRIPTION FILES

Mapping data to audio parameters from within the application itselfhas one drawback: if one wishes to change the mapping, then the applicationhas to be recompiled. Not only is this burdensome for auditory displaydesigners who have access to the source code, it can prohibit those whoare not themselves programmers from being able to make changes in thatmapping at all.

Audio description files define data-to-renderer maps within a text file. The application reads this file during its initialization, and the mappingdesignated within that file is applied inside the application. Designersand composers can then effect changes in data-to-rendering mapping by makingchanges with respect to this file.

DEFINING AN AUDITORY DISPLAY OF LARGE DATASETS

Such an organization can be used to represent an arbitrarily largedata set whose values unfold at any number of rates. Imagine, forinstance, a data set involving three Cartesian maps, such as the one describedabove, plus additional data points. Figure 5 depicts the patch withwhich the corresponding acoustical representation might be defined. Each of the three Cartesian sets are shown as Sets A through C. Synthesisobjects are labeled Synth1, Synth2, and Synth3. As shown, Cartesianset A is mapped to synthesis object 1, and so on. Meanwhile, otherdata points (grouped together, for the sake of illustration, as "Otherdata points") are mapped to parameters by which a reverberation processorunit is controlled and by which channel placement is controlled.

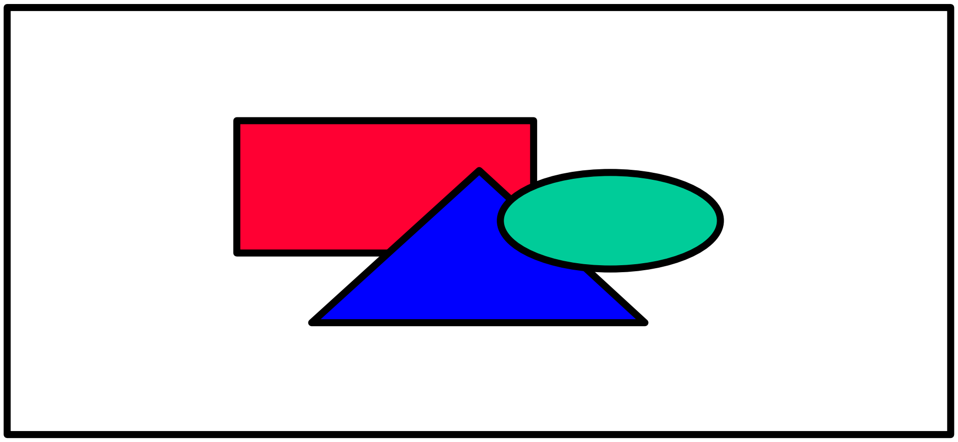

Figure 5

The resulting signal reads from left to right, beginning with the additivesynthesis objects. Samples produced by each of the synthesis objectsare summed, and placed through the reverberation processor. Thisreverberation processor, has for the sake of simplified illustration, twoparameters: reverberation time and delay time. The resulting samplesare then passed to the channel placement module, where each is multipliedby the value defined by that module's single control parameter. Foreach sample, this value is placed in the right-channel buffer cell, whileits inverse value is placed in the left-channel buffer cell.

With this particular arrangement, we are able to map a 12-D data setto elements of an auditory display. Using an Audio Description file,a designer can fine-tune this mapping until s/he finds the configurationwhich best reflects an informative view of the data being represented.

FUTURE DEVELOPMENTS

AREAL proposes a possible solution to the "build it yourself" problemwhich currently plagues research and development in auditory display. As such a proposal, it is in the infant stage of a potentially viable technology. As such, its future development is dependent on feedback from other researcheswithin the auditory display community. Currently, we are planningto develop AREAL as an ActiveX server. We are also planning on addingnetwork capabilities so that AREAL can act as a server in a network environment. Thus, a low-cost NT workstation could serve as the audio server, whilethe applications whose data it renders can run on other computers. We are also hoping to develop a Netscape plugin so that auditory displaysmay be rendered over the Web. Finally, we wish to develop graphicaltools for specifying and investigating data-to-renderer maps.

REFERENCES

1. Blattner, M.M., Papp III, A. L., Glinert, E. P. "Sonic Enhancementof Two-Dimensional Graphics Displays." in *Auditory Display: Sonification,Audification, and Auditory Interfaces*, ed.

G. Kramer. Reading, Mass:Addison-Wesley Publishing Co. 1994.

2. Brewster, S. A., Wright, P. C., Edwards, D. N. "A Detailed Investigationinto the Effectiveness of Earcons." in *Auditory Display: Sonification,Audification, and Auditory Interfaces*, ed. G. Kramer. Reading, Mass:Addison-Wesley Publishing Co. 1994.

3. Fitch, W. T., Kramer, G. "Sonifying the Body Electric: Superiorityof an Auditory over a Visual Display in a Complex, Multivariate System."in *Auditory Display: Sonification, Audification, and Auditory Interfaces*,ed. G. Kramer. Reading, Mass: Addison-Wesley Publishing Co. 1994.

4. Gaver, W. W. "The SonicFinder: An Interface that Uses AuditoryIcons." *Human-Computer Interaction* 4(1): 67-95. 1989.

5. Gaver, W. W., and Smith, R. B. "Auditory Icons in Large-scaleCollaborative Environments." in *Human-Computer Interaction - INTERACT'90*, ed. D. Diaper. North-Holland: Elsevier Science Publishers. 1990.

6. Gaver, W. W. "Using and Creating Auditory Icons." in *AuditoryDisplay: Sonification, Audification, and Auditory Interfaces*, ed. G.Kramer. Reading, Mass: Addison-Wesley Publishing Co. 1994.

7. Kramer, G. and, Ellison, S. "AUDIFICATION: The Use of Soundto Display Multivariate Data." *Proceedings of the 1991 InternationalComputer Music Conference,* Montreal. San Francisco: Computer MusicAssociation, 1991. pp. 214-221.

8. Kramer, G. "An Introduction to Auditory Display." in *AuditoryDisplay: Sonification, Audification, and Auditory Interfaces*, ed. G.Kramer. Reading, Mass: Addison-Wesley Publishing Co. 1994.

9. Moore, F. R. "The dysfunctions of MIDI," *Computer Music Journal*12(1): 19-28. 1988.

10. Sloman, A. "Afterthoughts on Analogical Representations." in *Readingsin Knowledge Representation*, ed. R. Brachman and H. Levesque. Los Altos, CA: Morgan Kaufman.