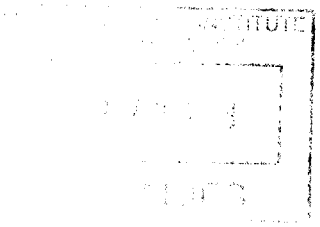
DoppelLab: Spatialized Data Sonification in a 3D Virtual Environment

by

Nicholas D. Joliat

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Engineering in Computer Science and Engineering



at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

February 2013

Author .....
Department of Electrical Engineering and Computer Science
October 15, 2012

Certified by.. .....
Joseph A. Paradiso
Associate Professor of Media Arts and Sciences
Thesis Supervisor

Accepted by ....
Prof. Dennis M. Freeman
Master of Engineering Thesis Committee

DoppelLab: Spatialized Data Sonification in a 3D Virtual Environment

by

Nicholas D. Joliat

Submitted to the Department of Electrical Engineering and Computer Science
on October 15, 2012, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Computer Science and Engineering

Abstract

This thesis explores new ways to communicate sensor data by combining spatialized sonification with data visualiation in a 3D virtual environment. A system for sonifying a space using spatialized recorded audio streams is designed, implemented, and integrated into an existing 3D graphical interface. Exploration of both real-time and archived data is enabled. In particular, algorithms for obfuscating audio to protect privacy, and for time-compressing audio to allow for exploration on diverse time scales are implemented. Synthesized data sonification in this context is also explored.

Thesis Supervisor: Joseph A. Paradiso
Title: Associate Professor of Media Arts and Sciences

Acknowledgments

I am very much indebted to my advisor, Joe Paradiso, for inspiration and support throughout this endeavor, as well as creating an excellent group culture and allowing me to be a part of it. I also thank fellow Responsive Environments RAs Gershon Dublon, Brian Mayton, and Laurel Pardue for collaboration, advice and friendship during this research; also thanks to Brian Mayton for much essential guidance in software design, system administration, and proofreading. Thanks also to Matt Aldrich for helpful discussion regarding time compression. Thanks to the Responsive Environments group as a whole for the road trips and jam sessions. Thanks to Amna Carreiro for helping me order essential software for this research. Thanks to Mary Murphy-Hoye of Intel for providing funding for much of this work, and for helpful insights.

Particular thanks also to my friends Daf Harries and Rob Ochshorn for great discussions and advice; in particular, thanks to Daf for help with audio hacking, and thanks to Rob for discussions about compression.

Finally, many thanks to Paula, to all my friends, and to my family.

Contents

List of Figures

Chapter 1

Introduction

Homes and workspaces are increasingly being instrumented with dense sensor networks, encompassing many modalities of data, from environmental (e.g. temperature) to usage data (e.g. movement). While closed-loop systems exist for addressing specific problems (such as temperature control), we are interested in creating a comprehensive interface which brings together all this complex data, and allows users to explore it and find patterns and connections between disparate kinds of data.

Graphical data visualization is a typical way to approach a problem of data display. Data visualization is currently a very popular area, in academia, industry, and art. In Edward Tufte's seminal books, most notably The Visual Display of Quantatative Information,[35] he discusses the challenges present in visualization, and strategies for contending with them; chief among them is the problem of 'flatland'. This refers to the challenge of trying to visualize many-dimensional data on a two-dimensional page or display. This problem is certainly present in the display of spatial sensor data; these data necessarily include the three-dimensional spatial coordinates of the sensors, in addition to time and whatever dimensions are present within the data itself for each sensor.

In light of these difficulties, I propose the use of data sonification in order to complement visualization of sensor data. Data sonification has been explored much less than visualization. Academic work on it does exist; there is a conference devoted

to it, the International Conference on Auditory Display,[17] and it also appears in other converences such as the International Computer Music Conference.[18] There also exist a very small number of well-known industry applications, such as the geiger counter, which sonifies radiation levels using the rhythmic frequency of beeps.

A particularly interesting technique that we can use with sonification is audio spatialization. Spatialization is the processing of audio so that it seems to be coming from a specific location in space relative to the user. This might be achieved by using a large number of speakers in a room so that sound can in fact be panned around in three dimensions. However, processing techniques exist such that 3D spatialization can be achieved even for audio played through headphones.

3D spatializated sonification is a technique particularly well suited for the display of dense sensor network data. The spatial locations of sensors can be straightforwardly mapped to virtual locations for spatialization. As a complement to a 3D graphical environment, this can work particularly well, because unlike with vision, we can hear in every direction. In this thesis, I will explore this technique of 3D spatialized sonification as a way to convey sensor data.

This thesis builds on DoppelLab, an ongoing project of the Responsive Environments group at the MIT Media Lab.[8] DoppelLab is a cross-reality virtual environment, representing a space in the real world, and populated with visualizations of diverse types of data, which can be explored in real time or across large spans of past time. Currently DoppelLab uses the new MIT Media Lab (E14) building as its test case; it pulls in temperature, humidity, sound, and movement data from existing sensor networks in the lab, as well as sensor networks of our own creation. DoppelLab uses the architectural model of the space as the framework for a 3D virtual environment for browsing, and allows users to explore real-time data, or to play back historical data at various speeds to see trends on different time scales; Figure 1-1 shows an example of the visualization inside the virtual space. DoppelLab aims to provide a *zoomable interface* where visualizations show the large scale contours of data from a zoomed-out view, and as the user approaches an object more detailed information appears. DoppelLab does not aim to be a closed-loop system which makes

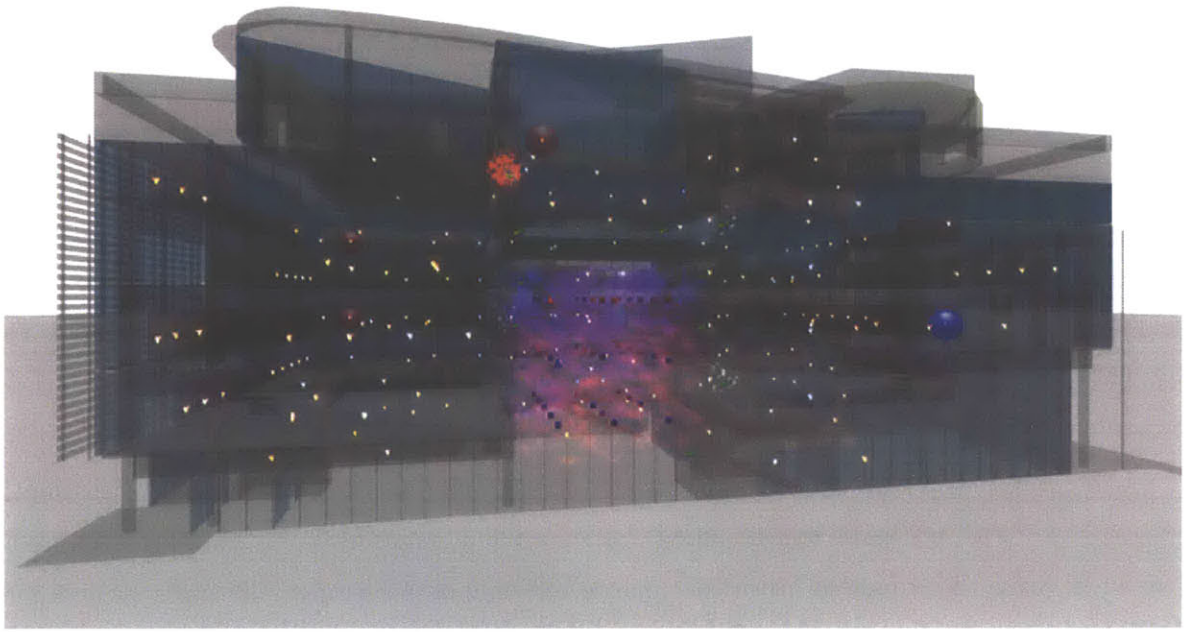


Figure 1-1: Full view of DoppelLab's representation of the Media Lab. Small orange flames visualize temperatures; red and blue spheres visualize temperature anomalies; blue and purple fog and cubes visualizes a dense array of temperature and humidity sensors in the building's atrium.

inferences from the data in synthesizes and controls something in the real space; rather, DoppelLab aims to provide a totally general browsing environment for spatial sensor data; the hope is that by bringing together many kinds of data and allowing the user to explore it at different speeds and vantage points, DoppelLab may allow users to make connections themselves which might not have been anticipated by its designers.

In this thesis, I build on DoppelLab to explore the potential of spatialized data sonification as a way to convey spatially oriented sensor data. I explore two kinds of data sonification: synthesized sonification of non-audio sensor data (e.g. temperature, movement and presence of people) and the use of recorded audio as sonification. I explore in particular several issues that arise in the latter case: how to respect privacy while using recorded audio from a shared space, and how to allow users to explore large bodies of audio data (time spans of days or months) efficiently. Not many high-

level tools exist for this kind of networked audio processing application exist, so my work also involved the design and implementation of this system.

The resulting system offers several ways to explore data through audio, through the existing DoppelLab user interface. Seven microphone streams from the public areas in the Media Lab, obfuscated for privacy, are played back in the application, spatialized according to the locations of their recording in the real space. As with the other data in DoppelLab, the user may explore real-time or historical audio. The default mode is to play real-time audio (within several seconds of strictly real-time, as a result of buffering in various parts of the system). DoppelLab provides a slider interface by which the user can select a historical time from which to play back data1-2; with my audio system, the recorded audio streams also play back archived audio from the requested historical time (at the time of this writing, the audio archive goes back several months and is continuing to accumulate.) Additionally, like the rest of DoppelLab, the program allows playback of archived audio at faster-than-real-time rates, using an audio time-compression algorithm of our creation. To protect privacy of users in the space, we obfuscate audio data at the nodes where it is recorded. In addition to this recorded audio system, I have implemented a number of synthesized sonifications of other building data; this sonification work is basic, but it allows us to explore the issues involved in sonifying spatially-oriented data, and doing so in conjunction with a graphical 3D interface.

In Chapter 2, I will discuss prior art in related areas, such as sonification, audio privacy, and audio analysis. In Chapter 3, I will give a brief review of what spatialization is, how it can be done, and what solutions I use in my work. In Chapter 4, I will discuss how my work navigates data privacy issues which come with using recorded audio. In Chapter 5, I will describe techniques I used for distilling down large amounts of audio data, in order to make it easier to explore. In Chapter 6, I will discuss several synthesized sonifications of non-audio data that I made. Finally, in Chapter 7, I address the design and implementation of the software system which makes the recorded audio sonifications possible.

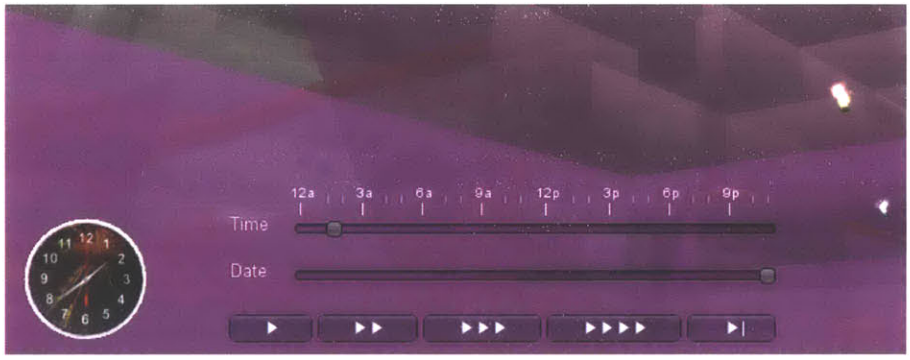


Figure 1-2: The GUI for DoppelLab's Time Machine functionality. Sliders allow coarse and fine selection of a new time to play back data and sound from; triangle buttons allow selection of playback speed for historical audio (2, 3, and 4 triangles denote speedup factors of 60, 600, and 3600, resp.), the 'forward' button returns to real-time data and audio.

Chapter 2

Related Work

2.1 Synthesized Data Sonification

Data Sonification is an active area of research, with work on perception of sonification, creation of tools for sonification, and creation of novel sonifications themselves. My work on DoppelLab relies on these perceptual studies as a basis for the effectiveness of musical sonification in general, and specifically some techniques such as pitch mapping and multi-channel techniques.

User studies have quantatatively shown the effectiveness of certain kinds of data sonifications for understanding features of graphs, both in seeing and blind users. Brown and Brewster performed a user study[5] where blind users were played sonifications which mapped functions of one variable to pitch, and were asked to draw graphs of the data; high accuracy was measured in terms of whether the drawings included features from the data which was sonified. In their work Musical versus Visual Graphs[11], Flowers and Hauer show that simple time-series visualization and pitch-mapping sonification were similarly effective for perception of data features such as slope and shape in time-series data.

Based on a number of perceptual studies like the above, Brown et al. published[6] a set of guidelines for sonification. The guidelines concern sonifications of one or several variables as a function of time, targeting blind users specifically. Among other guidelines, they recommend mapping the dependent variable to pitch in gen-

eral, and for multiple variables they recommend stereo separation and allowing the user to change relative amplitude levels; these latter two qualities can be effectively accomplished through spatialization.

In terms of specific sonification designs, we are particularly interested in the prior work which specifically concerns spatial data, audio spatialization, or both. This is discussed in Section 2.2.

2.2 Spatialized Data Sonification

An early precedent for this work can be found in R. Bargar's work in creating interactive sound for the CAVE system, which augments a graphical virtual environment with spatialized sounds.[3] Spatialization is accomplished using reverberation and delay and can be performed using speakers or headphones. This system is probably restricted by performance issues which are no longer problematic; for example, it can only spatialize 4 sound sources; the ability to spatialize many more than that is essential for our ability to sonify large sensor arrays.

Hunt and Hermanns Importance of Interaction [15] discusses how sonification is most effective in interactive systems with real-time feedback, which behave in ways that correspond to natural acoustic phenomena (e.g. when sound is produced by striking an object, a harder strike produces a louder sound.) 3D spatialization, a technique which I make heavy use of, is a good example of such a real-time feedback phenomenon. A main design criterion of my system is that it performs spatialization on the client, rather than the server; this removes network latency as a significant potential source of latency in the interaction, and allows for more realistic, real-time feedback.

Some work on sonification with an emphasis on spatialization exists. In their sonification of a Mixed-Reality Enfironment[21], Le Groux et al. sonify the positions of participants in the system, but spatialization itself is not used to directly convey spatial information, but as another effect to make the piece more immersive.

Nasir and Roberts's Sonification of Spatial Data[24] is a survey of sonification

work where either the data is spatial, the sonification uses spatialization, or both. One relevant conclusion they note is that spatialized sonification can enhance visualization of the same or related data.

Many of the spatialized sonifications of spatial data mentioned in [24] use data with fewer than three spatial dimensions; for example, for example, Franklin and Roberts[12] sonify pie chart data using horizontal spatialization; Zhao et al.[40] sonify geographical data which has two spatial dimensions.

Andrea Polli's Atmospherics/Weather Works[30] is a striking and ambitious example of spatialized sonification of data with three spatial dimensions. This work sonifies data from a storm, over a 24 hour period. It includes multiple data parameters, including pressure, humidity, and wind speed, sampled in a three-dimensional grid spanning the atmosphere and a $1000km^2$ horizontal area. All parameters were mapped to pitch, with different timbres differentiating kinds of data; a multi-speaker setup was used to spatialize the data according to heights at which it was sampled.

2.3 Privacy

Previous work from our group, the SPINNER project, [20] addresses the importance of privacy in distributed sensor networks, particularly with respect to audio and video data. SPINNER uses an opt-in model for sharing of audio and video data, where it requires building occupants to wear a badge, and recording is enabled only if a badge is in range and the user's preferences are set accordingly. If these conditions do not hold, or if recording is manually interrupted (by design, there are a number of easy ways to do this), audio and video recording is disabled entirely. This comprises a viable model for privacy, but in DoppelLab, we would like to avoid requiring users to wear badges, and losing significant windows of data, if possible. DoppelLab is primarily interested in audio as a way to sonify the usage of a space, especially compressed over large time spans, the semantic content of recorded speech is not of interest to us. As this is the primary sensitive aspect of audio data, we address the privacy issue by obfuscating the audio, with the goal of obscuring the semantic content while preserving loudness

and timbre to the greatest possible extent.

Some work exists on obfuscating speech content of audio while preserving timbre. Chris Schmandt's ListenIn [32] system uses audio for domestic monitoring among family members or caregivers. ListenIn detects speech, and scrambles the audio by shuffling short buffers whenever speech is detected, aiming to make speech unintelligible but otherwise preserve timbre. More recent work by Chen et al. [7] alters vowel sounds in speech, and includes a user study which demonstrates that it significantly reduces intelligibility while leaving concurrent non-speech sounds recognizable. While these works present interesting methods, they do not address the question of whether a third party could process the obfuscated audio and restore intelligibility. In [32], this is reasonable because the application is meant to be a closed system where data is shared among a small number of individuals; in [7], applications are not discussed.

In their work on Minimal-Impact Audio-Based Personal Archives, Lee and Ellis[9] address the issue of obfuscating audio in a way that would be difficult to reverse. Their application involves people carring portable recorders with them, rather than recording a space at fixed locations, but in both these applications, obfuscation which is difficult to reverse is important. Their method is similar to the one in [32]; they classify the audio as speech or non-speech, and then obfuscate the speech; in this case obfuscation is performed by shuffling, reversing, and cross-fading between short windows of audio. They claim that given certain parameters (shuffling 50ms windows over a radius of 1s, with "large" overlap between adjacent frames) would make reversing the obfuscation "virtually impossible". They note that this kind of obfuscation should leave spectral features in the audible range largely untouched, so that audio analysis would not be disrupted. For their application, Lee and Ellis claim that unobfuscated speech is the most interesting part of the recorded audio; they cite among other reasons a "nostalgia" appeal in listening to old conversations. They discuss the possibility of turning off obfuscation for speakers who have given the system "permission", possibly via voice recognition.

2.4 Audio Analysis and Time Compression

In this project, I use recorded audio from the space in order to help convey the usage of the space. In particular, I archive all that data and try to make it easy for users to explore the archived data efficiently. My general strategy for doing this is to allow users to seek around the audio archives and listen to versions which are radically time-compressed, by factors ranging from 60 to 3600. There exist large bodies of work related to time-compressing audio data, and exploring large bodies of audio data, but the main areas here differ from what this thesis attempts in significant ways.

There exists a large area of audio time-compression and content-aware time compression work, but it is focused on time-compressing recorded speech. This work differs from mine first in that in order to achieve this goal, relatively small compression ratios are used– generally less than 3. A bigger difference is that with this specific objective, there exist analysis techniques, based on speech structure, which can guide compression more effectively than a general (non-speech-related) algorithm could; also, there are more specific metrics for evaluation, based on speech comprehension.[26][14]

Music Information Retrieval (MIR) is a major area of research which aims to help users understand huge bodies of audio data. MIR, though, is typically based on search, while DoppelLab's aim is primarily discovery.

One piece of work with more direct relevance is Tarrat-Masso's work in [34], which uses spectral analysis to guide audio compression. This work, too, has a more specific problem domain than DoppelLab; it concerns time-compression for music production. However, the analysis and resynthesis are decoupled, and the analysis isn't specific to musical inputs. The analysis is based on Image Seam-Carving, [2] involves calculating an energy map of the spectrogram (a three-dimensional representation of an audio sequence, which plots power as a function of time and frequency) of the audio, and then applying the most compression to times when there is the least amount of spectral change. This is similar to the method that I will describe in Chapter 5,

although I build on it in several ways, including using a perceptual weighting of the audio spectrum.

2.5 DoppelLab

This thesis builds on the Responsive Environments Group's ongoing project, DoppelLab[8], which we introduced in Chapter 1. In this thesis I am exploring in particular ways to use sonification to complement 3D graphical visualizations, and to do so I will integrate my implementation with the existing DoppelLab system. Here I will review how DoppelLab approaches data visualizations, and how its system is designed.

DoppelLab aggregates and visualizes data from a large and still growing set of sensor modalities within the MIT Media Lab. An architectural model of the Media Lab serves as an anchor for these visualizations; all data visualizations are localized in virtual space according to the locations of their corresponding sensors in physical space. The building's structures themselves are by default visible but transparent, so that they highlight the physical locations of the data but do not obstruct the view. A common thread of the visualizations is the idea of *micro and macro design*; visualizations consist primarily of multiples of simple geometric forms, such that differences across sensors can be seen from a view of the entire lab, but display more detailed quantatative information when the user zooms in or clicks on them. One visualization shows temperatures throughout the lab based on over two hundred thermostats; warmer or cooler colors represent relative temperatures; numerical temperatures are displayed upon zooming. Another visualization shows both audio and motion levels in about ten locations; the audio level visualization uses the form and colors of familiar audio level meters; this visualization itself also oscillates, as if blown by a gust of wind, to indicate motion. Some visualizations show higher-order phenomena based on inferences made on more basic sensor data. For example, significant trends in motion and audio levels at a location can indicate the arrival or dispersal of a social gathering; DoppelLab shows clouds of upward or downward arrows when this happens.

A major part of DoppelLab's visualization power is an interface for exploring time. By default, DoppelLab shows real-time data. A heads-up display (HUD) (pictured in Figure 1-2) allows for exploration of past data; at the time of this writing, most of the data is archived for around one year. Using the HUD, a user can select a past time to visit, and once in historical data mode, the user can select among three faster-than-real-time speeds at which to view data. These speeds allow the user to view large-scale patterns which would not otherwise be apparent; for example, at the fastest speed-up of one hour per second, it is easy to see the temperature anomaly visualizations appear every night in a periodic fashion, indicating that the air conditioning system is overactive at these times.

DoppelLab's client is built with the Unity game engine[37]. Unity allows for rapid creation of visualizations, using a combination of a graphical user interface and C# and a JavaScript-based scripting language. The client has a main loop which queries the DoppelLab server every few seconds, receives sensor data from it in XML, parses the XML, and calls the appropriate callback functions to update the visualizations. The client maintains state indicating the current time and speed at which the user is viewing data, and that information is included in the server queries.

The DoppelLab server is written using Python and a MySQL database. The server has scripts which poll sensor data sources and aggregate the data in the central database. The server also includes a web server which responds to queries from DoppelLab clients, and returns the appropriate data from the database; for queries from clients running through historical data at a fast rate, the server computes and returns averages over correspondingly large ranges of data.

Chapter 3

Spatialization

Spatialization is the processing of audio to give the impression that it is coming from a specific location in space relative to the listener. In spatialization, we usually discuss a listener, which represents the simulated position and angle of the listener's head in virtual space, and one or more sources, which represent different mono audio sources, each with its own simulated position in space.

One way to implement spatialization is to set up many speakers so that sound can be panned between them; a typical way to do this in a sound installation is to have eight speakers placed in the eight corners of a rectangular room. However, to make it easier for more people to test our system, we focus mainly on using audio-processing-based spatialization which will work with a headphone setup.

This thesis is about way to apply spatialization, and thus does not focus on the implementation of spatialization itself. This chapter includes a brief discussion of what spatialization is and how it is often performed, and a discussion of the tools used in our implementation.

3.1 Review of Spatialization Techniques

There exist many methods which may be combined to implement sonification; within these there is a spectrum of levels of sophistication, which can lead to more or less realistic spatialization.

The most basic way to implement spatialization is with amplitude attenuation and panning between the ears. Amplitude is attenuated as a function of distance from the source, according to a variety of models. Panning between right and left channels simulates the angle between the source position and the listener's facing.

Spatialization engines often offer a variety of amplitude rolloff functions to the programmer. An inverse distance model, which corresponds with inverse-square attenuation of sound intensity, is typically offered; for example, it is the default option in OpenAL[28], the spatialization engine used in this thesis[27]. Other models are often available, though; for example, both OpenAL and Unity3D have an option where gain decreases linearly with distance, and Unity also has an option where the game designer can draw in an arbitrary gain function using the GUI[38]. These options exist because simple spatialization models fail to take into account other real-world factors, so a designer might find that cutting off gain artificially quickly at some point is a suitable approximation of other real-world factors, such as sound obstruction and other ambient noise. Another common modification is 'clamping', which imposes an artificial maximum on the gain function; this is important because in simple game models we might be able to get arbitrarily close to a sound source, which if an inverse gain model is used, can result in arbirarily high gain.

Panning and attenuation can offer us some cues in terms of left-right source movement relative to the head, and relative distance from the head, but other techniques are necessary in order to achieve full 3D spatialization. One common technique is the use of head-related transfer functions (HRTF). The HRTF is an experimentally-measured transfer function, which attempts to imitate the way the shape of the listener's outer ears, head, and torso filter audio frequencies that come from different directions relative to the listener.[4] The effectiveness of this technique is limited by the extent to which a given listener's anatomy differs from the model used in the creation of the HRTF; however, it has been shown that listeners can adapt to different HRTFs over time.[16]

Simpler kinds of filtering can also be used to achieve distance cues. In particular, applying a low-pass filter to sounds that are farther away simulates the fact that

higher frequencies attenuate more over these distances.

Spatialization can also be improved by using reverberation effects. Sound processed using a reverb algorithm is typical mixed with unprocessed sound in a "wet/dry mix", to simulate reflected and direct sound. Using relatively more dry sound simulates a sound closer to the listener. This can be done more effectively if a physical model of the room that the listener is in is used. Using effects such as these last two in addition to attenuation is particularly important for displaying distance; otherwise, it is unclear whether a sound is closer or just louder.

3.2 Using Spatialization in DoppelLab

One of the main design decisions I made in the DoppelLab sonification system was to perform spatialization on the client and not remotely; I discuss this issue more in Chapter 7. The client for display of recorded audio data is implemented in C as a Unity plugin. In the recorded audio sonification system, spatialization is performed using OpenAL[27]. OpenAL is cross-platform library for 3D audio and provides a simple interface in the style of the ubiquitous graphics library OpenGL. The specification of OpenAL states that the exact method of spatialization (e.g. HRTF) is implementation- and hardware-dependent[28].

I use the default distance attenuation model in OpenAL; this scales audio by a gain factor proportional to the inverse of the distance; since sound intensity is proportional to the square of gain, this corresponds correctly to the inverse square law of sound intensity in physics. We then set a reference distance which sets OpenAL's distance scale correctly according to how DoppelLab communicates listener and source information to it. The result is that audio streams' volumes are proportional to how they would be in the real world if no obstruction or other ambient sounds existed. These qualities are of course not maximally realistic, but they make some conceptual sense with the graphical representation in DoppelLab. By default, the architectural model in DoppelLab is rendered to be visible but transparent. The transparency of the architectural model can be toggled; a nice potential feature would be to also allow

toggling of the building's walls as acoustic obstructions.

The synthesized data sonifications in this thesis were implemented as Max/MSP patches[22]. For prototyping purposes, spatialization of those sounds is implemented within Max using a custom spatializer, which does a primitive spatialization using panning and amplitude attenuation. Section 6.1 contains more discussion of this issue.

Chapter 4

Privacy

The goal of obfuscating the audio streams is to protect the privacy of the occupants of the space while keeping as much useful data as possible. We consider useful data to be data which characterizes the activity happening in the space; for example, we should be able to hear if there is a lecture, a small conversation, or a loud gathering happening; we should be able to hear elevators, ping-pong games, and the clattering of silverware. To this end, we use a combination of randomized shuffling and reversing of grains (short sections of audio, on the order or $100ms$) on the time scale of spoken syllables.

4.1 Obfuscation

In order to prevent the privacy violation of recording, storing, and transmitting peoples' speech in a shared space, I aim to obfuscate recorded audio streams directly at the nodes where they are recorded. By 'obfuscate', I mean process the audio so that no spoken language can be understood. It is important that we perform this processing on the machines where audio is being recorded; otherwise we would be transmitting unprocessed audio over the network, where it could be intercepted by a third party. Also, we archive all of our audio data so that our application can allow users to explore data over larger time scales; it is important to only store obfuscated data, to protect it both from unauthorized users who have gained access

to our database, and from actual developers on the DoppelLab system. Encryption would not work as well as obfuscation for protecting the data during transmission and storage, because DoppelLab would have to be able to decrypt the data for audio processing and display, and thus DoppelLab developers would be able to gain access to the unencrypted audio.

For our obfuscation algorithm we have several requirements. We would like it to be unintelligible; that is, listeners cannot understand significant segments of it (i.e. more than an occasional word or two.) We could like it to preserve timbre, both of speech, and of environmental sounds, including the shape of transient sounds, and e.g. the number of speakers). Finally, we would like it to be difficult to reverse, or deobfuscate; we will discuss this idea further in Section 4.1.2.

Another requirement for the obfuscation algorithm is computational efficiency. In our current setup, the node machines where audio is recorded are primarily deployed for a different purpose[1]; therefore, we are not entitled to run particularly computation-intensive jobs on them. While that constraint may be peculiar to this particular deployment, it is reasonable in general that we would need an inexpensive algorithm. If we wanted to achieve higher microphone density, or if we were deploying in a situation where we didn't have access to computers around the space, we might want to run the algorithm on a microcontroller.

4.1.1 Algorithm

Our algorithm works in a similar way to the one presented in [32], but attempts to improve on timbral preservation. To that end, we shuffle fewer, larger grains, instead of a larger number of smaller grains. To offset the increase in intelligibility that this causes, we randomly reverse some of the grains. We also significantly overlap and crossfade between sequenced grains, to create a smoother sound.

The algorithm works as follows. We maintain a buffer of the `n_grains` most recent grains of audio, with each grain `grain_length + fade_length` milliseconds

[1]The node machines' primary purpose is to run instances of the Media Lab's Glass Infrastructure.[13]

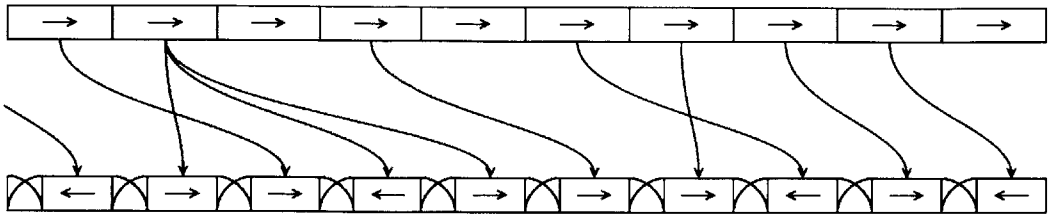


Figure 4-1: Visualization of the obfuscation process. In this example, `n_grains = 4` and `p_reverse = 0.5`

long, overlapping by `fade_length` ms. At each step, we record a new grain of audio from the microphone, and then discard the oldest grain. We then read a random grain from the buffer. With probability `p_reverse`, we reverse the audio we have just read. We then sequence that grain; i.e. we write it to the output, crossfading it with the last `fade_length` milliseconds of output. Figure 4-1 visualizes how the algorithm acts on several grains of data.

Currently the algorithm is deployed with the following parameters: `n_grains = 3`, `p_reverse = 0.6`, `grain_length = 8192`, `fade_length = 2048`, where the latter two parameters are in terms of PCM samples. Logarithmic crossfades are used to keep constant power during crossfades; this sounds less choppy aesthetically and makes it less obvious for a decoder to detect where exactly grain transitions occur; as an exception, we perform linear crossfades when the transition would be in-phase. These parameters were chosen based on optimizing for quality of timbre while maintaining unintelligibility. However, prior work such as [9] suggests that more aggressive scrambling, i.e. shuffling among more grains of smaller size, while maintaining significant crossfades, would provide more of a guarantee against an eavesdropper reversing the algorithm; accordingly, we intend to revisit these parameters soon.

The audio output of this algorithm is unintelligible, by observation, and it preserves the timbre of many environmental sounds. For example, the distinctive bell sound of the Media Lab's elevators is preserved nearly perfectly– since this is a relatively constant tone which lasts several seconds, shuffling grains which are significantly smaller than a second has little effect on it. This corresponds to a nice general prop-

erty of the obfuscation algorithm: since we only shuffle grains of a certain length, significantly longer and shorter sounds are largely unaffected. Significant changes in the tone of speech can be heard; laughter, for example, is often recognizable. Very short sounds, such as the percussive tones of a ping-pong game in the atrium, are also recognizable. Such short sounds are sometimes less clear since the reversal can affect them, but if a sound repeatedly occurs (such as, again, ping-pong hits), probabilistic independence of grain reversal makes a near-guarantee that many instances will be played forward.

4.1.2 Deobfuscation

In the interest of privacy, it is important not only that our obfuscated audio is unintelligible, but that it is difficult to reverse the obfuscation procedure. Otherwise, a user could download and archive the audio streams and attempt to process them so that the original audio is intelligible; either by manipulating it manually using an audio editor, or by writing a program which analyses the audio and attempts to reconstruct contiguous speech passages.

Currently, we do not know of a way to prove that the obfuscation would be practically impossible to de-obfuscate, or even exactly what that condition would mean. We can make some conjectures. Reversal of the algorithm, if possible, would probably involve either spectrally analyzing grains and finding grain boundaries which seemed to match up, or applying phoneme detection and then using a phonetic model to look for common phoneme sequences. Significant crossfading will make both of these approaches difficult, particularly the first one– with a large logarithmic crossfade, as we get near the edge of a grain, the spectrum of that grain will be mixed close to equally with the spectrum of the next grain; if the time scale on which syllables change is similar to the time scale of the crossfade size, it will be difficult to undo that corruption. If cross-fades comprise a large proportion of the duration of a grain, it will be more difficult for phoneme detectors to identify phonemes.

Another property of the algorithm that we note is that if we randomly sequence grains from a set of the most recent N grains, then a given grain will be in the set

during N sequencing intervals, and then the probability of that grain never being played is $((N-1)/N)^N$. If we maintain a set of 3 grains, this probability is approximately 0.30; if the set has 10 or 20 grains, the probability is approximately 0.35 or 0.36. Thus, approximately one of every three grains will be dropped. Thus, for an algorithm reversing this process, only short contiguous sequences of grains would even exist, and then if those were all identified, the algorithm would have do recognize words with 1/3 of the audio missing.

Chapter 5

Time Compression of Audio Data

One of DoppelLab's main features is the ability to explore historical data on faster-than-real-time time scales (up to several thousand times) in order to understand larger-scale patterns. My focus has been on using recorded audio data from the Media Lab to provide information about activity within the space. Unlike many graphical visualizations, which can be sped up simply by fetching sequential data at a higher rate, audio data is not trivially sped up. Since we perceive audio data in terms of frequencies, speeding up the data by the kind of factors we deal with in DoppelLab (e.g. 60 to 3600) would bring the data out of the human range of hearing. I have designed and implemented an algorithm for speeding up audio data for this application, which has two parts: one part uses a method called granular synthesis (defined in Section 5.2), which is used to resynthesize audio from any time offset in the source audio without altering its frequency; the other part determines which input audio we resynthesize from. I experimented with both traversing the input file at a constant speed, and traversing at a variable speed, where I spend more time on audio which has more interesting features. I perform some spectral analysis to determine which sections of the audio are more or less interesting.

5.1 Analysis

DoppelLab involves speeding up audio by several orders of magnitude; in the current application, by factors ranging from 60 to 3600. To compress sound by such a high ratio, it is necessary to discard some data. If we proceed through the sound at a constant rate, we will lose information equally from all sections of the data. If we can identify sections of the data which are more interesting to us, we can apply less steep compression to those sections, making it more likely that they will still be recognizable and meaningful to the listener.

Of course, the question of which audio is more interesting is very open-ended. Given the exploratory nature of DoppelLab, we do not want to restrict the user to a specific type of event, such as human speech or activity. Also, we do not want to simply select audio that has more noise or activity, as this could create an innaccurate representation of certain data. For example, if an interval of time has both loud conversation and silence, we would like to represent both. To accomplish this, we use a perceptual frequency scale to get a compact representation of the audio features that humans perceive in greatest detail. We then look at the amount of change in this representation over time, and bias the compression to preserve these times of transition, and apply more compression to times when the sound is more constant.

5.1.1 Bark Frequency Scale

The Bark frequency scale, proposed by E. Zwicker, is a subdivision of the audible frequency range into 24 distinct frequency bands, corresponding to the experimentally measured *critical bands* of human hearing. The division into these critical bands is thought to be closely related to the perception of loudness, phase, and other auditory phenomena.[41]

5.1.2 Audio Interestingness Metric

Given a Bark vector as a perceptually-scaled representation of the audio spectrum, we consider the magnitude of the derivative (or difference between vectors from two

consecutive windows) of the Bark vector as a metric representing how much change is happening in the audio signal. Since the Bark vector has 24 components corresponding to frequency bands, measuring the magnitude of the derivative will capture many meaningful changes in ambient sound as a listener would perceive it; e.g. changing pitch of a pitched sound, introduction of a new frequency, change in volume, increase or decrease in noisiness of the signal. Therefore the magnitude of the derivative constitutes a metric for how much perceptible change is happening; for lack of a more lexically pleasing name I refer to it as the "Interestingness metric".

To compute the Interestingness metric over a signal, I compute the FFT at contiguous, nonoverlapping windows of some window size. From the FFT, which has many frequency bands compared to the Bark (i.e. thousands or more), we need only sum the power of each frequency between successive Bark band edges to get the power in each Bark band.

Note that the window size over which we compute the Bark vectors is critically important, and is related to the time-scale of events that we could expect the listener to perceive at given time-compression ratios. For example, suppose we are compressing an hour of audio into a minute. We might be able to show that a conversation took place over several minutes. However, we probably don't care about the sonic details of every phrase or sentence– we only have a minute in total, and there may be other interesting events which will get encoded. If we use a 'typical' FFT window size for audio analysis, e.g. 1024 or 4096 samples, this will register significant spectral change as the conversants' syllables change and their sentences begin and end. On the other hand, if we use a larger window size, e.g. 10 seconds or a minute, the Bark vector will be averaged over this time, and will smooth over the small sonic changes that happen during a conversation. However, it will register change on the beginning or end of a short conversation.

To interface with our granular synthesizer, we want to provide a map indicating, for a given time offset in the audio output, where to draw a sample from in the audio input. Given the interestingness metric, we would like to spend the largest amount of playback time on the most interesting data; therefore we invert the metric data to get

the desired playhead speed. We then integrate with respect to time to get something proportional to how the playhead position in the input file should vary with respect to position in the output file. Since the map's domain is output file position and the range is input file position, and the beginnings and ends of the input and output file should match up (since we want to compress the entire input and nothing more, and we want to move through it monotonically), we scale the map so that the domain and range correspond respectively to output file length and input file length.

5.2 Synthesis

This section describes the algorithm which synthesizes time-compressed audio. We use a technique called granular synthesis (detailed in the following sections), which allows us to resynthesize audio from a recording and warp playback speed while not altering frequency.

5.2.1 Granular Synthesis

Granular synthesis[31] is a type of audio synthesis that involves creating sounds by sequencing and layering many short 'grains', or samples typically of duration 1 to 100 ms. Sometimes grains are synthesized; we use a technique called "time granulation", in which grains are created by sampling an existing audio source. By sampling grains around a specific time in the audio input, or a 'playhead', we can capture the timbre of that moment in the audio. We can then extend that moment arbitrarily, by keeping the playhead in that location.

Time granulation employs a variety of techniques to avoid artifacts when resynthesizing audio (or create them, at the programmer's discretion.) A naive implementation, simply creating a short loop at the location of the playhead, will have artifacts; longer loops will create an audible rhythmic pulsing, and shorter loops, where the period of repetition is above $20Hz$ or so, will produce an audible low-frequency hum. Granular synthesis solves this problem by windowing and layering together many short samples, or 'grains', using some randomness in the length and start time in

order to avoid artifacts.

5.2.2 Compression Algorithm

I tried two approaches to audio time compression using granular synthesis. In the first, the playhead moves through the input file at a constant speed. In the second, I try to identify moments of interest in the audio (using the analysis described in Section 5.1) and bias the playhead to move relatively more slowly during those parts, so that the interesting moments of transition can be played back with higher fidelity, and longer, monotonous sections can be passed over quickly.

The compressor program is written so that these different playback patterns can easily be swapped in and out. The compressor takes as an argument a 'playhead map'; a function which indicates, for a given moment in the output file, where the granular synthesis playhead in the input file should be. So for the constant-speed playback, the playhead map is a linear function which returns an offset proportional to the input by the compression ratio. For the variable-speed playback, then, the offset map should be a monotonically increasing function which increases more slowly during areas of interest in the audio. In Section 5.1.2 I describe how that map is created.

During synthesis, grains are sampled at the location of the playhead, with a random duration within a specified range. Grains are then multiplied by a three-stage linear window to avoid clicks, and added to the audio output. Figure 5-1 shows pseudocode for the main loop of the compression algorithm, given a playhead map. (I use python/numpy-style syntax where `array[start : end]` can be used to access a subarray of an array, and operators like `+=` distribute element-wise if operands are arrays.) Figure 5-2 shows a flow chart of how the analysis and synthesis comprise the time-compression functionality.

```
HOP_SIZE, MIN_GRAIN, MAX_GRAIN, OUTPUT_LENGTH =
    [0.03, 0.1, 0.5, 60] * 44100
output_offset = 0
while output_offset < OUTPUT_LENGTH:
    grain_size = random int in range (MIN_GRAIN, MAX_GRAIN)
    new_grain = audio_in[playhead_map(output_offset) :
        playhead_map(output_offset) + grain_size]
    apply three-stage linear window to new_grain
    audio_out[output_offset : output_offset + grain_size]
        += new_grain
    output_offset += HOP_SIZE
```

Figure 5-1: Pseudocode for main loop of granular synthesis algorithm. Numerical parameters are specified in samples; accordingly, HOP_SIZE is 30 ms, etc.

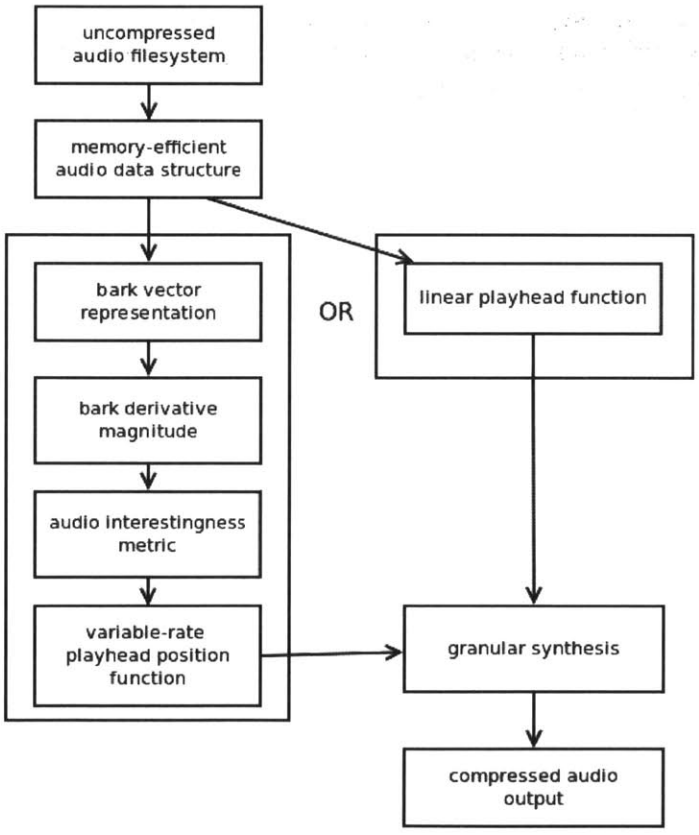


Figure 5-2: Flow diagram of audio analysis and compression algorithms

5.3 Results

In this section I show some data from intermediate steps and output from the time compression algorithms, and discuss some informal testing by myself and a few colleagues.

5.3.1 Data

My process for variable-rate time compression involves a number of steps; I will use some data to demonstrate how it works. Figure 5-3 shows a section of input data and successive representations, for compression ratio 60. The input data are chosen as an instance where we have some interesting events which we would like to show in higher resolution; the beginning is mostly quiet, we have a few brief periods when people walk by and talk, and then near the end the beginning of a musical jam session can be heard. All four subfigures are on equivalent time scales, so corresponding data line up vertically. The first subfigure is a spectrogram showing the 60 minutes of input data; a spectrogram is a plot which shows successive windows of spectral representation over a longer signal; power is mapped to color. The second subfigure shows, for successive windows, the values of the Bark vector. The third subfigure shows the Interestingness metric over the same data, or the magnitude of the derivative of the Bark vector; the fourth shows the playhead map, which indicates the location in the input file at which the granular synthesizer should be sampling from for each moment in the output file. Note the several more flat regions; these are moments of interest where the playhead spends more time.

Figure 5-4 shows spectrograms of the minute-long audio outputs from contant- and variable-rate time compression, given the hour of input data shown in Figure 5-3. The first image, predictably, looks similar to the original data in the first subfigure in Figure 5-3, since we are compressing using a constant speed, and since the images don't have resolution to show the details in the hour-long sample which don't exist in the compressed version. In the second subfigure, as intended, we can see many of the same spectral features, but time is stretched or compressed at different moments in the

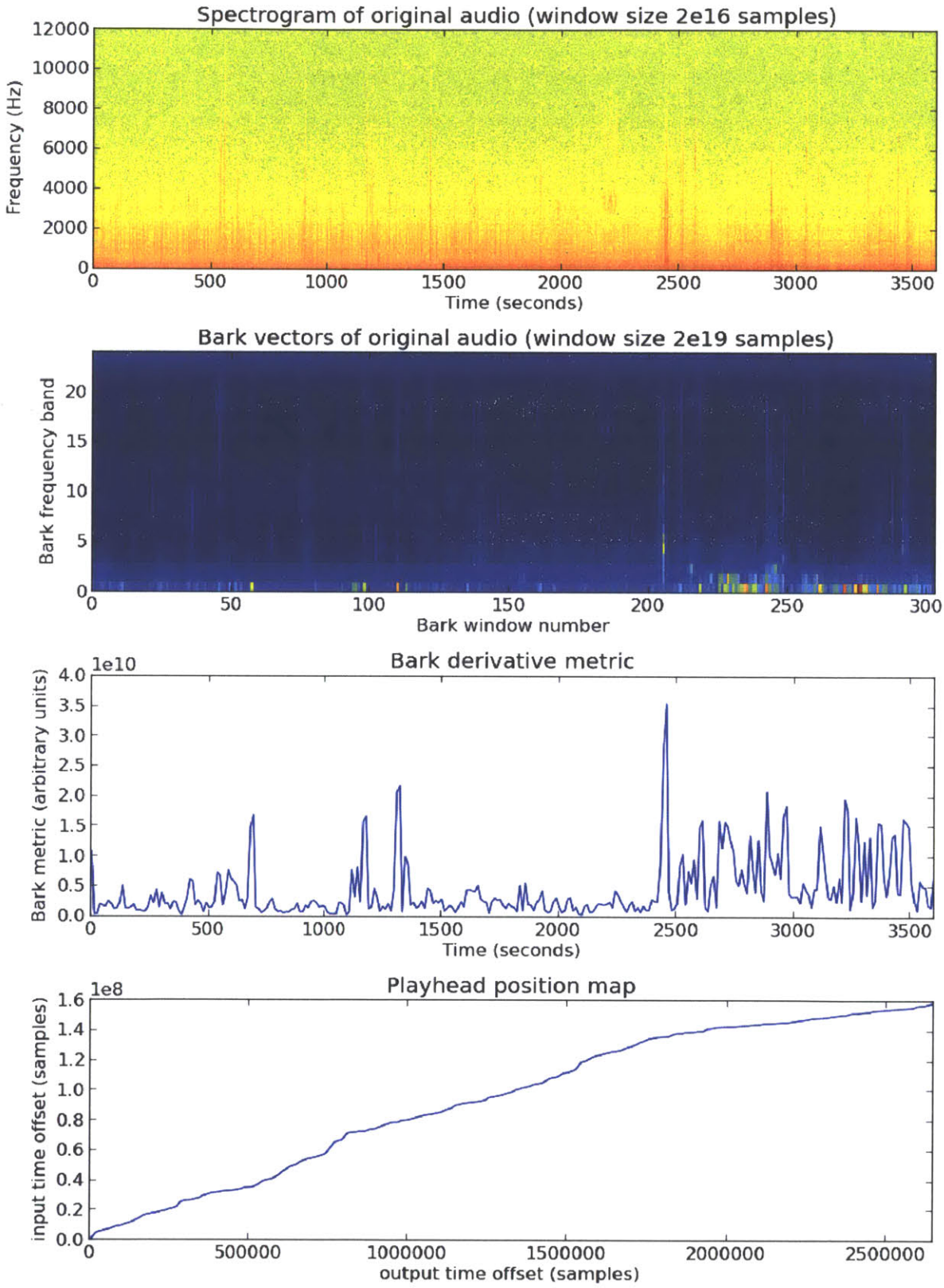


Figure 5-3: Analysis of 60m audio data: input and successive representations.

audio. The "event" annotations point out a few moments where we can clearly see the same audio feature showing up in both outputs, based on the spectral characteristics. In particular, with "event B", we note that the event, which is brief, barely shows up in the first subfigure, and is even more difficult to see in the input data. In the second subfigure though, the event is bigger and darker, and farther away from the peak immediately to the left of it. Similarly, with "event C", we see a pair of short, dark peaks, and some lighter peaks to their left; in the second subfigure, all this dense activity is more spread out in time. These two examples illustrate the variable-rate compression algorithm dwelling on events with more change, or with unusual frequencies, over relatively longer periods of time, thus showing them in higher resolution. This is at the expense of more monotonous sections, like the several more static periods early in these spectrograms.

Note that slight variations in the outputs of the different algorithms (e.g. in Figure 5-4) can appear apart from the main time-warping effects. This can result from the randomized nature of the granular synthesis algorithm; since grain length is randomized within a range, successive longer or shorter grains can produce a small fluctuation in intensity. In addition, large-scale changes in how the playhead for grain sampling moves through the input file will affect, on a grain-sized scale, exactly which grains in the input are sampled.

5.3.2 Testing

The first evaluation I did of this compression was testing it myself. In general, I was surprised by how good even the constant-rate compression sounded; because of the granular synthesis, social gatherings and music in particular were often smoothed into a sped-up, but still recognizable, soundscape. When comparing the two methods side by side on the same data, I did find that the variable-rate compression sounded like it spread out activity in the audio more. The effect was particularly noticeable on relatively short sounds; with constant-rate compression, these sounds could come out sounding clipped, probably having been only sampled in one or a few grains. In the variable-rate compression, the same sound was sometimes played slowly enough that

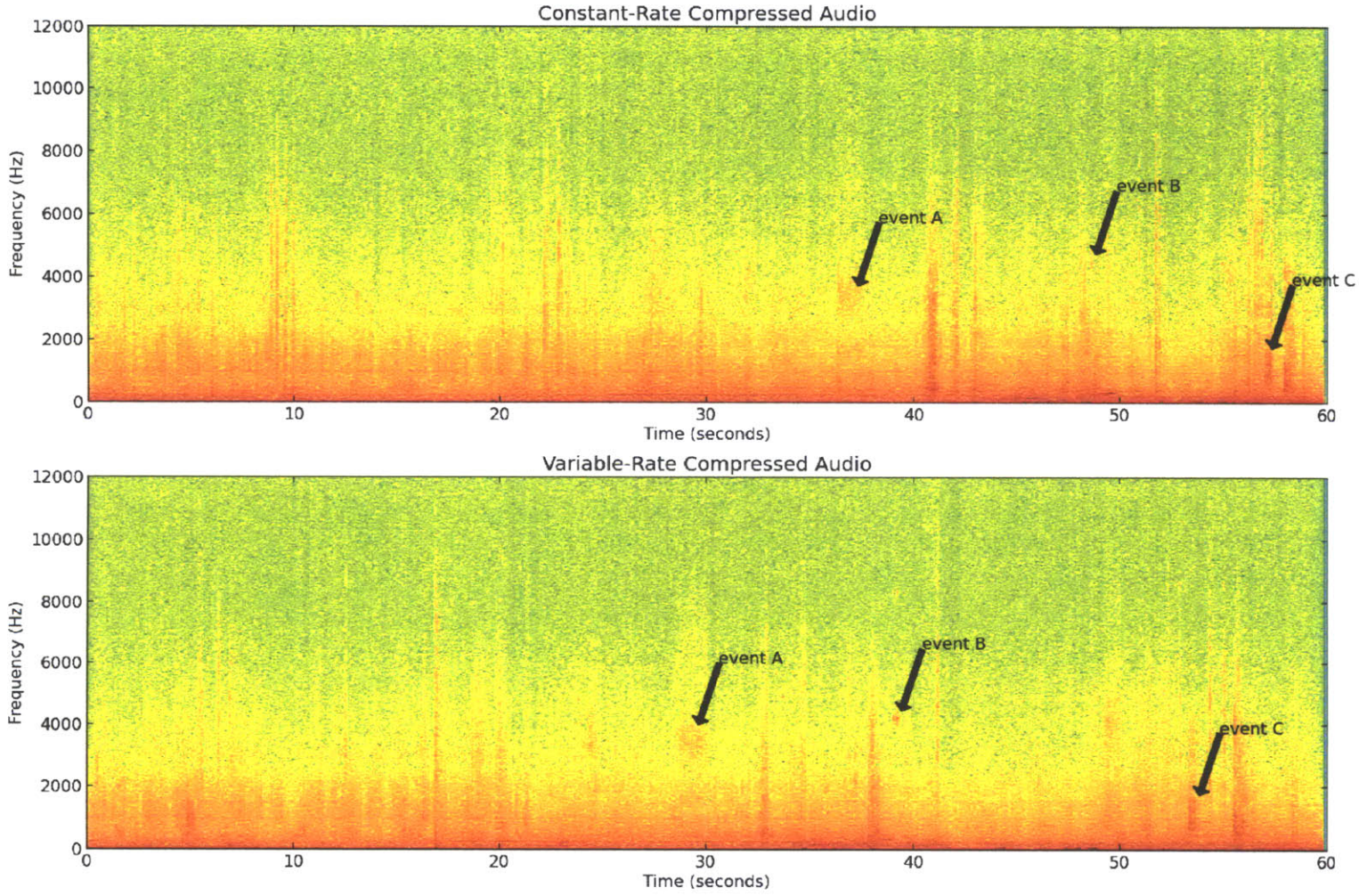


Figure 5-4: Comparison of outputs of constant- and variable-rate time compression. Compression ratio is 60; input is the input data from Figure 5-3

its natural contour could be heard; making it more recognizable.

I then did an informal user test with two of my colleagues; Participant 1 was familiar with the project beforehand, and Participant 2 was not. For the test, I asked subjects to listen to four minute-long clips of time-compressed audio; output of both the constant- and variable-rate compression algorithms, run on 60 minutes of audio with compression rate 60, and then on 600 minutes of audio with compression rate 600. For each of the two compression ratios, I first played the constant-rate output, and asked the participant what they heard; then I played the variable-rate output and asked what differences they noticed. The audio clips were both chosen from a week night at the lab, including conversations, a musical jam session, and custodians vacuuming.

Of the constant-rate compression on 60 minutes, Participant 1 thought they heard several-minute-long conversation, 2/3 of the way through, and the "usual background noise" of the Media Lab, with doors opening and closing, and a hint of something melodic towards the end. Participant 2 noted that it sounded like a room with conversations and silences, and that it sounded "natural", as opposed to sounding like a compression. Both participants reported hearing more activity during the variable-rate compressed audio. Participant 2 said that these sounded more "full", and that it sounded like there was a boost in mid-range frequencies. Participant 1 noticed that during the 10-hour compressed audio, he heard what sounded like a sequence of chords, which he speculated be the dominant harmonies of a series of songs; on hearing the variable-rate compressed version, he reported hearing more 'attack', and more of the transients, during the musical sequence.

This testing suggests that in general, the compression is successful in conveying main events which occur in the lab, as well as the general sonic character of the space. However, the difference made by the variable-rate compression was more subtle; in some cases, users seemed to hear it as a change in timbre rather than something which enabled perception of new effects. Also, while comments suggest that the variable-rate compression is working, by finding more activity, it is not clear that more activity is desirable; perhaps the distilling of audio to times of greatest change

could be accompanied by a smoothing or averaging of timbre. This work could benefit greatly for a more substantial user study, in which different parameters (especially more extreme parameters for variable-rate compression) could be tested.

Chapter 6

Sonification of Non-Audio Data

In addition so displaying recorded audio data, I did some basic work on sonifying non-audio data in the context of DoppelLab. My objective is to make sonifications which are useful in the way that good visualizations are useful. To this end, I try to make the mappings from data to sound simple and transparent. I also give particular consideration to making sonifications which make effective use of spatialization.

6.1 Implementation

The synthesized sonifications I made used data that the DoppelLab server was already archiving and displaying. For that reason, and because the other data in general is much lower-bandwidth than audio data (so e.g. it can be easily passed around using Open Sound Control (OSC)[29], and does not require data compression), implementation of this portion of the project was relatively simple.

DoppelLab has a database of all the data it displays on a central server, and a web server which interfaces with that database. For all modes of behavior (real-time, historical, faster-than-real-time playback), DoppelLab gets data by making HTTP requests to the database server. To sonify a given type of data, I would add a script to DoppelLab which would get the XML of the relevant data as it came into DoppelLab from the server. Where necessary, I would perform processing on the data within DoppelLab, and then I would send it as OSC to a local port. Then, I would have

Max/MSP[22] patches which would listen on that port and parse the OSC data. The actual sonification takes place in the Max patches. DoppelLab also sends position data of the player and the data sources to the Max patches, and spatialization takes place within those patches.

There are a few issues with this implementation. One is that as it currently works, spatialization of the sonifications is performed within Max/MSP. While sophisticated spatializers have been implemented in Max/MSP, the problem is mainly that this necessitates using a different spatialization system than the microphone sonification (the latter uses the OpenAL API[27].) As discussed in Chapter 3, spatialization engines use a variety of techniques to simulate sounds in 3D space. Therefore two different spatializers could apply different effects to two sounds which were colocated; hence, use of different spatializers could distort spatial perception or detract from the illusion they are meant to create. One solution to this problem would be to use an audio routing library such as JACK[19] to route audio from Max into another C program which would spatialize them using OpenAL.

Another issue is that this sonification implementation is not easy to deploy with the rest of the system. While the microphone sonification client can be compiled into the DoppelLab executable, the Max patches must be executed separately using the Max runtime.

6.2 Mappings

I prototyped and tested several sonifications of different modalities of DoppelLab data. I aimed to try sonifying diverse types of data which would require different kinds of mappings. In particular, I considered categories of continuous data, where sensors have numerical values which vary over time, and event-like data where sensors register discrete events as they occur.

Some restrictions to the kinds of mappings we make are helpful in terms of integrating with spatialization. For instance, loudness is a basic parameter which may be used for sonification; increasing volume for a new or anomalous event in the data

might be an intuitive mapping. However, as loudness also corresponds to spatial proximity in a spatialized system, a source getting louder would be difficult to distinguish from a source getting closer.

6.2.1 Continuous Data Sonifications

We can classify a number of the kinds of data we sonify in DoppelLab as continuous data. These are data types where each sensor gives a numerical parameter which always has a value that varies over time. This includes many kinds of physical data, such as temperature, humidity, and audio levels.

To explore this type of data, I made a simple sonification of temperature. DoppelLab incorporates streams of temperature data from a network of between two and three hundred temperature sensors around the Media Lab. I pick a subset of those sensors and assign a sine-wave oscillator to each, spatialized at the corresponding locations. I map their temperatures proportionately to the frequency of the sine waves.

One issue I encountered is that due to the performance of Max/MSP I had to limit the number of sine wave oscillators I was using to 16. I tried two different methods for choosing the set of thermostats to sonify: one method involved continually updating a list of the closest 16 sensors; the other involved simply sonifying an arbitrary set of 16 thermostats. While the first method gives the listener more control over which sounds they hear, in testing I found that it gives less of a sensation of spatial depth, since the thermometers that sound are all close enough to be close to be similarly loud. A more efficient implementation might mitigate the problem of having to select.

The resulting soundscape has a very resonant, bell-like character. The lack of pitch quantization and the large number of voices give an effect reminiscent of later 20th century music such as the micropolyphony of Gyorgi Ligeti. While the continuum of pitches is at odds with the suggestion of discrete pitches in [6], it allows for nuanced presentation of this dataset, where at real-time or near-real-time speeds, data often changes very gradually.

6.2.2 Event-Like Data Sonifications

Another major category of data in DoppelLab is event-like data. This includes data where discrete events happen, associated with a physical location and point in time. For this kind of data, we might use a sonification where we assign a transient note or sample to the events; this way, the density of sounds in time, or large-scale rhythmic structure, encodes large-scale patterns in the data. If data is sparse, the sonification can act as an alert to indicate the presense of new data. I tried sonifying two such types of data; RFID data and Twitter streams.

In both of these sonifications, I found the most salient data to sonify to be the username. Since this is not quantitative data, it cannot be directly expressed by mapping it to a musical parameter. I used hashing to associate the username with the chosen parameters; the goal is not to 'encode' the username in a meaningful way, but to have an association so that if a username appears frequently, the listener may learn to recognize the associated sound.

DoppelLab's RFID data provides a rough-granularity representation of where different lab members or visitors are in the building. Many members and visitors choose to carry a badge which has an RFID tag; if the tag is detected at one of the RFID readers around the lab, DoppelLab registers the event, and some associated data, such as a name, and if the tag owner has submitted one to the Lab-wide database, a photo. DoppelLab includes a visualization of this data, wherein a cube appears if a tag is registered in the corresponding location; if a photo is available, the photo appears on the cube. Figure 6-1 shows a screenshot.

To sonify an RFID event, I synthesized a simple, short bell-like tone. I associated usernames with pitch; pitches were chosen among multiples of a base frequency, for a just-intonation-like effect; tones in a scale could also be used. One important characteristic of the RFID data is that it has great variance in frequency of events; on a normal day, around 10 RFIDs might appear; during semesterly sponsor meetings, hundreds can appear within an afternoon; this density is compounded in the case of faster-than-real-time playback. In these cases, a single data request may yield dozens

Figure 6-1: Cubes with faces on them appear when RFID tags are near sensors.

of new RFID events; the sounds from such events are triggered sequentially, so the effect is a fast rhythmic pattern.

DoppelLab also includes Twitter data.[36] Public tweets from lab members are aggregated, and the several most recent tweets are rendered in the visualization, situated according to the office location of the account's author. Figure 6-2 shows an example.

For the Twitter sonification, I used samples to denote events; I chose a set of bird call samples, for aesthetic interest, and to provide a clear association between the sonification and the data modality in the presence of other sonifications. The greater sonic variety between the bird samples, as compared to the synthesized RFID sounds, made it relatively easier to associate the sounds with stream authors. On the other hand, this complexity can be problematic at the higher frequencies of events; many such samples sounding simultaneously is difficult to sonically parse.

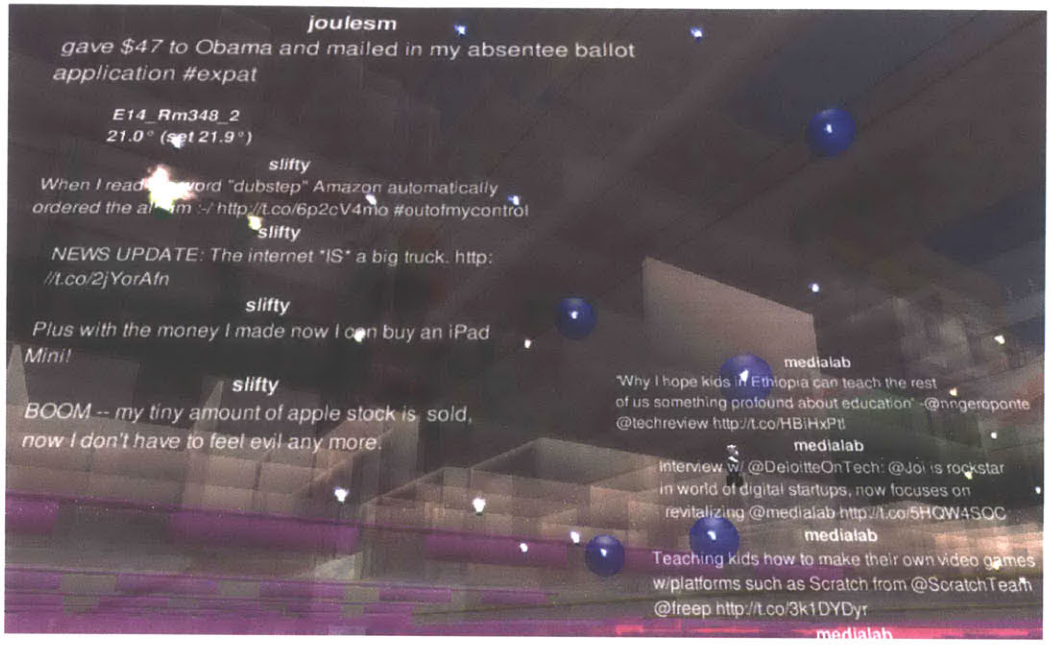


Figure 6-2: Twitter streams are rendered in space according to the offic location of their authors. They update when new tweets appear.

6.3 Results

I informally evaluated these sonifications by testing them myself. Even though the work is preliminary, testing it yielded a number of insights particular to using sonification to augment a 3D graphical environment.

With event-like data (Twitter and RFID), one of the main goals of the sonification is to give the user an awareness of events which they are not currently looking at (either because the user's eyes aren't on them or because the UI's camera is not on them.) These sonifications are successful to the extent that they indicate the frequency and general spatial distribution of events. One problem I quickly noticed, though, was that connecting a sonic event to the specific visualization it corresponded to was very difficult. If I heard a sound, I would sometimes navigate to see what new data had appeared, based on the direction of the spatialization. Typically, a number of RFID cubes or Twitter streams would be present, and it would not be clear which event had just taken place. The Twitter stream visualizations are particularly

problematic because they are always present; new data simply changes the content they show.

Thus, the tests indicate that it is difficult to learn these purely associative sonifications. One way to improve this situation might be to make the sonification interactive through the GUI; another might be to allow occupants to submit meaningful sonic signatures which would be more meaningful than arbitrarily assigned sounds. These ideas are discussed further in Chapter 8.

One way to build the association between sounds and data for the user would be to make the sonification more interactive. For example, the user could click on an RFID cube or Twitter stream to hear the sound associated with a particular user. This would help the user identify which data the sound they just heard was associated with, and over time could help the user learn the association between at least some data and its sound.

Chapter 7

System Design

Significant infrastructure is needed to implement the recorded audio sonifications. Part of this is that the game engine we're working with doesn't provide low-level, flexible audio APIs. I have designed a modular and scalable system to allow the streaming sonifications to work. I hope that the infrastructure I created will be useful to others who are using large amounts of sampled audio data for sonification, and/or using game engines for visualization and sonification. In this section, I describe the server infrastructure, which includes recording, obfuscating, and time-compressing audio and serving it to clients; and the client, which downloads real-time, historical, and time-compressed audio and spatializes it.

Figure 7-1 shows an overview of how all the components in this application are related, and how they are integrated into the greater DoppelLab infrastructure, as a dataflow diagram.

7.1 High-Level Design Choices

One major design decision we made is to spatialize audio streams on the client. There exist reasons to perform spatialization on the server; most notably, the spatialization process takes in one audio stream per source and outputs a stereo mix, so computing it on the server can save bandwidth. However, spatializing on the client is important both for user interface and scalability. As [15] suggests, given that spatialization is

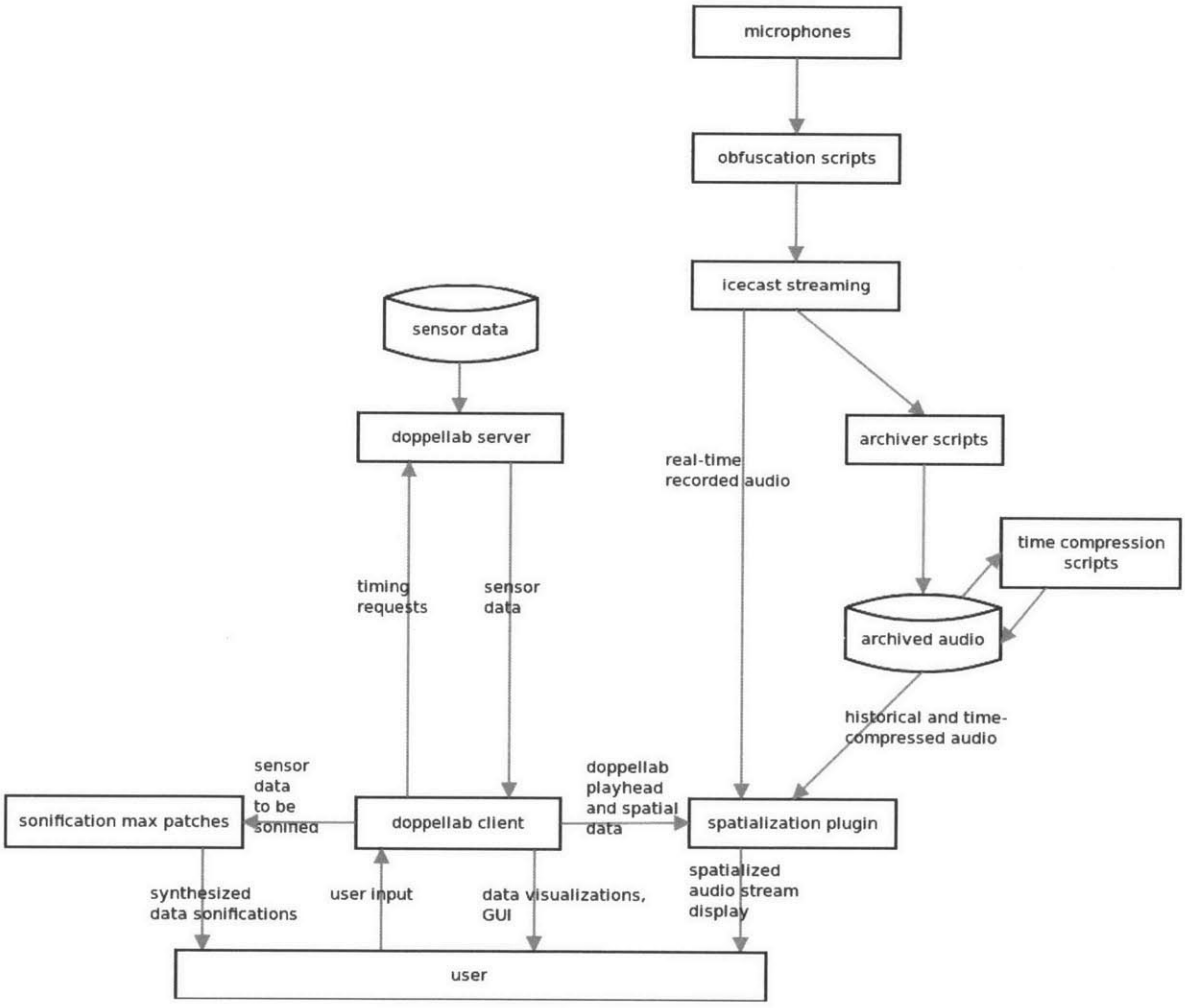


Figure 7-1: Data Sonification and its Integration with Browsing Environment

a simulation of a physical phenomenon, it is important that this effect is rendered in real time. Performing spatialization on the server would require sending player position to the server, then computing spatialization, and then sending the resulting mix to the client; this would therefore add twice the network latency. According to AT&T[1], average network latency between U.S. cities is 34 ms, so the delay would be about 70 ms; more for users farther away from our server. For reference, humans can detect fine-grained delays in audio, on the order of 40 ms; therefore, performing spatialization on the server would diminish its effectiveness. Additionally, spatialization is computed relative to player position, so if we compute spatialization on the server, we must use a separate spatialization process for each client; this would scale poorly with more clients.

Another major decision was to serve historical audio using a totally standard web server. Given that the real-time audio is served as a stream, we could have simplified the client's work by having the server sequence and serve streams of historical audio data according to clients' requests. This presents a similar scalability problem as spatializing on the server, though: we would need a seperate server process to create and serve different streams for each client which was downloading historical audio. By storing the audio as minute-long compressed audio files, we have small enough granularity that the client can work by downloading entire files at a time, and the server doesn't need to be able to serve partial files.

Scalability implications of these design decisions are discussed further in Section 7.2.1.

7.2 Server for Recorded Audio

We currently record audio in 7 locations around the lab. At these locations we have boards with electret microphones on them, as part of DoppelLab's red board infrastructure [8]. These microphones are recording to a set of commodity machines whose primary purpose is an information kiosk for media lab visitors. An efficient C program runs on each of these machines which reads audio from the microphone

input, obfuscates it (see Section 4.1 for details), compresses it using Ogg Vorbis[25], and sends it to an Icecast server that we run on our main sonification server.

We maintain a central sonification server which is responsible for serving the real-time audio streams, and preparing and serving historical and time-compressed audio data. We run an Icecast server here to serve the real-time Ogg data which is streamed from the recording nodes. To maintain the audio archive, we run one script per audio stream; this script continuously downloads and buffers the stream, and at each minute boundary, writes the most recent minute of audio data to a file. The audio archive is stored in a standard Unix file system; it is organized in a tree-like directory heirarchy, with a level for each unit of time (e.g. month, day, hour) and leaf directories which correspond to hours, each with the appropriate 60 minute-long audio files. The audio uses up a significant but easily maintainable amount of disk space; each minute of Ogg Vorbis data takes close to 500 KB, so if we have about 10 audio streams, we need 7.2 GB per day, which corresponds to one 2 TB drive every 277 days. We note that since the archiving scripts download data from the Icecast streams just like a client would, the archived data can be trivially stored and served on a separate machine from the one running the Icecast instance.

The server is also responsible for serving time-compressed historical audio data. Using the algorithms described in Chapter 5, we precompute and save to disk all compressed audio data (the application offers 3 time-compression ratios: 60, 600, and 3600, in addition to real-time; we note that the steepest ratio corresponds to one hour per second.) Since even the audio produced by the smallest compression ratio takes up only $\frac{1}{60}$ of the space of the normal-speed audio, and the higher ratios take yet less space, the pre-computation is not a significant issue storage-wise. Pre-computation is also critical for allowing users to request sped-up audio from different times in the past interactively; with the highest compression ratio, preparation of one minute of audio for a single stream requires processing 60 hours of audio data, or about 18 GB uncompressed, which would require extreme performance in order to serve without adding latency. We run the time compression algorithms as a batch process once per day to compress new audio; we store this audio in a tree-like directory heirarchy

similar to that which we use for non-time-compressed data.

The use of minute-long files makes recovery from server crashes relatively straightforward. If the recording nodes crash, the archiving scripts try to reconnect every minute, so recording resumes automatically when the nodes go back up. If the central server crashes, these scripts restart automatically on restart. Currently, if the archiving scripts fail to get the expected amount of audio data from the nodes, they write a copy of the last full minute recorded; this is in place particularly so that several minutes of missing audio will not disrupt the process of time-compressing a large span of data. Further testing could investigate whether looping recent sound is preferable to simply having silence during times of missing audio.

7.2.1 Server Scalability

One strength of our design is that to the extent that the server interacts with the client, it is implemented using very standard tools: real-time audio is served via Icecast, a popular media streaming server; historical audio, time-compressed or not, is provided using a standard file server. The more custom components of the server, the archiving and compression scripts, do not have to scale with the number of users. These systems are frequently deployed in popular applications; for example, media streaming servers like Icecast are used to implement internet radio. Our server uses these technologies in a way which is not wildly dissimilar from standard usage; if we have 7 microphone streams, then each client at any moment is either downloading 7 audio streams or downloading 7 500 KB files per minute.

In terms of expanding the scope of the recorded audio sonification, or deploying similar projects in different locations, scalability in the number of audio streams is also important. Unlike with scalability in the number of clients, in this case we do have custom software for which more instances must run if we have more audio streams, including the archiving and compression scripts. Fortunately, for each audio stream, these tasks are independent, so if necessary, we could have multiple servers running these jobs, all mounting the filesystem where the archive is located. Another possible bottleneck is the number of audio streams the client can download at once; this would

create a hard limit on the number of audio streams and is one limitation of our design. It is possible that in these cases, the client could send position information to the server and the server could only send audio streams which are closer to the client. If the set of streams the client is close to changes too quickly, the setup and teardown of download streams may be too much. However, a deployment with so many streams might also involve a much larger game world; in this case the client position might change relatively slowly enough that culling more distant streams is a viable strategy.

7.3 Client for Recorded Audio

Unity3D, the game engine on which DoppelLab is built, provides tools for using audio in ways that games traditionally do[38]; for example, audio tracks on disk can be looped as background music, or located in the 3D space and triggered by game logic as sound effects; these uses can take advantage of Unity's sound engine, which handles, for example, spatialization. However, these functions act on audio files, and not, e.g., remote audio streams; a lower-level (e.g. audio-buffer-level) API is not available. Unity does, however, have a plugin framework that allows game scripts to make calls against dynamic libraries, which can be compiled into a game executable. So to implement the client for recorded audio, I wrote a C program which handles audio streams outside of Unity, but with which Unity can communicate. The plugin handles downloading and spatializing real-time or archived audio.

7.3.1 Client API

The plugin presents a simple API for communication with DoppelLab's Unity code. The Unity code is where the user interfaces for exploring the virtual space, and exploring historical data, are located, and our audio stream sonification should respond to those controls. The main API functions include one to begin real-time playback; one to begin historical playback, given a time and a time compression factor as arguments; and and one to upate the player's position for spatialization purposes.

7.3.2 Client Implementation

The plugin is implemented with two main threads. The download thread is responsible for downloading real-time audio streams or historical audio, as requested, decoding it, and writing each audio stream to a corresponding circular buffer; the playback thread reads audio from the circular buffers and spatializes each stream with respect to the player's position. With this design, only the download thread has to know about requests for a new playback position or time compression factor. In addition to the two main threads, external requests, such as playing audio from a different historical time, execute in a separate thread. We implement the functionality around the playhead in such a way as to minimize mutable shared state, to avoid concurrency problems. Rather than maintaining shared state describing the requested audio playback position, whenever we have a request for a new playback time or speed, we increment a volatile `thread_id` variable which causes the previous download thread to terminate, and then we start a new download thread within the function call; this way all playhead request information can be separated by function scope.

Because the server serves historical audio as files instead of streams, the client is responsible for sequencing that audio correctly. Thus, when the application is playing back real-time data, the client plugin just downloads the audio streams and the Icecast server ensures that audio is downloaded at a rate appropriate to the playback speed. When the application is playing back historical data, the download thread downloads the first minute of audio from the appropriate time offset and compression ratio for each microphone stream, decodes the audio and writes it to the circular buffer. It then monitors the shared circular buffers, and then the buffers are almost empty, it downloads the next set of audio files. We note that for higher time-compression rates, it is important to have better resolution in our audio requests than just playing the corresponding minute of audio data. For example, if we are at the highest compression ratio, each minute-long file represents 60 hours of recorded audio; therefore, if the user moves the playhead in the UI to a certain hour, it does not suffice to begin playing audio from the nearest start of an archived file. Our solution is simply to download

the nearest predecessor file, in terms of time, and throw away the audio which comes before the requested time. We note that if we're throwing away data, we have to download that data before receiving data that we can start to sequence; this could cause a delay when the user requests playback from a new time or speed. In practice, though, the delay should be small: since we never have to throw out more than a minute of audio, if we have 7 500 KB files, we have to throw out between 0 and 3.5 MB of data at the start of a request. According to a 2012 FCC report [10], the average download speed tier for users in the United States was $14.3Mbps$; at this speed, a request for time-compressed historical audio would have an average delay of 1 second and maximum 2 seconds. We note that this delay has no bearing on, for e.g., the responsiveness of the spatialization, which is computed locally.

7.3.3 Deployment

Another factor in the decision to implement the client as a C plugin for Unity is ease of deployment. One strength of Unity3D is that all projects can be compiled to executables for Windows or Mac operating systems. Provided we compile binaries of the plugin which run on both platforms, the plugin can then be statically built into the DoppelLab binary. This way we don't have to run a separate program in parallel with DoppelLab to run the sonification. This is in contrast to the synthesized sonifications, which are implemented in a more standard way using Max/MSP, but as a result depend on that runtime (see Chapter 6 for details.)

There are still limitations in terms of deployment, though. Most notably, Unity programs can also be compiled to run on the web (although they depend on a Unity plugin.) Native plugins, however, will not work in this context. During the course of this thesis, exciting developments have happened in browser-based audio APIs. In 2011, both Google Chrome and Mozilla Firefox implemented separate (incompatible) high-level APIs; since then, the W3C has endorsed the Google-developed Web Audio API Specification[39], and Mozilla has stated that it will implement it[23]. This API includes support of 3D spatialized audio; once it has more widespread support, a port of the client to the web would probably be well within reach.

Chapter 8

Conclusions and Future Work

Increasingly, the spaces in which we live and work are instrumented with sensors. To understand this increasingly dense and multimodal data, we will need increasingly powerful ways to display it. This thesis combines spatialized sonification of spatial sensor data with interactive 3D visualization to demonstrate a new kind of immersive display. In addition, this work explores the use of recorded audio data as a timbral sonification of a space, and how to use extreme time compression to make this audio data understandable; these topics have seen little prior work. Finally, this work poses questions about data privacy which will become increasingly relevant, and takes steps toward finding a solution which best suits this new use of audio data.

This thesis touches on many rich topics, including audio analysis, compression, sonification, and privacy. There are various interesting possibilities for future work.

While some parts of this thesis, such as the Unity plugin, were implemented in a neat way that made distribution and testing practical, other parts of the system implementation were not as polished. Notably, the implementation of the synthesized data sonifications was not integrated as well with the rest of the system. Improving this implementation would make deployment easier and thus facilitate more user testing; also, it might lead to useful reusable designs for integrating game engines with sonification. One of the main issues with the data sonifications is that they must run separately from the unity code in the Max runtime. Possible solutions would include creating a system to compile Max/MSP patches to executables, or

implementing the sonifications in a language which allowed compilation to binaries. An especially good solution might be to reimplement all the client-side functionality using the Web Audio API; Unity can already target the browser, and integrating sonifications with the web build would be ideal for deployment.

The presentation of historical recorded audio would likely benefit from some visual cues. Currently, if the user hears a sound in DoppelLab and would like to replay it, they need to notice the time at which it happened and navigate back using the sliders. A simple visualization of the recorded audio in some radius around the region of current playback might allow users to recognize, or discover, interesting features. An amplitude waveform or a spectrogram might be appropriate.

One general issue with the exploration of time in DoppelLab is clock synchronization. Some synchronization issues have been noticed between the clock in DoppelLab's UI and the data visualizations; this could be due to the fact that DoppelLab downloads sensor data at 3 second intervals. The recorded audio streams seem to be well synchronized with DoppelLab's clock, up to the point of the delay caused by downloading the first minute of data from a new location (see Section 7.3.2 for details). For the user's experience, the synchronization between the audio streams and the graphical visualizations is particularly important. Future work should involve improving this synchronization. It is possible that visual cues for the recorded audio, as mentioned just above, could help in investigating this problem.

It would also be interesting to have a standalone client for exploration of the recorded audio at different times and speeds. In light of the lack of prior work on highly time-compressed recorded audio as a non-verbal sonification, it might be beneficial to pursue this aspect of the work independently.

Substantial literature on sonification and spatialized sonification exists; as a real-time interactive display of spatial sensor data, DoppelLab provides an interesting application. While this thesis included a few simple data sonifications, that subject could be explored much more deeply in general; more types of DoppelLab data could be sonified, sonifications could be combined (and combined with recorded audio sonifications), and the work could be done with a focus on the perceptual studies on

effective sonification.

One particularly interesting area is all the non-quantatative data in DoppelLab, such as the IDs of people who appear at the RFID sensors or on Twitter. ID data has hitherto only been sonified using associative sound mappings; more meaningful mappings might be possible by allowing people in the Lab to submit their own short musical nametags, or Leitmotifs. The way these tags could be created and submitted poses an interesting question; something general like allowing arbitrary samples affords people expressivity but allows the sonification designer little aesthetic control; a system where users entered a note pattern and chose among a curated set of synthesizer sounds would be another approach.

One way to make these associative sonifications more learnable, with occupant-submitted Leitmotifs or otherwise, would be to make the sonification more interactive. For example, the user could click on an RFID cube or Twitter stream to hear the sound associated with a particular user. This would help the user identify which data the sound they just heard was associated with, and over time could help the user learn the association between at least some data and its sound.

This thesis involved only informal user studies, in the form of testing by the author and a few labmates. Various aspects of this work could benefit from more rigorous testing, such as the methods of time compression and their parameters, and the extent to which the sonifications help users notice data features in DoppelLab.

This thesis prompts some deep questions about privacy with respect to recording audio in shared spaces. This thesis proposes that such audio data is useful and interesting. If we are to continue to explore uses of this data, we must pursue the questions of how irreversibility can be defined for obfuscation algorithms, and then how an algorithm can guarantee that it is irreversible. The next question after those is how we can optimize for preserving interesting data and aesthetics in such an algorithm.

The quality and character of the sounds created in this thesis are highly relevant. On the website[33] for this thesis, audio clips are available which demonstrate some of this work, including obfuscation and time-compression.

Bibliography

[1] At&t network latency report. `http://ipnetwork.bgtmo.ip.att.net/pws/network_delay.html`. Accessed: 2012-10-10.

[2] S. Avidan and A. Shamir. Seam carving for content-aware image resizing. In *ACM Transactions on graphics (TOG)*, volume 26, page 10. ACM, 2007.

[3] Robin Bargar, Insook Choi, Sumit Das, and Camille Goudeseune. Model-based interactive sound for an immersive virtual environment. In *Proceedings of the International Computer Music Conference '94*, pages 471–474, 1994.

[4] D.R. Begault et al. *3-D sound for virtual reality and multimedia*, volume 955. Ap Professional Boston etc, 1994.

[5] L.M. Brown and S.A. Brewster. Drawing by ear: Interpreting sonified line graphs. International Conference on Auditory Display, 2003.

[6] L.M. Brown, S.A. Brewster, SA Ramloll, R. Burton, and B. Riedel. Design guidelines for audio presentation of graphs and tables. International Conference on Auditory Display, 2003.

[7] Francine Chen, John Adcock, and Shruti Krishnagiri. Audio privacy: reducing speech intelligibility while preserving environmental sounds. In *Proceedings of the 16th ACM international conference on Multimedia*, MM '08, pages 733–736, New York, NY, USA, 2008. ACM.

[8] G. Dublon, L.S. Pardue, B. Mayton, N. Swartz, N. Joliat, P. Hurst, and J.A. Paradiso. Doppellab: Tools for exploring and harnessing multimodal sensor network data. In *Sensors, 2011 IEEE*, pages 1612–1615. IEEE, 2011.

[9] D.P.W. Ellis and K. Lee. Minimal-impact audio-based personal archives. In *Proceedings of the the 1st ACM workshop on Continuous archival and retrieval of personal experiences*, pages 39–47. ACM, 2004.

[10] Fcc broadband report. `http://www.fcc.gov/measuring-broadband-america/2012/july`. Accessed: 2012-10-01.

[11] J.H. Flowers and T.A. Hauer. Musical versus visual graphs: Cross-modal equivalence in perception of time series data. *Human Factors: The Journal of the Human Factors and Ergonomics Society*, 37(3):553–569, 1995.

[12] K.M. Franklin and J.C. Roberts. A path based model for sonification. In *Information Visualisation, 2004. IV 2004. Proceedings. Eighth International Conference on*, pages 865–870. IEEE, 2004.

[13] C. Havasi, R. Borovoy, B. Kizelshteyn, P. Ypodimatopoulos, J. Ferguson, H. Holtzman, A. Lippman, D. Schultz, M. Blackshaw, G. Elliott, et al. The glass infrastructureusing common sense to create a dynamic, place-based social-information system. *AI Magazine*, 33(2):91, 2012.

[14] Liwei He and Anoop Gupta. Exploring benefits of non-linear time compression. In *Proceedings of the ninth ACM international conference on Multimedia*, MULTIMEDIA '01, pages 382–391, New York, NY, USA, 2001. ACM.

[15] Thomas Hermann and Andy Hunt. The importance of interaction in sonification. In *In Proceedings of International Conference on Auditory Display (ICAD)*, Sydney, Australia, 2004.

[16] P.M. Hofman, JG Van Riswick, A.J. Van Opstal, et al. Relearning sound localization with new ears. *Nature neuroscience*, 1(5):417–421, 1998.

[17] International community for auditory display. `http://www.icad.org/conferences`. Accessed: 2012-10-09.

[18] International computer music conference. `http://www.computermusic.org/page/23/`. Accessed: 2012-10-09.

[19] Jack audio connection kit. `http://jackaudio.org/`. Accessed: 2012-10-03.

[20] Mathew Laibowitz, Nan wei Gong, and Joseph A. Paradiso. Wearable sensing for dynamic management of dense ubiquitous media. In *6th Int'l Workshop on Wearable and Implantable Body Sensor Networks (BSN 09)*, pages 3–8, 2009.

[21] S. Le Groux, J. Manzolli, and P. Verschure. Interactive sonification of the spatial behavior of human and synthetic characters in a mixed-reality environment. In *Proceedings of the 10th Annual International Workshop on Presence*, pages 27–34. Citeseer, 2007.

[22] Max/msp. `http://cycling74.com/products/max/`. Accessed: 2012-10-09.

[23] Mozilla page on web audio api. `https://wiki.mozilla.org/Web_Audio_API`. Accessed: 2012-10-03.

[24] T. Nasir, J. Roberts, et al. Sonification of spatial data. In *The 13th International Conference on Auditory Display (ICAD 2007)*, pages 112–119. ICAD, 2007.

[25] Ogg vorbis. `http://www.vorbis.com/`. Accessed: 2012-10-10.

[26] Nosa Omoigui, Liwei He, Anoop Gupta, Jonathan Grudin, and Elizabeth Sanocki. Time-compression: systems concerns, usage, and benefits. In *Proceedings of the SIGCHI conference on Human factors in computing systems: the CHI is the limit*, pages 136–143, New York, NY, USA, 1999. ACM.

[27] Openal documentation. `http://connect.creativelabs.com/openal/Documentation/Forms/AllItems.aspx`. Accessed: 2012-10-03.

[28] Openal specification. `http://connect.creativelabs.com/openal/Documentation/OpenAL%201.1%20Specification.htm`. Accessed: 2012-10-08.

[29] Open sound control. `http://opensoundcontrol.org/`. Accessed: 2012-10-10.

[30] A. Polli. Atmospherics/weather works: A spatialized meteorological data sonification project. *Leonardo*, 38(1):31–36, 2005.

[31] Curtis Roads. *The Computer Music Tutorial.* Massachusetts Institute of Technology, 1996.

[32] Chris Schmandt and Gerardo Vallejo. "listenin" to domestic environments from remote locations. In *Proc. the 2003 International Conference on Auditory Display*, pages 853–856, Boston, MA, USA, 2003.

[33] Doppellab sonification web page. `http://resenv.media.mit.edu/sonification`. Accessed: 2012-10-15.

[34] Josep Maria Tarrat-Masso. Adaptation of the seam carving technique for improving audio time-scaling. Master's thesis, Pompeu Fabra University, 2008.

[35] Edward R. Tufte. *The Visual Display of Quantitative Information.* Graphics Press Cheshire, CT, USA, 1986.

[36] Twitter. `https://twitter.com`. Accessed: 2012-10-03.

[37] Unity3d game engine. `http://unity3d.com/`. Accessed: 2012-10-09.

[38] Unity3d audio documentation. `http://docs.unity3d.com/Documentation/Manual/Sound.html`. Accessed: 2012-10-08.

[39] Web audio api specification. `http://www.w3.org/TR/webaudio/`. Accessed: 2012-10-03.

[40] H. Zhao, C. Plaisant, and B. Shneiderman. I hear the pattern: Interactive sonification of geographical data patterns. In *CHI'05 extended abstracts on Human factors in computing systems*, pages 1905–1908. ACM, 2005.

[41] E. Zwicker. Subdivision of the audible frequency range into critical bands (frequenzgruppen). *The Journal of the Acoustical Society of America*, 33:248, 1961.