MASTER IN COMPUTER SCIENCE

Sonification of time-dependent data

Demoulin, Christophe; Schöller, Olivier

Award date:
2001

Facultés Universitaires Notre-Dame de la Paix, Namur
Institut d'Informatique
Année Académique 2000-2001

Sonification of time-dependent data

Christophe Demoulin
Olivier Schöller

Mémoire présenté en vue de l'obtention du grade de Maître en Informatique

Abstract

Sonification is the process of transforming visual information into sound. This dissertation focuses more particularly on the problem of representing time-dependent data with sound. This kind of data often requires some preprocessing before any sonification mapping can be applied. Pitch-based mapping can then be executed on the data, and possibly enhanced with additional sonification techniques, such as slope indicators and extreme values detectors.

Two applications are developed in order to ease the sonification process of two-dimensional and three-dimensional data. Finally, an online questionnaire is presented for each application, by which the chosen sonification methods are evaluated.

Keywords: Sound, Sonification, Time-dependent data, MIDI.

Résumé

La sonification est le processus de transformation d'informations visuelles en son. Ce mémoire se concentre plus particulièrement sur le problème de la représentation de données temporelles par le son. Ce type de données nécessite souvent un pré-traitement avant qu'une quelconque transformation sonore puisse être appliquée. La transformation basée sur la fréquence peut alors être exécutée sur les données, et éventuellement enrichie par des techniques additionnelles de sonification, telles que des indicateurs de pente et des détecteurs de valeurs extrêmes.

Deux applications sont développées dans le but de faciliter le processus de sonification de données bi-dimensionnelles et tri-dimensionnelles. Finalement, un questionnaire en ligne est présenté pour chaque application, à partir duquel les méthodes de sonification sélectionnées sont évaluées.

Mots-clés : Son, Sonification, Données temporelles, MIDI.

Acknowledgements

First of all, we would like to thank our supervisor, Monique Noirhomme-Fraiture, for being there for us when we needed her. Despite the personal problems she went through, her door was always open and we were always welcome.

We would also like to thank our training course master, James L. Alty, for all his help during our three months stay at the Loughborough University. When we arrived, he had recently been promoted dean of the Faculty of Science, leaving him even less time than before. Nevertheless, we were able to come together on a few occasions. We particularly remember the first one, where we underwent a musical test for over half an hour, unfortunately without much success ...

Our gratitude also goes to the people who have answered the questionnaires and provided us with constructive and critical feedback for our applications. We know it required a fair amount of time and energy, but we can assure them that every single answer has been taken into account, and has contributed to the final results.

We want to thank all the people who have helped us during the making of this report. The technical problems were often solved via the Internet, which again has proven to be an invaluable source of help and information. The moral support was principally provided by Jennifer and Julie, who kept us focused and motivated all along the writing process. As for all the other persons who have helped and supported us, we sincerely thank them and hope they will recognize themselves.

Last but not least, we would like to thank each other for all the support, feedback and assistance we exchanged during the last couple of months. Working so closely with another person for so long can sometimes be tiresome and laborious, but even at difficult times we managed to forget our divergences and let our friendship prevail.

Contents

List of Figures

Chapter 1

Introduction

Since the dawn of computer science, the interaction between humans and computers has been merely visual. While the first computer terminals, equipped with monochrome screens and alphanumerical keyboards, provided limited modalities of interaction with the computer, most recent devices allow to display texts and pictures in several millions of colours and high resolution, to quickly render two-dimensional and three-dimensional graphs and to print them out on paper. This predilection of visual representation of data to the detriment of sonic representation exposes an explicit lack since the human treats both visual and sonic information simultaneously in the real world. From this observation is born the sonification, a new emerging field of research, which closes a gap between the usually complex visual computer displays and the very poor acoustic interface to the user. Sonification, sometimes synonymous with auralization, is an audio counterpart to visualization. By using audio rather than visual enhancements, the main goal of sonification is to provide more information while shifting additional cognitive load to a different modality. However, sonification alone can also be a useful technique for presenting information to visually impaired individuals.

The objective of the present dissertation is to study in which way sound can represent data, in particular time-dependent data, which are values varying with time, like a stock value on the exchange market or the audience of a television channel. As sonification still remains a recent field of research, the aim is to find, implement and evaluate efficient techniques to translate time-dependent data into sound, while maintaining the underlying characteristics of the time series.

Conceptually, the dissertation is divided into three parts. The first part offers a theoretical background related to the whole process of time-dependent data sonification. The second part introduces two Java applications that we have developed in order to sonify two-dimensional and three-dimensional charts. Finally, the last part gives an evaluation of the sonification techniques implemented in both applications, based on an on-line questionnaire.

Chapter 2 offers a theoretical overview of the concepts of sound and sonification. Both the advantages and limitations of sonification are described as well as some examples of existing implementations.

Chapter 3 focuses on time-dependent data series in a sonification context. Preprocessing techniques are described to filter out errors and noise from the data set. The chapter explains how undesired or unnecessary values are removed and how the remaining ones are prepared for sonification.

Chapter 4 describes the most widely used technique for the sonification of time-dependent data, the pitch-based mapping. From this technique, several other features are introduced and explained to enhance the sonification.

At this point of the dissertation, the reader has acquired a theoretical knowledge of the sonification as well as the available techniques to translate time-dependent data into sound. However, before putting into practice the theory exposed in previous chapters, it is important to consider some crucial aspects of sound implementation. Thus, chapter 5 provides some guidelines when developing a sonification application, in particular when using the Java Sound API.

Chapter 6 and 7 describe respectively the two Java applications that we have developed, SoundChart and SoundChart3D. Each chapter explains how the whole process of sonification was implemented, from the preprocessing of data to the sonification techniques themselves. The SoundChart application provides tools for the sonification of two-dimensional time-dependent data while SoundChart3D gives a sonic representation of three-dimensional data.

As sonification is not necessarily straightforward or intuitive, an experimentation of time-dependent data sonification, based on both applications, is given in chapter 8. An Internet Web site, including two questionnaires, was created to get an evaluation of the sonification techniques implemented in SoundChart and SoundChart3D. The chapter describes each question, comments the results and draws conclusions about the efficiency and the practical use of time-dependent data sonification.

Conclusions of the present dissertation are to be found in Chapter 9.

Chapter 2

Sound and Sonification

2.1 Introduction

The purpose of this chapter is to get the reader used to the concepts of sound and sonification. First of all, we briefly introduce the definition of sound and its main attributes. Secondly, we explain the notion of sonification and we give different types of sonification techniques. The main advantages and limitations of sonification are then described as well as some examples of existing implementations.

2.2 Basic overview of sound

2.2.1 Definition of sound

A sound is the sensation of pressure variations in the air caused by a vibrating source [MR95], like a violin, an automobile horn, or a barking dog. They all produce vibrations that disturb the air in such a way that sound waves are produced. These waves travel out in all directions, expanding from the source of the sound. If the waves happen to reach someone's ear, they set up vibrations that are perceived as sound [Com98].

The pattern of variations in the air over time determines the waveform of a sound. Figure 2.1 shows two sample, sinusoidal waveforms. In both cases, the Y-axis represents pressure, and the X-axis represents time. The measure of pressure is the amplitude of the sound. If the waveform is composed of a repeating pattern, as these waveforms are, they are said to be periodic, and the smallest repeating pattern is a cycle [MR95]. The frequency of a waveform is the number of repeating cycles of change in air pressure that occur in one unit of time, usually a second. It is expressed in Hertz (Hz) or cycles per second (cps). Complex sounds are made up of many pure

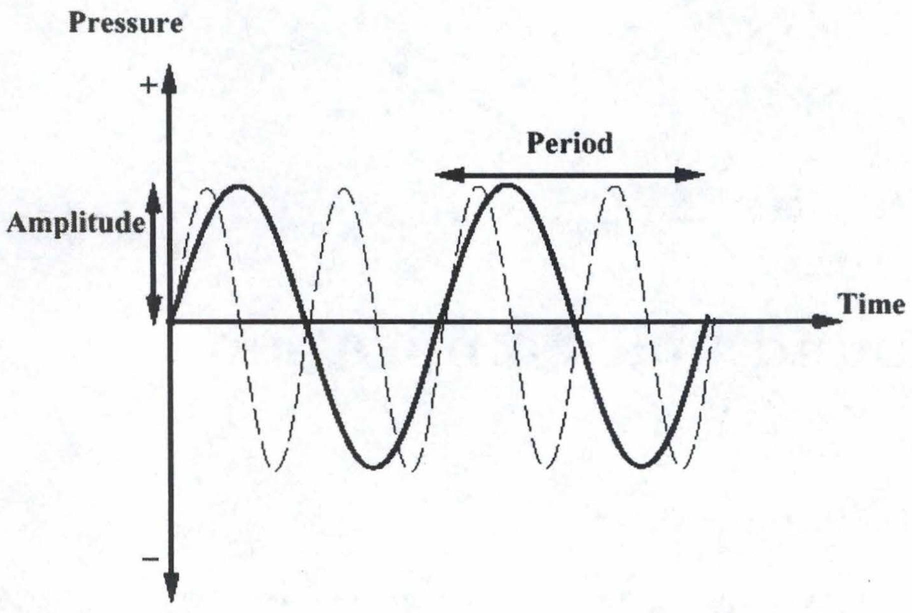


Figure 2.1: Two periodic waveforms

tones of different frequencies. The waveform depicted by the dashed line has a frequency twice that of the other. The range of human hearing varies from individual to individual, but normally falls between 20 Hz and 20,000 Hz. For convenience, this range is divided into three rough bands: high frequencies (between about 5 kHz and 20 kHz), mid frequencies (between about 200 Hz and 5 kHz) and low frequencies (between about 20 Hz and 200 Hz).

2.2.2 Sound attributes

Although there are only a relatively small number of sound characteristics, they can be manipulated to produce a rich set of sounds. Typically, four main characteristics are distinguished: pitch, loudness, timbre and location.

The perceived pitch of a sound is just the ear's response to frequency and for most practical purposes the pitch is just the frequency. A high pitch sound corresponds to a high frequency and a low pitch sound corresponds to a low frequency. Amazingly, many people, especially those who have been musically trained, are capable of detecting a difference in frequency between two separate sounds which is as little as 2 Hz. When two sounds with a frequency difference greater than 7 Hz are played simultaneously, most people are capable of detecting the presence of a complex wave pattern resulting from the interference and superposition of the two sound waves [Hen01].

The loudness and intensity of a sound refers to the amplitude of the sound and is specified in decibels (dB). While the intensity of a sound is a very objective quantity which can be measured with sensitive instrumentation, the loudness of a sound is more of a subjective response which will vary with a number of factors. The same sound will not be perceived to have the same loudness to all individuals. Despite the distinction between intensity and loudness, it is safe to state that the more intense sounds will be perceived to be the loudest sounds [Hen01].

Timbre or quality is equivalent to the waveform of a sound. It is timbre that determines whether one hears a trumpet or a piano, even if the pitch is the same [MR95]. Timbre is then a general term for the distinguishable characteristics of a tone.

Location (also called localization or spatialization) is the perception of a sound source's placement. Indeed, the perception of a sound is greatly affected by the direction and position of the corresponding sound source, as well as by the environmental acoustics.

2.3 Data sonification

2.3.1 Sonification: definition and types

Sonification can be defined as an auditory representation of data [Anr99]. It is often used in combination with visualization techniques in order to reveal data properties not easily rendered by visual graphics. It is also a useful technique for presenting information to blind or visually impaired individuals [ARV97], and for displaying data to users whose visual attention must be devoted elsewhere [FBT96].

Data sonification finds application in many fields of research. According to their proper needs and objectives, these fields require a specific sonic integration into their projects. Therefore several sonification methods have been developed. Commonly six types of sonification techniques are distinguished [Her99].

Alarm signals

The first applications of acoustic signals in human-computer interfaces have been alarm signals. The signal reaches the user even if (s)he is occupied with other things. The simplest data sonification is done in a regular PC speaker: the BIOS announces the state of the hardware acoustically. Alarm signals are mainly used in high-tech domains where the user is confronted with a huge amount of visual information (e.g. in airplane cockpits and nuclear plants). Alarms are usually triggered to indicate that an abnormal program behaviour or an error has occurred. However, they are rather poor for complete presentation of complex data.

Auralization

The auralization method translates the data itself to amplitude values of the waveform. It is applicable if the data itself is a time series, e.g. values from a dynamic system like seismic data. This sonification and related forms are quite useful to monitor large data sets in strongly compressed time.

Earcons

Earcons have been proposed by Blattner et al. Earcons are simple tonal combinations or arbitrary acoustic patterns whose meaning must be learned by the user, and which can be combined to build non-verbal messages of a higher complexity and meaning. There is no intuitive link between the sound and what it represents, as earcons are mainly abstract sonic events. The disadvantage is the learning effort and the limited range of application fields [BWE93].

Auditory icons

Auditory icons use recordings of real-world sounds to signal events. These sounds are used in a metaphorical sense, making the learning effort less difficult. Indeed, real-world sounds, if selected correctly, can carry lots of intrinsic meaning because of our experience with them. For example, filling a bottle with water produces a commonly known sound evolution, which might be applied to a "state of download - meter" sonification. However, real-world sounds can be confusing as they don't mean the same thing to all people.

Parameter mapping

Parameter mapping is a kind of sonic scatter plot. For each data record an acoustic event is generated as data attributes are mapped to sound parameters such as pitch, loudness, timbre, time stamp, etc. Parameter mapping is the richest method to present high dimensional data, but the dimensionality is limited to the number of sound parameters. Furthermore, there is no unique way of mapping between attributes and parameters. The listener therefore requires some learning time to get acquainted to a chosen mapping.

Model-based sonification

Model-based sonification partly overcomes the drawbacks of parameter mapping. It uses the viewpoint that sound is better characterized by its sound source and the sound generating processes than by isolated attributes of the sound signal like pitch or envelope shape [HHR01]. The idea is to take the way that we use sound in our natural environment as a model for the usage in data sonification. Our auditory system is well-optimized for the interpretation of sounds occurring in our physical environment. Model-based sonification tries to build a sonification interface similar to the physical model, where objects only produce sound when they are excited by the user. Thus sound is an important feedback which communicates properties of the material. Before building such a sonification model, one needs to identify dynamical elements of the data set, provide a recipe for the dynamics, and specify the types of allowable interactions. The main advantages of this approach are as follows [HHR01]:

- Sonification models need only a few parameters (given in the model definition) whose meaning is intuitively understood and that remain the same independent of the data under consideration.
- Properly designed models can be applied to arbitrary high-dimensional data. As the data is implicitly encoded into sound there is no need for prior feature selection or dimension reduction.
- Interaction with a sounding object by excitation is something human users are familiar with and find more natural, so they will find the interface more pleasing.

2.3.2 Advantages

Hearing being very different from vision, the auditory channel has many advantages over the visual channel. Here are the major ones distinguished by G. Kramer [Kra94].

The first advantage is that sound can be integrated without interfering with the visual display. Therefore, whatever the amount of data displayed, the user does not become overloaded by the addition of sound. The workload can even be reduced as sharing between two sensory modalities (e.g. hearing and sight) may be superior to sharing within a single modality [NF00].

Unlike vision, hearing is non-directional. One doesn't need to have his head or body pointed toward something in order to hear it. This means that sound has an advantage for signals that appear infrequently: you don't have to be glued to a display, waiting for the thing to appear. This way, sound has a better response time than visual representations and allows rapid detecting and alerting when monitoring high stress situations [Anr99].

Another advantage is the facility to notice variations in a sound. When looking right at the screen, the user could miss something, especially if it appears only briefly. It's much more difficult to miss a change in a sound. Human hearing is particularly sensitive to temporal changes in sounds. We notice very fine alterations to continuous sounds, and we can pick out the changes from a complex array of information.

Adding sound to visual representations offers additional dimensionality. Indeed, using sound provides up to seven dimensions by taking advantage of the available sound attributes [CBL95]. Therefore, when the same complex information is presented in a visually or auditory form, the latter requires significantly less effort to interpret.

In addition, some characteristics of visual and auditory displays are presented in table 2.1.

The message length in auditory displays is usually shorter than in visual displays. While graphical displays can be very visually loaded, data sonification has to be quite short in order to be effective and easily interpreted. Earcons for example consist only of a couple of notes to ease the user's learning process.

Although data sonification can be very useful, sound often conveys less information than graphical data. Consequently, the info rate of auditory displays is limited.

As already stated, the non-directionality of sound allows an immediate response of the user, even if he is not looking at the signal's source (directed attention is not required). Moreover, changes in sound are more quickly detected than visual changes, as our visual sense has a rather small area of high focus. For example, while typing a text with a wordprocessor, the user doesn't notice the time display being updated every minute at the bottom right corner of the screen. This wouldn't be the case if a sound was played every minute (although it would be quite annoying).

Placing a sound source at a particular three-dimensional location as heard through stereo headphones is difficult and the perception of elevation of a sound source is relatively poor. However, the one-dimensional mapping offered by stereo balance can augment visualizations by providing locational cues [MR95].

2.3.3 Limitations

While there are many advantages to using sound for data display, there are also disadvantages that must not be ignored [Wil96].

Characteristic	Auditory	Visual
message length	short	long
info rate	low	high
response	immediate	later
attention-getting	good	limited
directed attention	not required	required
localization	limited	good

Table 2.1: Auditory vs. visual displays

The most important disadvantage is that a human ear cannot ignore auditory stimuli. Unlike the human eye, the ear cannot ignore the stimuli by simply moving its focus or closing its eyelid. This can cause severe problems as users are easily disturbed by sounds coming from other users [Anr99]. Consequently, auditory displays are frequently disabled by users due to an increase in frustration and subjective workload.

Sonifications that produce chaotic and unmusical sounds, while they may still be useful, can be hard to listen to. Auditory fatigue may reduce the ability of a listener to recognize the information conveyed by a sonification.

Just as in graphical displays one object can mask another, so can one sound mask another. A predominating timbre, or volume differences between data streams, may render a portion of the sonification undetectable. Moreover, some sound properties may make others undetectable. For example, if the duration of a note is too short, we may not be able to perceive its pitch.

2.3.4 Examples

As already stated, sonification is a way to provide information to visually impaired individuals, using sound instead of visualization. But sonification can also be used to extend human-computer interaction, using sound in addition to visual display. This section briefly describes some examples where sound replaces or improves visual representations.

Soundtrack

The Soundtrack project is a word processor with an auditory interface proposed by Edwards in 1989 [Edw89]. This kind of interface is useful for visually impaired persons, but also for sighted persons when the screen becomes visually overloaded. The Soundtrack interface uses both speech and non-speech sounds. The screen is divided into areas corresponding to different menus. When the mouse pointer goes over one of these areas, a characteristic sound is played, notifying the user of the pointer location. If the user wants more information, he can click on the area and the title is dictated by a speech synthesizer.

The SonicFinder

The SonicFinder is a sonic interface developed by Bill Gaver in 1989 [Gav89]. This interface adds auditory cues to the graphical Finder environment on the Macintosh. It uses sound in a way that reflects how it is used in the everyday world. Thus, dragging a file icon with the mouse sounds like dragging something along the floor, and dropping an icon into the trashcan sounds like dropping a large object into a metal trashcan. Moreover, thanks to SonicFinder, the user is able to determine the type of an object (e.g. an application, disk or file folder) and its size, as small objects have high-pitched sounds and large ones are low-pitched.

WebMelody

The WebMelody project is a novel technique developed by italian researchers to monitor the behaviour of a World Wide Web server by associating sounds to events describing the server activity [BCS$^+$00]. This sonification technique is an efficient solution to allow on-line monitoring of large amount of log data that are available. The webmaster is able to monitor the correct functioning of the WWW server in real time without interfering with other activities. Indeed, the webmaster can perceive the server behaviour as "background music" as he is executing other tasks at the same time. Moreover, this real-time information can quickly redirect the focus of the webmaster to the problem whenever needed so that he can take appropriate actions.

CAITLIN

The CAITLIN project, developed by Paul Vickers, aims to determine whether musical auralizations of Pascal programs can usefully assist the novice programmer with debugging [Vic99, VA96]. CAITLIN musically auralizes programs written in Turbo Pascal and presents the user with an integrated environment for carrying out auralization, compilation and running of programs. Auralization is done at the construct level. A WHILE loop is auralized in one way and REPEAT, FOR, CASE, IF...THEN...ELSE and WITH constructs in others. The user may select, for each construct, the nature of the auralization to be applied. Music helps novice programmers with locating errors in their code but might also be used to assist visually impaired programmers.

2.4 Conclusion

In conclusion, data sonification can significantly increase the bandwidth of human-computer interfaces. Sound properties, as described in this chapter, offer interesting ways of communicating and interacting with the computer. Moreover, by explaining some techniques of sonification, some examples of systems implementing them, as well as some of its advantages and disadvantages, it has been shown that sonification remains an interesting field of research. Making our listening capabilities usable can extend visual displays and ultimately even replace them. Sound, alone or in combination with visual imaging techniques, offers a powerful means of transmitting information.

Chapter 3

Preprocessing time-dependent data

3.1 Introduction

Before sonification can be applied, the raw data set must be treated in order to remove undesired or unnecessary values, and to prepare the remaining ones for sonification. This chapter covers the preprocessing techniques that allow time-dependent data to be sonified. After a small introduction to the concepts of time series, we present the problems caused by extreme values and the ways to detect them, as well as the main smoothing methods used to filter out errors and noise from the data set.

3.2 Time-dependent data

3.2.1 Time series

Time-dependent data is a time series, i.e. a sequence of observations which are ordered in time. Each time stamp is associated with a data value. The data values are time-dependent, while time is the independent variable. Time series are best displayed in a scatter plot. The series value is plotted on the vertical axis and time on the horizontal axis. There are two kinds of time series [EM97]:

Continuous time series There is an observation at every instant of time, e.g. lie detectors and electrocardiograms.

Discrete time series There is an observation at (usually regularly) spaced intervals, e.g. weekly share prices.

They will not be distinguished in this dissertation, as they are quite similar and share the same characteristics. Moreover, the preprocessing and sonification techniques further discussed can be applied to both of them.

3.2.2 Components

There are two main goals of time series analysis: (a) identifying the nature of the phenomenon represented by the sequence of observations, and (b) predicting future values of the time series variable (forecasting) [Sta01]. It is assumed that the data consist of a systematic pattern (usually a set of identifiable components) and random noise (error) which usually makes the pattern difficult to identify. Smoothing and filtering techniques are then used in order to make the pattern more perceptible. Both of the above-mentioned goals require that this pattern is identified, which is achieved by analysing the components that make up the pattern. Most time series patterns can be described in terms of two basic components: trend and seasonality.

Trend

Trend is a long term movement in a time series. It is the underlying direction (an upward or downward tendency) and rate of change in a time series, when allowance has been made for the other components. A simple way of detecting trend in seasonal data is to take averages over a certain period. If these averages change with time there is evidence of a trend in the series [EM97].

There are no proven automatic techniques to identify trend components in the time series data. However, as long as the trend is monotonous (consistently increasing or decreasing), that part of data analysis is typically not very difficult. If the time series data contain considerable error, then the first step in the process of trend identification is smoothing [Sta01].

Seasonality

Seasonality may have a formally similar nature to the trend component, but it repeats itself in systematic intervals over time. Usually it describes any regular fluctuations with a period of less than one year (e.g. unemployment figures and average daily rainfall).

It is formally defined as correlational dependency of order k between each i^{th} element of the series and the $(i-k)^{th}$ element. It is measured by autocorrelation, i.e. a correlation between the two terms. If the measurement error is not too large, seasonality can be visually identified in the series as a pattern that repeats every k elements [Sta01].

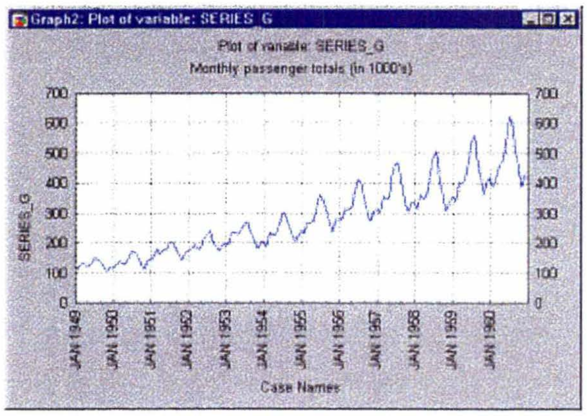


Figure 3.1: Example of multiplicative seasonality

Multiplicative seasonality

The multiplicative seasonality pattern indicates that the relative amplitude of seasonal changes is constant over time, and thus related to the trend. A classic example of multiplicative seasonality is illustrated in figure 3.1. It represents monthly international airline passenger totals (measured in thousands) in twelve consecutive years from 1949 to 1960. The trend is almost linear, indicating that the airline industry enjoyed a steady growth over the years (approximately 4 times more passengers travelled in 1960 than in 1949). At the same time, the seasonal component shows that more people travel during holidays then during any other time of the year. Moreover, the multiplicative seasonality component becomes quite evident, as the amplitude of the seasonal changes increases with the overall trend.

3.3 Extreme values

3.3.1 Outliers and mistakes

When analysing time-dependent data, one may find a particular value that is unusually large or small relative to the others. Such a value is called an extreme value or an outlier, terms that are usually not defined rigorously [Une00]. What happens to outliers depends how the outlier achieved its unusual value. If it is due to a mistake, like an experimental error or an error while entering the data, it should be discarded, since including an erroneous value in the analyses will give invalid results. This is especially true for sonification applications, where the slightest oddity in the signal will be noticed. If the unusual value is due to chance it should be kept in the analysis, as it is a correct value and comes from the same population as the others.

3.3.2 Detecting outliers

By their informal definition, outliers are characterized by their distance from other observations. But how distant does an observation need to become before it is classed as an outlier? Below are described three of the most widely used statistical methods for detecting extreme values [Une00].

Z-score method

In a z-score test, the mean and standard deviation of the entire data set are used to obtain a z-score for each data point, according to following formula:

$$z_i = \frac{(x_i - \overline{x})}{s}.$$

Using Chebyshev's theorem, it has been stated that an observation with a z-score greater than three should be labelled as an outlier.

Chebyshev's theorem *For any positive constant k, the probability that a random variable will take on a value within k standard deviations of the mean is at least* $1 - \frac{1}{k^2}$.

Although this method is extremely simple and easy to implement, it is not a reliable way of labelling outliers since both the mean and standard deviation are effected by the outliers.

Box plot method

The box plot method uses a graphical representation of the data set to label outliers. The median and the lower and upper quartiles of the data set are needed in order to build the plot. The median is the middle value in the ordered data set, which creates two halves. Each half can be further subdivided to give quarters, that define the lower and upper quartiles. Here are the steps to follow in constructing a box plot.

1. Calculate the median M, lower and upper quartiles, Q_L and Q_U, and the interquartile range, $IQR = Q_U - Q_L$, for the data set.

2. Construct a box with Q_L and Q_U located at the lower corners. The base width will then be equal to IQR. Draw a vertical line inside the box to locate the median M.

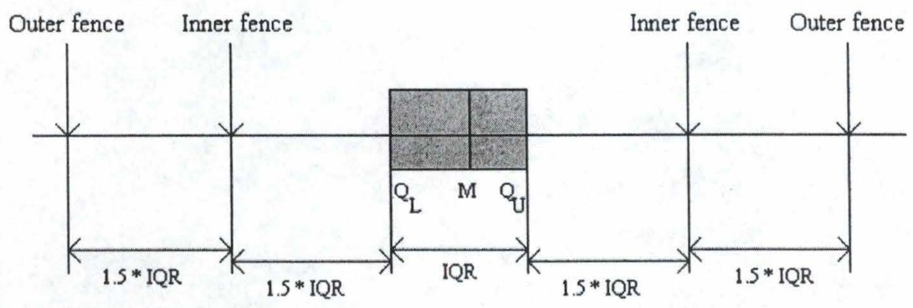


Figure 3.2: The box plot method

3. Construct two sets of limits on the box plot: inner fences are located a distance of $1.5 * IQR$ below Q_L and above Q_U; outer fences are located a distance of $3 * IQR$ below Q_L and above Q_U (see figure 3.2).

Using this box plot, it becomes very easy to locate possible outliers. Observations that fall between the inner and outer fences are suspect outliers. Observations that fall outside the outer fences are highly suspect outliers.

Grubbs' method

Grubbs' method is composed of three different tests, each one for detecting a particular type of outlier. While the first test looks for single extreme values (figure 3.3a), the other tests look for pairs of extreme values, i.e. outliers that are masking each other (see figure 3.3b and 3.3c) [Bur98].

Before the three tests can be applied, the data needs to be arranged in ascending order. The test values are then computed using the following formulae:

$$G_1 = \frac{|\overline{x} - x_i|}{s} \qquad G_2 = \frac{x_n - x_1}{s} \qquad G_3 = 1 - \left(\frac{(n-3)\ s_{n-2}^2}{(n-1)\ s^2} \right)$$

where x_i is the suspected single outlier i.e. the value furthest away from the mean, x_n and x_1 are the most extreme values (since the data is ordered), and s_{n-2} is the standard deviation for the data set excluding the suspected pair of outlier values i.e. the pair of values furthest away from the mean.

If the test values (G_1, G_2, G_3) are greater than the critical value obtained from the table at figure 3.4, then the extreme value(s) are unlikely to have occurred by chance at the stated confidence level [Bur98].

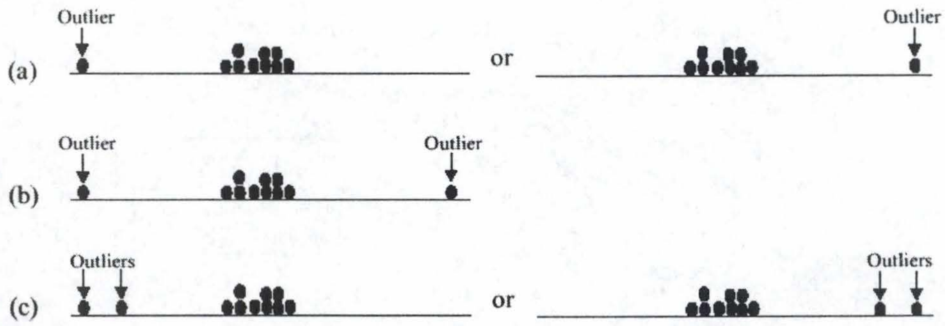


Figure 3.3: Three outlier situations

	95% confidence level			99% confidence level		
n	G_1	G_2	G_3	G_1	G_2	G_3
3	1·153	2·00	---	1·155	2·00	---
4	1·463	2·43	0·9992	1·492	2·44	1·0000
5	1·672	2·75	0·9817	1·749	2·80	0·9965
6	1·822	3·01	0·9436	1·944	3·10	0·9814
7	1·938	3·22	0·8980	2·097	3·34	0·9560
8	2·032	3·40	0·8522	2·221	3·54	0·9250
9	2·110	3·55	0·8091	2·323	3·72	0·8918
10	2·176	3·68	0·7695	2·410	3·88	0·8586
12	2·285	3·91	0·7004	2·550	4·13	0·7957
13	2·331	4·00	0·6705	2·607	4·24	0·7667
15	2·409	4·17	0·6182	2·705	4·43	0·7141
20	2·557	4·49	0·5196	2·884	4·79	0·6091
25	2·663	4·73	0·4505	3·009	5·03	0·5320
30	2·745	4·89	0·3992	3·103	5·19	0·4732
35	2·811	5·026	0·3595	3·178	5·326	0·4270
40	2·866	5·150	0·3276	3·240	5·450	0·3896
50	2·956	5·350	0·2797	3·336	5·650	0·3328
60	3·025	5·500	0·2450	3·411	5·800	0·2914
70	3·082	5·638	0·2187	3·471	5·938	0·2599
80	3·130	5·730	0·1979	3·521	6·030	0·2350
90	3·171	5·820	0·1810	3·563	6·120	0·2147
100	3·207	5·900	0·1671	3·600	6·200	0·1980
110	3·239	5·968	0·1553	3·632	6·268	0·1838
120	3·267	6·030	0·1452	3·662	6·330	0·1716
130	3·294	6·086	0·1364	3·688	6·386	0·1611
140	3·318	6·137	0·1288	3·712	6·437	0·1519

Figure 3.4: Grubbs' critical value table

3.3.3 Notifying outliers

If the extreme value is due to chance, it should be kept and added to the sonification. As this value represents a rare event, it could even be sonified as such, e.g. with an alarm signal or another unmistakable sound. Unfortunately, using this kind of signals often prevents the user from hearing the outlier's value, because at that time the signal is the center of attention and keeps the ear away from other sonic information.

If the outlier is due to a mistake, it could still be added to the sonification, since it may be worth knowing that the time series contains an error.

In both cases, even if the unusual value is mapped like a normal one, the resulting sound will be out of the ordinary and noticed by the user.

3.4 Smoothing

3.4.1 Objectives

Time-dependent data often contains random short term fluctuations. Since the main goal of sonification is to give a clearer view of the long term behaviour of a series, these random variations need to be filtered out. Smoothing techniques are used to reduce these irregularities. In some time series, seasonal variation is so strong it obscures any trends or cycles which are very important for the understanding of the process being observed. Smoothing can remove seasonality and makes long term fluctuations in the series stand out more clearly [EM97].

3.4.2 Smoothing techniques

Moving averages

Moving averages range among the most popular smoothing techniques for the preprocessing of time series. A moving average is a form of average which has been adjusted to allow for seasonal or cyclical components of a time series [EM97]. It replaces each element of the series by either the simple or weighted average of p surrounding elements, where p is the length of the smoothing window. For example, the formula of the simple moving average is

$$\mu_n = \frac{1}{p}\sum_{i=0}^{p-1} x_{n-i},$$

where μ_n represents the new value being computed and p represents the length of the smoothing window.

Running medians

Running medians smoothing is a technique analogous to that used for moving averages. The running median is calculated by finding the median of all the values in a neighbourhood of μ_n. Like a moving average, the running medians makes a trend clearer by reducing the effects of other fluctuations. However, it is more robust than a moving average, because it is least likely to be affected by erroneous data points.

Differencing

Differencing is a popular and effective method for removing trends from time series. The first difference of a time series μ_t, written $D\mu_t$, is defined by the transformation $D\mu_t = \mu_t - \mu_{t-1}$. Higher-order differences are defined by repeated application. Thus the second difference $D^2\mu_t$ is defined by $D^2\mu_t = D(D\mu_t) = D\mu_t - D\mu_{t-1} = \mu_t - 2\mu_{t-1} + \mu_{t-2}$. In general, the effect of a polynomial of degree k can be reduced to a constant by differencing k times [Wan00].

3.5 Conclusion

When the preprocessing step is over, the extreme values have been identified and taken care of, while the smoothing methods have removed the short term fluctuations that would obscure the sonification. The time-dependent data values are now ready to be sonified.

Chapter 4

Pitch-based mapping

4.1 Introduction

As we have seen, parameter mapping is the most widely used sonification technique for time-dependent data. Indeed, the possibility to freely combine and mix up sound attributes makes it the richest method for the presentation of high dimensional data. This chapter describes the main mapping technique used for parameter mapping, known as pitch-based mapping. This type of mapping is very easy to produce, and is often used as the starting point for complex sonification techniques.

4.2 Mapping methods

Pitch-based mapping represents numerical data by tones at specific frequencies. These frequencies typically have a lower and upper limit, as the human ear has trouble perceiving too low frequencies, and can be irritated or even damaged by too high frequencies. While it seems quite reasonable to map big values to high frequencies and small values to low frequencies, there are several ways to convert data values to a given frequency range. The two common mapping methods are linear mapping and chromatic scale mapping [Sah99].

4.2.1 Linear mapping

Linear pitch-based mapping is quite straightforward. Data values have a linear correspondence to frequency. Thus, if the variable y_1 is twice the value of y_2, the frequency of y_1 will be twice the frequency of y_2 (when taking into account their respective ranges). Linearly mapping a value y to a frequency f is accomplished by the relation

$$\frac{y - y_{min}}{y_{max} - y_{min}} = \frac{f - f_{min}}{f_{max} - f_{min}},$$

where y_{max} and y_{min} are the maximum and minimum data values to be mapped, and f_{max} and f_{min} span the desired frequency range. Thus, the resulting linearly mapped frequency is given by

$$f = f_{min} + (f_{max} - f_{min})\frac{y - y_{min}}{y_{max} - y_{min}}.$$

4.2.2 Chromatic scale mapping

Chromatic scale mapping, also called logarithmic mapping, maps the data points to actual notes of the chromatic scale. Thus the resulting pitch values share a relationship found in musical instruments. For more information concerning the chromatic scale, we refer to [PB01]. The formula for converting a value y to a frequency f is

$$f = f_{min}\left(\frac{f_{max}}{f_{min}}\right)^{\frac{y-y_{min}}{y_{max}-y_{min}}}.$$

Figure 4.1 shows the mapping results of an arbitrary graph using both mapping methods. The graph formed by the linearly mapped values has exactly the same shape as the original graph, since the linearly mapped values and the original graph's values are proportionally the same. The chromatic scale graph is not proportional to the original one. Indeed, the frequencies corresponding to the top and bottom values are quite similar, but the middle ones are very distant from each other. In practice, it is hard to tell which one is better, but the chromatic scale mapping usually sounds a little better.

4.3 From mapping to sound

Pitch-based mapping transforms the initial data values to frequency values, which are then used to create notes and eventually produce sound. There are many ways to create and assemble these notes, each one resulting in a unique sound signal. Besides the desired frequency range, other parameters can greatly affect the sonification.

Timbre Whether using a single instrument or more, changing the timbre obviously influences the signal, and thus the possible interpretation, to a great extend.

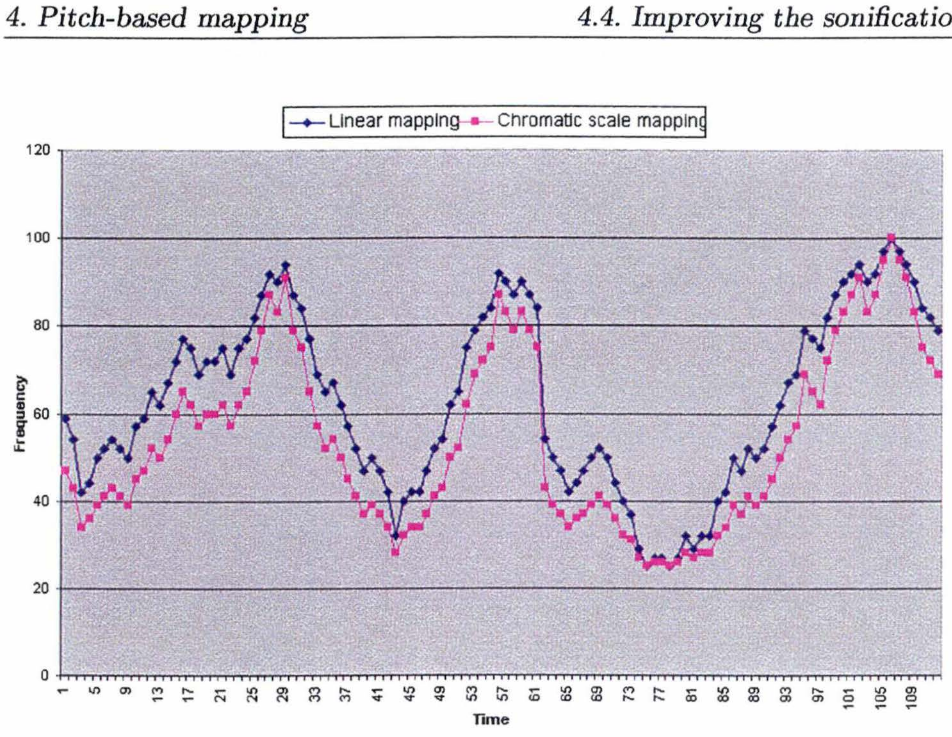


Figure 4.1: Linear vs. chromatic scale mapping

Time stamp Every sonification process needs some sort of timing method to assemble the notes in a realistic and efficient way. The simplest timing method is to choose a fixed interval as the time duration between consecutive notes.

Loudness The loudness or amplitude of a note is more of a subjective factor and has a limited influence on the signal.

4.4 Improving the sonification

Pitch-based mapping, using a linear or chromatic scale method, is typically the initial mapping technique for time-dependent data. While the created signal sounds already quite convincing, several features can be added to improve the sonification.

4.4.1 Slope indicator

Using only the basic pitch-based mapping method, it is difficult to tell when a graph is linearly increasing (e.g. $Y = X$), and when there is some curvature to the graph (e.g. $Y = X^2$). For this reason, sound can be added to alert the listener to the slope and curvature of the graph [Sah98]. The first derivative of the initial data represents the slope and is defined by

$$\frac{\mathrm{d}Y}{\mathrm{d}X}.$$

Generally a series of drum beats are used to indicate this slope value (the greater the slope, the more frequent the beat). In addition, it is possible to indicate the curvature of the graph by varying the pitch of the drum beat. The curvature represents the change in the slope and is defined by

$$\frac{\mathrm{d}^2Y}{\mathrm{d}X^2}.$$

The drum beat should have at least three different pitch levels, as the graph can be bowl shaped (curvature > 0), hill shaped (curvature < 0) or linear (curvature $= 0$).

When using slope indicators, the importance of smoothing becomes more evident. Indeed, a typical time-dependent data graph contains a lot of small fluctuations, which would cause the slope to be highly irregular. By using smoothing methods, the slope variations are minimalized, resulting in a better sonification.

4.4.2 Panning

Panning is a form of sound localization, and relies on intensity variations between adjacent speakers to achieve the desired spatial localization. When the sonification involves multiple different signals, panning can help to differentiate them more easily by playing every signal at a different spatial location.

Stereo panning is the most common type of panning and involves a single pair of speakers. By varying the level of otherwise identical signals played from this pair of speakers, stereo panning can alter the perceived laterality of an audio source [SSC99]. Besides being used for sonifying multiple signals, it is typically used as a duration indicator of the signal. The signal starts on the left speaker, and gradually moves to the right one. Thus, at the end of the signal only the right speaker is playing. This mechanism enables the user to hear the progression (and thus the duration) of the signal. For example, when both speakers are playing at the same intensity, the user knows that half of the signal is over, and that the other half is still to come.

A crucial problem of stereo mapping is the location of the listener with respect to the speakers. While the best location is equidistant from the speakers positioned symmetrically in front of the listener, the perceived direction changes quite easily as the user starts to move around. For example, a duration indicator would be useless if the listener is sitting too far away from the speakers.

The ability to generate an accurate spatial simulation using loudspeakers increases dramatically as the number of speakers used in the display increases. With an infinite number of speakers around the listener, one would simply play the desired signal from the speaker at the desired location of the source to achieve a perfect reproduction. Therefore, panning between multiple pairs of speakers (for instance, speakers arrayed in front of and behind the listener) is often used to improve spatial simulations using loudspeakers [SSC99].

4.4.3 Extrema values

Although extrema values can be outliers, and outliers can be extrema values (they often are), outliers (extreme values) and extrema values are two different concepts. Extrema values are maximum and minimum values, which can be absolute or local. If the time-dependent data begins at time t_0 and ends at time t_n, an extrema value at time c has the following properties:

Absolute (global) maximum $y_c \geq y_i$, with $t_0 \leq i \leq t_n$

Absolute (global) minimum $y_c \leq y_i$, with $t_0 \leq i \leq t_n$

Local (relative) maximum $y_c \geq y_i$, when i is near c, with $t_0 \leq i \leq t_n$

Local (relative) minimum $y_c \leq y_i$, when i is near c, with $t_0 \leq i \leq t_n$
(Near c implies all i in a chosen interval containing c)

There are several methods for sonifying extrema values. One could represent each type of extrema value by an distinct sound, and simply play this sound when the signal reaches a maximum or minimum value. But this sonification method prevents the user from hearing the extrema's value. Moreover, if for example two local maximum values are detected, it is impossible to hear whether they have the same value or not, since they are represented by the exact same sound. A possible solution would be to vary the sound's frequency in function of the extrema's value, thus allowing to differentiate two different values.

Another sonification method makes use of a speech synthesizer, which converts numerical values to their spoken sonic counterpart. These numerical values can be either the time values at which an extrema point occurred (on X-axis), or the corresponding data values (on Y-axis). This method is also useful if the user is interested in the actual numeric values of data elements around a maximum or minimum point.

4.5 Conclusion

Parameter mapping is a rich and relatively unexplored technique for representing single and multi-dimensional data. The ability to combine more than a dozen sound parameters in lots of different ways leads to an abundance of possible sonifications. While this seems like an obvious advantage, it can cause some difficulties, as the plurality of choices makes it very tricky to find the best solution to a sonification problem. Moreover, the lack of sonification standards and research on the subject prevents one from easily identifying the optimal solution, and forces him to carry out some laborious experiments.

Chapter 5

Sound implementation

5.1 Introduction

When implementing a sonification application, it is important to select the best sound format for the demanded task. Depending on the type of sonification, several parameters can have an influence on the chosen format. For example, one can choose to sonify data in real-time or not. This crucial choice will have a major impact on the performance and possibilities of the chosen sonification.

This chapter starts with an overview of the available sound formats. It then introduces the critical parameters that assist the designer in selecting the best format for a particular sonification. The last section is much more concrete as it discusses the implementation of MIDI using Java. It particularly focuses on some implementation issues that could arise when using the Java Sound API.

5.2 Available sound formats

5.2.1 Sampled sounds

Sampled sounds are the most widely used of all the available sound types. Typically they have been digitally recorded by a sampling technique, but they can also be computed (possibly at run time). The sampled sound data consist of a bit by bit recording of a sound, by sampling and storing the intensity of the soundwave at frequent time intervals. The sampling rate determines the sound quality. For example, audio CDs are recorded using a 44 kHz sampling rate, which means that the soundwave is sampled over $40,000$ times a second, producing very high quality sound. The main advantage of sampled sounds is that they can describe a wide variety of sounds, such as speech or special sound effects.

Generally the recordings are stored in WAV files, so named for their file extension. Because the soundwaves need to be sampled frequently to obtain good quality audio, the sampling procedure generates a great deal of data. Thus the resulting files are usually very large, especially when the recorded sound is quite complex. About 16K to 48K of storage space is needed for every second of sound, even if there is "no" sound. The amount of storage space needed depends on the sampling rate, and whether the sample is recorded in stereo or not [Sch97]. Often complex compression techniques are used to reduce the file size.

WAV files are digitally stored sound samples. When playing them on a sound card, the sound reproduced is deemed to be "authentic" as long as the sound card can reproduce the sample in the original resolution, i.e. at the same sampling rate. The sample sound files contain the complete sound information, so they are not depending on the synthesizer ability of the sound card [Sch97].

5.2.2 MIDI

MIDI stands for Musical Instrument Digital Interface. It is a standard for specifying how a computer should play digital instruments. A computer plays sound via a synthesizer on the sound card. Simply stated, MIDI information tells a synthesizer when to start and stop playing a specific note [Lip89]. Moreover, the MIDI standard contains instructions for specifying the parameters of a note, such as volume, instrument, etc.

MIDI files do not transmit sound. They transmit instructions that tell the synthesizer what notes to play, how to play them and at what time. The synthesizer uses previously stored sound sequences to produce the requested notes. MIDI files contain data that are merely player instructions for the synthesizer. Therefore the sound quality depends totally on your sound card's synthesizer capability. While with MIDI instructions the timing and volume is strictly the same, the sound quality depends entirely on the tonal quality of the synthesizer sequences that are used to play them [Sch97].

In order to make MIDI files more compatible, the General MIDI specification was created. It primarily defines a standard set of 128 instruments (called programs), and gives them an associated number. For example, it was decided that program number 1 on all synthesizers should be the sound of an Acoustic Grand Piano. In this way, no matter what MIDI synthesizer is used, switching to program number 1 always plays some sort of Acoustic Grand Piano. Because General MIDI does not specify how each instrument should be synthesized, the quality of the sound is still dictated by that of the hardware.

The great advantage of MIDI over sampled sound is that it only requires a fraction of the information. Because MIDI files only contain instructions on how to generate sound, and not the sound itself, the information is very compact. This causes MIDI files to have a very small size. In fact, normal MIDI files are always shorter than any sampled sound file.

MIDI messages

The basis for MIDI communication is the byte. Each MIDI instruction has a specific byte sequence. The first byte is the status byte, which tells the MIDI device what function to perform. Depending on the status byte, a number of different byte patterns will follow. For example, the Note On status byte tells the MIDI device to begin sounding a note. Two additional bytes are required: a pitch byte, which tells the MIDI device which note to play, and a velocity byte, which tells the device how loud to play the note [Lip89].

MIDI channels

MIDI channels make it possible for instruments in a MIDI system to play a unique part of a more complex composition. Each channel carries a different part of the final sound and corresponds to a single instrument. Therefore the MIDI messages sent to a channel will be processed using this channel's instrument. MIDI operates on 16 different channels, numbered 0 through 15. MIDI messages that apply to a specific channel are called Channel Messages and the channel number is included in the status byte for these messages. For example, a Note On status byte always includes the channel on which the specified note should be played, i.e. what instrument should be used to play the note. System messages are not channel specific, and no channel number is indicated in their status bytes. MIDI files are mainly composed of the following messages:

Note On The Note On message tells the synthesizer to start playing a note. Two additional bytes specify the pitch and velocity value of the note. The note is played using the instrument of the channel specified in the status byte.

Note Off The command to stop playing a note is not specified in the Note On message. Instead there is a separate Note Off command. The two additional bytes have the same functions as the Note On bytes. However the Note Off velocity information is normally ignored.

Program Change The Program Change message is used to specify the type of instrument which should be used to play sounds on a given channel. This message needs only one data byte which specifies the new program number.

Control Change MIDI Control Change messages are used to control a wide variety of functions in a synthesizer. Control Change messages, like other MIDI Channel Messages, only affect the channel number indicated in the status byte. The Control Change status byte is followed by one data byte indicating the "controller number", and a second byte which specifies the "control value". The controller number identifies which function of the synthesizer is to be controlled by the message, and the control value forms the parameter of the function. For example, the controller number 10 is used to change the panning settings, and the control value represents the new panning value (e.g. a number between 0 and 127 where 0 means everything on the left and 127 everything on the right speaker).

5.3 Choosing the right format

The selection of best sound format for a sonification application is based on the application purposes and the implementation context. While sampled sounds and MIDI may be used in parallel, a sound system created with a single format offers a much easier implementation and management of the system.

The first thing to consider should be storage space. If the application is executed on a system with limited storage capacities, the amount of possible sampled sound files will be restricted. This is especially true for Internet-based applications, where the produced sound quality depends on the communication speed. Compared with WAV files, MIDI files take considerably less time to load and the sound quality only depends on the presented sound card.

Another essential point is the type of sonification that the application needs. For example, auditory icons use recordings of real-world sounds, and are thus perfectly suited for sampled sounds. One more area where sampled sounds are very popular is speech synthesis, since the very own nature of MIDI makes it inappropriate for speech or other special sound effects. On the other hand, MIDI allows one to easily edit and alter sounds, e.g. by modifying the playback speed and the pitch of the notes independently. This is particularly helpful when using the parameter mapping sonification, which demands a lot of tweaking and testing before finding the optimal solution.

If the desired sound quality is very high, MIDI could cause some problems, as the same MIDI file may sound quite different on several sound systems. Sampled sound though almost guaranties that the musical output will correspond to the recorded version. However, modern sound cards now generally provide high quality sound and the pre-recorded instruments on the MIDI synthesizer are improving. This minimizes the risk of having an application sound totally different on two sound systems.

Last but not least, the implementation context must be considered. Whether using sampled sounds or MIDI, playing an existing sound file is quite straightforward. The problems arise when real-time sounds need to be created. WAV files are particularly hard to generate in real-time, as they require a lot of expertise and specialized software. Creating real-time MIDI is much more feasible, although time management can be tricky. Indeed, creating and sending a MIDI message to a synthesizer is quite easy, but it requires the programmer to handle time intervals between messages, since upon receiving a message the synthesizer instantly performs the requested action.

5.4 MIDI and Java

5.4.1 The Java Sound API

Before the release of Java 2, sound support was the main weakness of Java, that supported only 8-bit, 8-kHz audio and an extremely limited range of file types. All that has changed with the incorporation of the Java Sound engine in Java 2. Another step forward has been taken with the beta release of Java 2 version 1.3, which exposes the Java Sound API. Java Sound can be made available to earlier Java 1.x platforms through the use of the Java Media Framework 2.0. The Sound API provided by version 1.3 of the Java 2 Platform offers low-level support for capturing, processing, and playing back sampled audio and MIDI data. Java Sound is a 16-bit, 48-kHz sound synthesis engine that supports a wide variety of file types [Hob00].

Java Sound provides low-level audio support for Java with a very high degree of control over audio-specific functionalities. It is not meant to function as a standalone application environment, but as a set of interfaces upon which applications can be built such as audio editors, MIDI sequencers, games, etc [Anr99].

The Java Sound API provides support for both sampled sound and MIDI. These two formats are completely disconnected in the API specification, as they are provided in separate packages. The Java Sound API consists of four packages:

```
javax.media.sound.sampled
javax.media.sound.midi
javax.media.sound.sampled.spi
javax.media.sound.midi.spi
```

The first two packages provide interfaces supporting sampled audio and MIDI sequencing and synthesis; the .spi packages provide service providers with abstract interfaces to enable the installation of custom components [Hob00]. This dissertation will focus on the second package, since the parameter mapping sonification technique primarily uses the MIDI sound format.

5.4.2 The MIDI package

The `javax.media.sound.midi` package is dedicated to MIDI. It provides all the classes and interfaces to play back, modify and create MIDI data. The package has been conceived with the MIDI file format in mind. Indeed, almost every MIDI notion has a corresponding Java Sound class or interface. For example, just as the sound card synthesizer plays messages from a MIDI file, the `Synthesizer` interface is used to process MIDI messages, which are represented by the `MidiMessage` class.

Messages

`MidiMessage` is the base class for MIDI messages. It is an abstract class that represents a "raw" MIDI message, with no timing information. `MidiMessage` is divided into three subclasses:

- The `ShortMessage` class represents the most frequent messages, like Note On, Note Off and Program Change. They have at most two data bytes following the status byte.

- The `SysexMessage` objects represent MIDI system exclusive messages. They are used to configure devices, and transfer digital information such as sample files between devices.

- A `MetaMessage` is a `MidiMessage` that is not meaningful to synthesizers, but that can be stored in a MIDI file and interpreted by a sequencer program. There are meta-events for such information as lyrics, copyrights, tempo indications, etc.

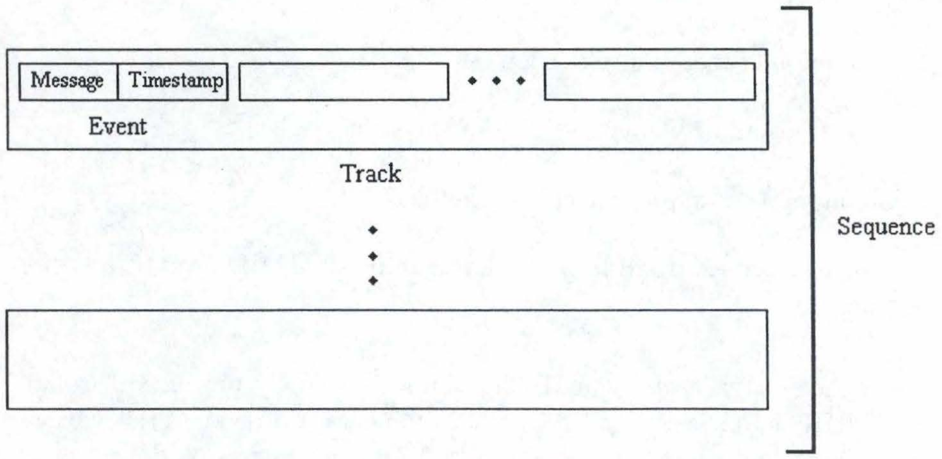


Figure 5.1: Structure of a MIDI sequence

Events

MIDI events are essentially MIDI messages with some timing information. A `MidiEvent` object contains a MIDI message and a corresponding timestamp expressed in ticks. The duration of a tick is specified by the timing information contained in the MIDI sequence. A `Sequence` can be read from a standard MIDI file or created from scratch by combining tracks made up of events. A `Track` is an independent stream of MIDI events that can be stored along with other tracks in a MIDI sequence. The MIDI specification allows only 16 channels of MIDI data, but tracks are a way to get around this limitation. A `Sequence` object can contain any number of tracks, each containing its own stream of up to 16 channels of MIDI data [Jav00a]. Thus sequences contain tracks, which contain events, which contain messages. This is illustrated in figure 5.1.

Devices

MIDI devices can send or receive MIDI events. The `MidiDevice` interface serves as a base interface for common devices such as synthesizers and sequencers. In order to exchange MIDI events, devices typically also implement the `Transmitter` or `Receiver` interface.

A sequencer is a device used to play back, create and edit MIDI sequences. To this end, sequencers use receivers to capture data and transmitters to send it. The `Sequencer` interface includes methods for the following basic operations [Jav00a]:

- Obtaining a sequence from MIDI file data
- Starting and stopping playback
- Moving to an arbitrary position in the sequence
- Changing the tempo (speed) of playback
- Editing the data by adding or deleting individual MIDI events or entire tracks

A synthesizer generates sound. The sound is actually produced through a set of MIDI channels controlled by the `Synthesizer` object. Many synthesizers support `Receivers`, through which MIDI events can be delivered to the synthesizer. In such cases, the synthesizer typically responds by sending a corresponding message to the appropriate `MidiChannel`, or by processing the event itself if the event is not channel specific [Jav00a].

The `Synthesizer` class also provides methods for manipulating soundbanks and instruments. An `Instrument` represents the specific sound played by a `MidiChannel` object, by specifying how the sound should be synthesized (i.e. how to create the audio signal). A soundbank is a collection of instruments, organized by bank and program number. These two elements form a `Patch` object that is used to select a specific instrument. Although a default soundbank is present, it is possible to modify the `Soundbank` object by reading a new soundbank file.

5.4.3 Implementation issues

While the concepts of sequencer and synthesizer are relatively easy to understand, it is much harder to comprehend how these objects can be combined to produce sound. A synthesizer generates sound, with or without a sequencer. Individual notes can be sent directly to the synthesizer, or it can obtain a sequence of notes via a sequencer.

To decide whether to use a sequencer or not, one must first select the appropriate timing method. MIDI messages can be created and sent in real-time, or they can be part of an existing sequence that will be sent to the synthesizer at some later time. Generally, real-time sonifications are much easier to implement as they do not use sequencers. In both cases, sending notes to the synthesizer often requires a precise scheduling of the messages containing these notes. As previously shown, this is done through the use of time stamps, but the implementation of time stamps within the Java Sound API is quite intricate.

When to use a sequencer

When implementing a MIDI-based sonification, one needs to determine whether the MIDI messages will be created in real-time or not. Real-time sonification is often used when the MIDI events are generated by the user, i.e. a MIDI event corresponds to a specific action from the user. In such cases, the MIDI messages are sent directly to the synthesizer, without using a sequencer. This is quite logical, considering that sequencers play sequences, and that no sequence is present at this point. For example, let us consider a program that lets the user play notes by clicking on an onscreen piano keyboard. When the program gets a mouse-down event, it immediately sends the appropriate Note On message to the synthesizer [Jav00b].

As MIDI events are contained in sequences, real-time sonification makes no use of `MidiEvent` objects. Instead, `MidiMessage` objects are sent to the synthesizer's `Receiver` object, along with a time-stamp argument. This may seem strange, since messages sent to the synthesizer are supposed to be handled immediately, but this kind of time stamp is only used for fine-tuning the timing. It is designed to help compensate for latencies introduced by the operating system or by the application program [Jav00b]. The time value is expressed in microseconds and represents the time elapsed since the device that owns the receiver was opened (e.g. the `Synthesizer` object). This value must be close to the present time, or the receiving device might not be able to schedule the message correctly. A message whose time stamp is very far in the future will not be handled correctly, and certainly not if its time stamp is in the past. It is up to the device to decide how to handle time stamps that are too far off in the future or are in the past. However, most of the time a value of -1 is used, which means that the device will simply try to respond to the message as soon as possible.

If an application program wanted to create a queue of MIDI messages for an entire piece of music ahead of time (instead of creating each message in response to a real-time event), it would have to be very careful to schedule each `MidiMessage` for nearly the right time [Jav00b]. Fortunately a program can add a sequencer to the synthesizer and let the `Sequencer` object manage the queue of MIDI messages. This requires the messages to be coupled with "true" time stamps to form MIDI events. The sequencer actually plays the `Sequence` containing these `MidiEvent` objects. It takes care of scheduling and sending the messages, i.e. playing the music with the correct timing. Generally, sequencers are used for playing data from MIDI files, and whenever the application program permits dynamic alterations of the MIDI signal (e.g. changing playback speed, rewinding or moving to particular points in the sequence).

Duration of a tick

MIDI uses two kinds of time stamps. Time stamps expressed in microseconds are used by a device's receiver along with a MIDI message to correct processing latency. The second kind of time stamps are encapsulated together with MIDI messages in a `MidiEvent`. In this case, the timing is expressed in abstract units called ticks.

The duration of a tick varies between sequences, but usually not within a sequence. The size of a tick is given in one of two types of units:

- Pulses per quarter note, abbreviated as PPQ.
- Ticks per frame, also known as the SMPTE time code (a standard adopted by the Society of Motion Picture and Television Engineers).

The main difference is that PPQ units use relative time and SMPTE absolute time. Representations of absolute time follow hours, minutes and seconds just like a watch. Therefore absolute time is always the same and cannot be speeded up or slowed down. Relative time is a reference to a musical piece that has an inner tempo. For example, a composition may take three minutes to perform at a tempo of 80 bpm (beats per minute), but would take only a minute and a half if the tempo was increased to 160 bpm.

If the unit is PPQ, the size of a tick is expressed as a fraction of a quarter note, which is a relative time value. A quarter note is a musical duration value that often corresponds to one beat of the music. The duration of a quarter note is dependent on the tempo, which can vary during the course of the music if the sequence contains tempo-change events [Jav00b].

On the other hand, SMPTE units measure absolute time, and the notion of tempo is inapplicable. There are actually four different SMPTE conventions available, which refer to the number of motion-picture frames per second. The number of frames per second can be 24, 25, 29.97, or 30. With SMPTE time code, the size of a tick is expressed as a fraction of a frame [Jav00b].

Concretely, an application program usually needs to convert standard time values (expressed in seconds) to ticks (expressed in PPQ or SMPTE units). This conversion is trivial once the duration of a tick is computed. The following formulae calculate the number of ticks per second:

$$ticksPerSecond = resolution * \frac{tempoInBeatsPerMinute}{60}$$

$$ticksPerSecond = resolution * framesPerSecond$$

where *resolution* is an arbitrary value representing the number of pulses per quarter note if the division type is PPQ, or per SMPTE frame if the division type is one of the SMPTE conventions. The size of a tick is then easily computed with this formula:

$$tickSizeInSeconds = \frac{1}{ticksPerSecond}.$$

To convert a standard time value (given in seconds) to a time-stamp, one would simply divide the time value by the tick size to obtain the corresponding number of ticks. For example, using the PPQ time code with a resolution of 100 pulses per quarter note, and knowing that the default tempo is 120 bpm, the conversion of 200 milliseconds (0.2 seconds) needs following calculations:

$$ticksPerSecond = 100 * \frac{120}{60} = 200$$

$$tickSizeInSeconds = \frac{1}{200} = 0.005$$

$$numberOfTicks = \frac{0.2}{0.005} = 40.$$

Thus, at the previously stated resolution and tempo, 200 milliseconds are equivalent to 40 ticks.

A last remark concerns the difference between time stamps contained in standard MIDI files and `MidiEvent` objects. The tick values contained in `MidiEvent` objects measure *cumulative* time, rather than *delta* time. In a standard MIDI file, each event's timing information measures the amount of time elapsed since the start of the previous event in the sequence. This is called delta time. But in the Java Sound API, the ticks represent the previous event's time value *plus* the delta value. In other words, in the Java Sound API the timing value for each event is always greater than that of the previous event in the sequence (or equal, if they occur simultaneously). Each event's timing value measures the time elapsed since the beginning of the sequence [Jav00b].

In conclusion, a sequence can either use PPQ or one of the SMPTE units. However, since SMPTE units are unaffected by tempo variations, it is impossible to dynamically change the playback speed of a sequence with a SMPTE-based time code. Therefore most MIDI sequences use the PPQ relative timing code.

5.5 Conclusion

Sampled sounds are best suited to sonifications that require natural or special sound effects, like alarm signals, earcons and auditory icons. On the other hand, parameter mappings (especially pitch-based sonifications) prefer the MIDI sound format. While both file formats are relatively easy to play back, it is much more feasible to create or modify a MID file than a WAV file. Once the particular implementation issues have been overcome, MIDI offers much more possibilities to implement the best possible sonification.

Chapter 6

SoundChart

6.1 Introduction

The opening chapters of this dissertation have introduced the concepts of sound, sonification and time-dependent data. While the chapter on pitch-based mapping covered the most widely used sonification technique, the next one reviewed the available sound formats, more particularly the MIDI format. In order to validate or criticize our findings, two application programs were developed. This chapter covers the first one, called SoundChart.

6.2 Description

SoundChart is a stand-alone Java application providing tools for the sonification of time-dependent two-dimensional data. The purpose of the application is to support the designer through the whole sonification process, from data display and preprocessing to the sonification procedure itself.

From now on, a designer will designate a person who uses the application and sets the sonification parameters. Users or listeners represent end-users of the application, who listen to the signals generated by a designer. Since sonification is a technique that can replace or support visual information, users do not necessarily need to be using SoundChart. For instance, the sound signal can be created by a designer using SoundChart and then sent to one or more users through some network.

The sonification process is achieved by following three successive steps, each one corresponding to a panel in the application. These three panels are described in the next sections.

6.3 Data management

The first step for any sonification process is to select the data that will be used. The data obviously need to correspond to the specific requirements of the application. In this case, SoundChart needs two-dimensional data for the representation of time-dependent series.

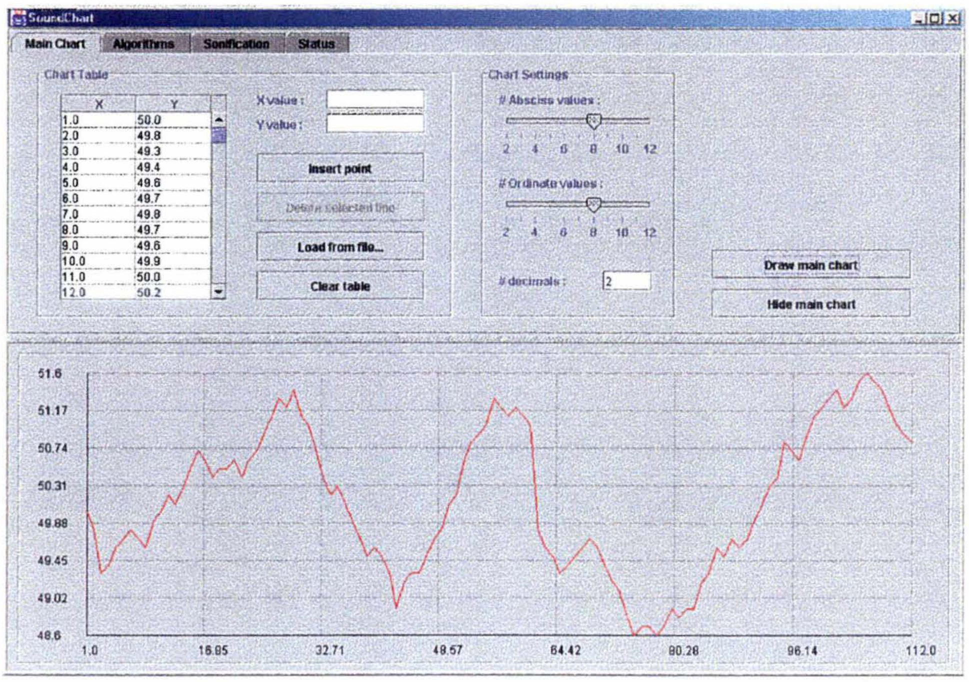
6.3.1 SoundChart file format

SoundChart has its own file format for two-dimensional data. It is a very simple, text-based format, with no predefined file extension. This means that any file can be opened with a text editor and arranged for usage in SoundChart. A SoundChart file contains an unlimited number of lines, each line representing a two-dimensional point, which simply consists in an X and Y value separated by a special character "|". The X value represents the independent time value, while the Y value represents the corresponding dependent data value. Both values are floating point numbers and can be negative.

Each couple is supposed to represent a single point in the time series. Time values are usually ordered in chronological way, but SoundChart does not impose this restriction. The arrangement of the X values in the file itself is left to the designer. Moreover, if the file contains two data points occurring at the same time (i.e. they have the same X value), SoundChart only considers the last one in the file to be valid. For example, the two following file contents result in exactly the same data.

```
0|0.0                   1|4.7
1|-0.5                  0|0.0
2|1.75                  3|0.2
3|0.2                   2|1.75
4|1                     4|1
5|2.3                   5|2.3
                        1|-0.5
```

While some time series are continuous, most of them contain discrete time data, i.e. observations at usually regularly spaced intervals. SoundChart only works with discrete time series, and accepts any interval size between two consecutive time values, as long as this size is constant over time. For instance, the interval size could be expressed in days (e.g. daily rainfall), months (e.g. monthly profits), or even years (e.g. crime figures). Of course these time values need to be given as numerical values, but no particular format is imposed. For example, the time series ((50.01|2), (50.02|4), (50.03|0), (50.04|3)) is perfectly valid.



X	Y
1.0	50.0
2.0	49.8
3.0	49.3
4.0	49.4
5.0	49.6
6.0	49.7
7.0	49.8
8.0	49.7
9.0	49.6
10.0	49.9
11.0	50.0
12.0	50.2

decimals : 2

Figure 6.1: Main Chart panel

6.3.2 Main chart panel

The first SoundChart panel is called the "Main Chart" panel, as it enables to load a time series and then display it graphically on a chart. This time series forms the main component of the application, as every possible sonification will be derived from it. An example of the Main Chart panel is displayed at figure 6.1.

Once the SoundChart file has been chosen or created, it becomes possible to import it into SoundChart. However, this is not compulsory, as data values can be manually added to the data set by using the specified textfields. This way one could build a complete time series without loading any data from a file. The designer can also delete or modify existing data points.

Whenever the designer is satisfied with the present data, he can ask SoundChart to display the associated two-dimensional chart. The scale is automatically computed by the application so that the chart fits in a single screen and remains understandable. The designer has only access to a couple of parameters, namely the number of values per axis and their decimal point precision.

At this point, it is important to specify that the graphical representation is only intended to help the designer during the sonification process and should be considered as an optional part of the application.

6.4 Preprocessing

Once the data has been selected, the second step of the sonification process is to prepare the data for sonification. Raw time series always contain random fluctuations that have to be filtered out. SoundChart does this by using moving average smoothing techniques. The preprocessing step in SoundChart does not handle extreme values, as this is done by the third and biggest step in the process, namely the sonification procedure.

6.4.1 Smoothing methods

SoundChart uses three different smoothing methods, all based on the moving average. Their names are the linear, exponential and weighted moving average. It is almost impossible to designate the best one, since the efficency of a moving average greatly depends on the present data and the size of the smoothing window. However, in practise the linear moving average proves to be the most effective one.

Linear Moving Average

The linear moving average is the most common moving average type. The smoothing amount is defined by an order p, which defines the length of the smoothing window. The formula of the linear moving average is

$$\mu_n = \frac{1}{\sum_{i=1}^{p} i} \sum_{i=1}^{p} i x_{(n-i+1)}.$$

Since every μ_n is computed using the $p-1$ previous values of the corresponding x value, the first point of the new linear moving average data set starts at the p^{th} time index. The optimal order p is impossible to predict, but is generally very small compared to the total number of data points.

Exponential Moving Average

An exponential moving average is calculated by applying a percentage of today's data value to yesterday's moving average value. The exponential percentage e can be linked with a time period p, according to this formula:

$$e = \frac{2}{p+1}.$$

Again, this time period is usually very small compared to the total number of data points. If p is too large, e will be too small, and the exponential moving average data set will almost be similar to the original data set. Most of the time a percentage value around 10% gives a good result. The new data set is then computed with the following formula:

$$\mu_n = (1 - e)\mu_{n-1} + e x_n.$$

The advantage of an exponential moving average is that the new time series has exactly the same length as the original one, since no values other than x_n are needed to compute μ_n.

Weighted Moving Average

A weighted moving average is designed to put more weight on recent data and less weight on past data. Like the other moving averages, it uses some sort of smoothing window, called the coefficient. A weighted moving average with coefficient c is computed with the following formula:

$$\mu_n = \frac{\sum_{i=0}^{c-1}((c-i)x_{n-i})}{\sum_{i=0}^{c-1}(c-i)}.$$

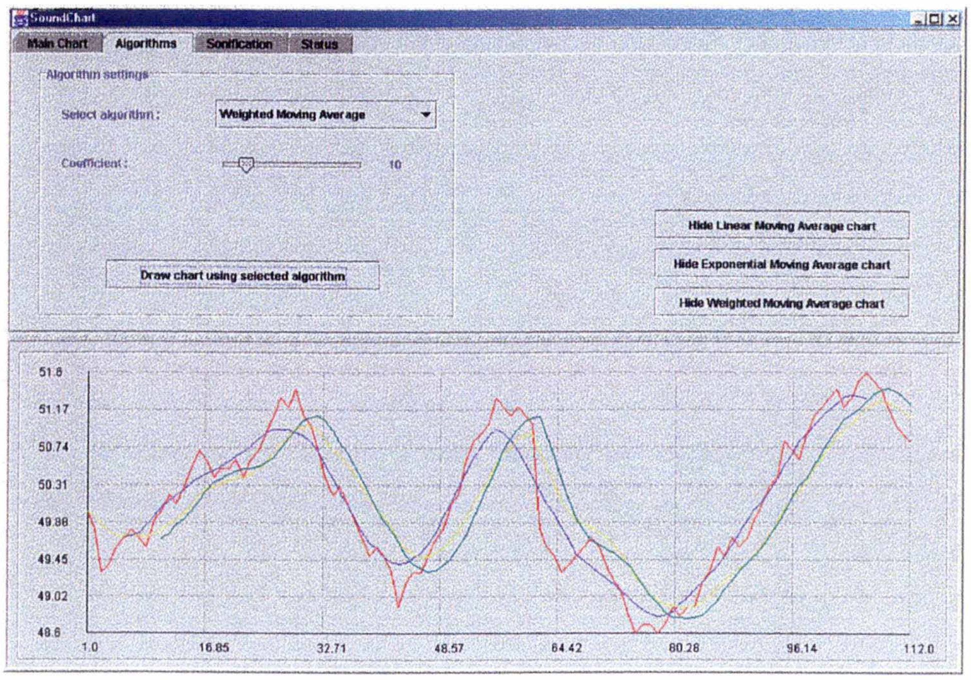
A weight is assigned to each data value in the smoothing window, so that the more recent values have larger weights. Thus the value x_n has a much greater weight then the value x_{n-c+1}, which means that x_{n-c+1} is less important in the computation of μ_n.

As usual, the coefficient c is only a fraction of the total number of data points. Moreover, the $c-1$ previous values of x_n are used to calculate μ_n, resulting in a shorter time series.

6.4.2 Algorithms panel

The second SoundChart panel, called the "Algorithms" panel, lets the designer choose between three smoothing algorithms. There are three parameters that he can change, one for each algorithm. These are the order of the linear moving average, the exponential percentage, and the coefficient of the weighted moving average.

When the designer chooses to display any of these methods, the corresponding time series is computed and displayed on top of the existing chart(s). For instance, figure 6.2 shows an example of the algorithms panel where all the available charts have been drawn. The red chart is the original one, and the linear, exponential and weighted moving averages correspond to the blue, yellow and green charts, respectively. As expected, the blue and green charts start a little later than the other ones, indicating that the linear and weighted moving averages have less data values.



Select algorithm: Weighted Moving Average

Coefficient: 10

Figure 6.2: Algorithms panel

6.5 Sonification

The third and final step of the sonification process is of course the sonification itself. This is the most complex part, as a lot of sonification techniques are available to the designer, but none of them is very well documented. For this reason, SoundChart offers several sonification methods, which all have a couple of parameters that can be adapted to suit the designer's requirements.

6.5.1 Extreme values

Before any sonification can be applied, SoundChart needs to test the original time series for possible outliers. Whatever time series is chosen for sonification, existing outliers only appear in the original data, since the moving average data sets contain smoothed versions of the data. Therefore the detection method is always applied to the original time series.

SoundChart uses the z-score method to detect extreme values, and does not differentiate outliers that are due to errors and ones due to chance. They are both treated as extreme values and sonified as such. Once a data value has been identified as an outlier, SoundChart stores its corresponding time value and keeps it ready for the ultimate sonification.

6.5.2 Pitch-based mapping

SoundChart uses parameter mapping for its sonification, more precisely pitch-based mapping. The data values from the chosen time series are mapped to frequencies, by using linear as well as chromatic scale mappings. These mappings use the maximum and minimum data values from the chosen time series, and a frequency range given by the designer. It is important to note that the calculation of these upper and lower bounds is done without taking into account the identified outliers, since the distribution of frequency values would otherwise be falsified by an incorrect maximum or minimum data value.

MIDI implementation

SoundChart was created using Java, and uses the MIDI package of the Java Sound API. Because the designer or the user is able to dynamically change the auditory signal, for example by pausing or forwarding the signal, SoundChart does not permit real-time sonification. The MIDI sequence is constructed prior to the actual playback. When the designer starts the sonification, the whole sequence is computed and then sent to the MIDI sequencer for playback. However, this process happens so quickly that the user gets an impression of real-time sequencing.

SoundChart constructs sequences composed of a single track and makes use of only three channels. The first channel is used for the actual data values (i.e. their mapped frequencies), while the others are used for enhancements to the sonification. All the messages used in SoundChart are `ShortMessage` objects. The sequence starts with three Program Change messages, which are used to select an initial instrument for each channel. Actually no other Program Changes will occur, so these instruments will be used all along the sonification. The rest of the sequence contains mostly simple Note On and Note Off messages, from which the majority belongs to the first channel.

The time stamps associated with these Note On and Note Off messages are based on a constant note interval, selected by the designer and expressed in milliseconds (ms) . SoundChart uses the assumption that all time values occur at regularly spaced intervals. Therefore a constant note interval is used to separate two consecutive notes in the sequence. For example, let us consider five consecutive data values, as shown in figure 6.3. Using a 200 ms interval, the first `MidiMessage` (corresponding to the first value) will have a time stamp of 0 ms, the second one a time stamp of 200 ms, and so on until

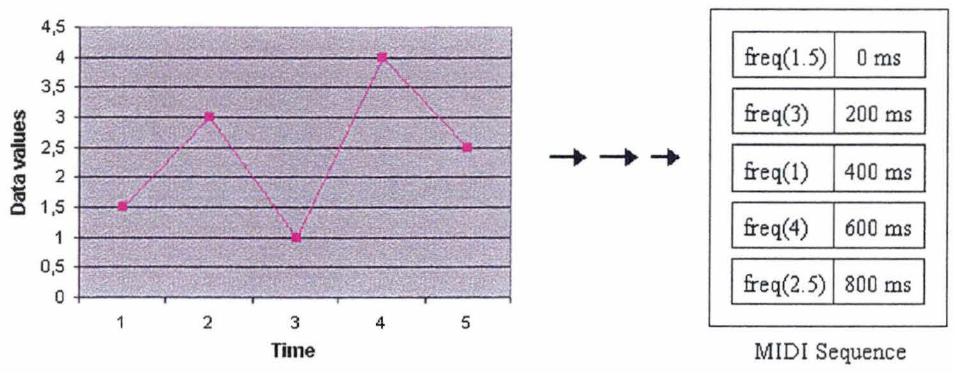


Figure 6.3: Five data values mapped with a 200 ms interval

the last one, which will have a time stamp of 800 ms. The notes are clearly separated by a 200 ms interval.

A musical note is in fact represented by two MIDI messages. One is used to start playing the note (Note On), the other to stop playing the note (Note Off). The duration between these two messages represents the length of the note. Although SoundChart uses discrete time series, the sonification will be better perceived by the user if a continuous sound is produced. Therefore, SoundChart stops playing a note exactly when the next one is started. Thus the Note Off message of a note always coincides with the following note's Note On message (except for the last note).

Every message is encapsulated into a MIDI event, together with its time stamp. A `MidiEvent` object is created using the following four components:

- The channel on which the message is going to be played. If this message represents a note (and not a Control Change for example), the channel defines which instrument will be used to play the note, i.e. the instrument indicated by the initial Program Change message corresponding to this particular channel.
- The status byte of the message, indicating which MIDI command the event represents.
- One or two additional data bytes (depending on the status byte), which contain the parameters for the MIDI command.
- The time stamp of the message, given in milliseconds but converted to ticks to suit the Java Sound API. In SoundChart, ticks are represented in PPQ units, thus depending on the tempo.

The first channel is used for playing the frequencies corresponding to the chosen time series. In its most basic form, SoundChart uses only two messages for this sonification, namely Note On and Note Off messages. The `MidiMessage` objects representing these messages are very similar, as only the status byte tells them apart. They both have the same frequency and velocity values, which form the two additional required data bytes of the object. SoundChart simply uses the highest possible velocity (127), which gives optimal results.

The second channel is used for notifying extreme values. While SoundChart is creating `MidiEvent` objects and adding them to the current sequence, it also checks the list of identified outliers. If the value being processed is part of the list, the event is not added to the sequence. Instead, SoundChart creates a new `MidiEvent` object that represents an alarm signal. The alarm uses a MIDI instrument like the other notes, that can be changed by the desginer. An alarm signal therefore also consists of a Note On and Note Off message, but the corresponding `MidiMessage` objects are sent to the second channel, i.e. using the specified alarm instrument. The interval between the two messages is set to 300 ms, meaning that alarm notes have a duration of 300 ms.

Sonification improvements

The first improvement is to add a slope indicator to the signal, by the use of drum beats. Because variating the pitch of the drum beats would ask too much efforts from the listener, SoundChart uses a constant pitch value for the drum beats. This value is set to 60 (knowing that 0 is the minimum and 127 the maximum pitch), in order to be clearly audible without interfering with the main notes.

At each data point, SoundChart calculates the slope by using the preceding data point. Depending on the slope's value, it then adds a certain amount of MIDI messages to the current sequence, each message couple (Note On, Note Off) representing a single drum beat. The corresponding `MidiMessage` objects are sent to the third channel, which again uses a specific instrument specified by the designer. The duration of a drum beat is set to 250 ms.

SoundChart also allows using stereo panning, to obtain some sort of progression indicator. Changing the intensity level of the speakers is achieved by a single Control Change message. The first additional data byte is set to 10, which is the value for panning control, and the second one to the desired panning value. This value must lie between 0 and 127, with 0 meaning all output to the left speaker, and 127 to the right speaker. At every data point, SoundChart adds such a Control Message to the sequence, representing the current percentage of the time data that has been processed. For example, a panning value of 38 indicates that 30% of the sequence has been played. The panning messages are sent to the first and third channel only, since outlier signals are usually quite rare and do not belong to the "normal" signal.

Putting it all together

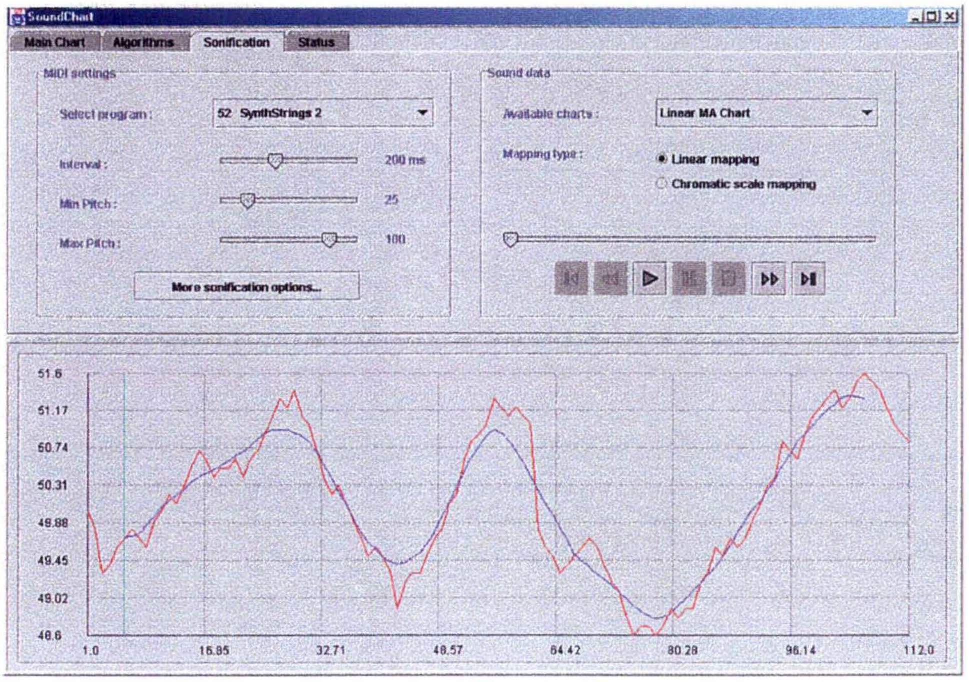
When the designer starts the sonification, the MIDI sequence corresponding to the chosen time series is computed with the selected parameters. The next few steps summarize the sequence creation process.

1. The original time series is analysed and every extreme value is identified.
2. Three Program Change messages are added to the sequence, one for each channel. The specified instruments are chosen by the designer.
3. For each data point from the chosen time series, SoundChart executes the following actions:
    - A note frequency is calculated based on the given frequency range, using linear or chromatic scale mapping.
    - If the data point has been previously identified as an outlier, an alarm signal is added to the sequence (a Note On, Note Off couple).
    - Otherwise, three messages are added to the sequence. First a panning Control Change message is calculated based on the current data point's index. Then the two main messages representing the note frequency are added. The interval specified by the designer is used to set the duration of the note, i.e. the difference between the Note Off and Note On messages.
    - Drum beats are added to the sequence. The number of inserted messages depends on the slope value. The higher the slope, the more drum beats are added.

6.5.3 Sonification panel

The third SoundChart panel is called the "Sonification" panel. The designer has access to several options that determine how the sequences are constructed. As shown in figure 6.4, the available MIDI settings are:

- The main MIDI program, which represents the instrument that will be used on the first channel.


SoundChart

Main Chart | Algorithms | Sonification | Status

MIDI settings

Select program : 52 SynthStrings 2

Interval : 200 ms

Min Pitch : 25

Max Pitch : 100

More sonification options...

Sound data

Available charts : Linear MA Chart

Mapping type : [✓ Linear mapping] Chromatic scale mapping

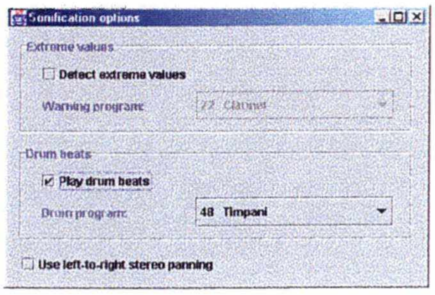
Figure 6.4: Sonification panel

- The interval between two consecutive notes.
- The minimum and maximum pitch values used for the linear and chromatic scale mappings.

Figure 6.5 shows some additional parameters. The designer can choose whether or not to detect extreme values, to play drum beats, and to use stereo panning. Moreover, he can change the instruments used for notifying outliers and for playing drum beats.

The right portion of the panel presents the time series available for sonification. Furthermore, the designer can select which mapping type will be used for mapping the selected time data to frequencies. The rest of the panel contains the sequence's slider and its command buttons. The seven buttons have standard meanings. Going from left to right, there is a button to reset the sequence to the beginning, rewind the sequence, play the sequence at the current position, pause the sequence, stop the sequence, forward the sequence and finally advance all through the end of the sequence.

The designer or the user starts the sequence by pressing the play button. Actually, the sequence is computed when the play button is pressed, but this happens so quickly that it seems as if the sequence already existed. Once the sequence starts playing, the slider follows its progression. This is also



Extreme values

☐ Detect extreme values

Warning program: 72 Clarinet

Drum beats

☑ Play drum beats

Drum program: 48 Timpani

☐ Use left-to-right stereo panning

Figure 6.5: Additional sonification options

indicated by a vertical line on the chart. When the sequence is stopped, it is possible to move the slider to any desired position. The vertical line then follows the indicated position, which can be useful to place the slider at a very particular point in the series.

It is important to remember that all this visual information should just be considered as an aid to the user. Because we are not familiar with auditory interfaces, some functions that could be implemented with sound are handled in a graphical way. For instance, the series of buttons controlling a sequence could be replaced by a speech recognition technique responding to the user's voice. Even all the options described above could be selected using this method, enabling the user to choose and change an option in a very natural way. This obviously requires some serious implementation and testing, but there are several software packages available that are able to do that.

6.6 Conclusion

The SoundChart application program allows the sonification of time-dependent data series. It is certainly not perfect, but it definitely illustrates and implements some of the best techniques to sonify two-dimensional data.

Chapter 7

SoundChart3D

7.1 Introduction

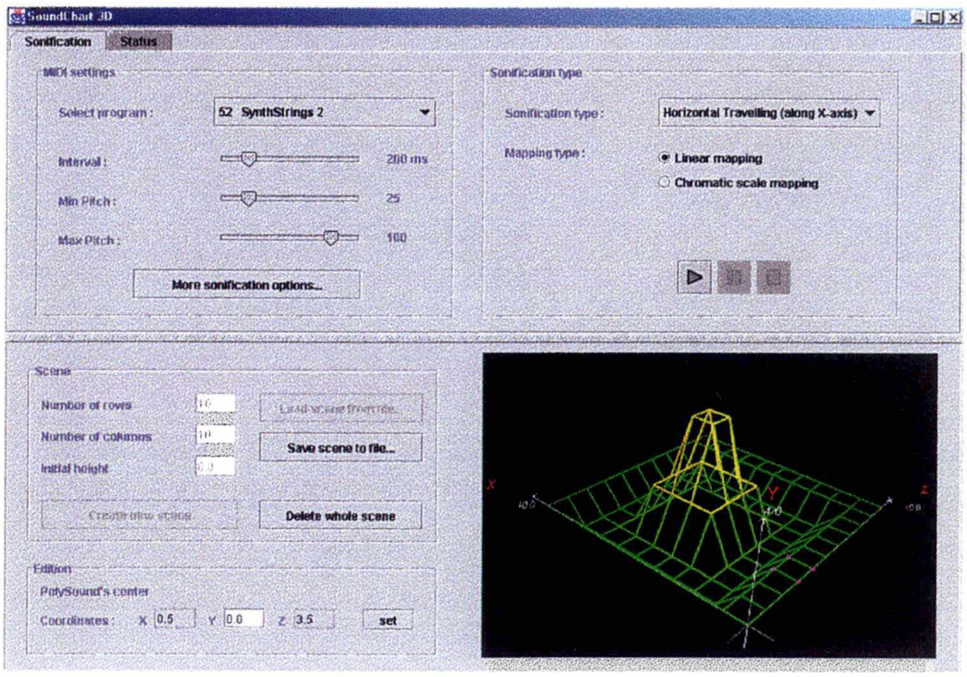
The previous chapter has described our first application, SoundChart, which offers the sonification of two-dimensional data. The goal was to translate a typical graph with X and Y-axis into music while keeping the main characteristics and trends from the chart. Satisfied with the results, we have developed a second application, providing this time the sonification of three-dimensional (3D) data. This application is called SoundChart3D. The aim was this time to translate a 3D graph into sound without using specific 3D audio hardware, that means using only two speakers.

7.2 Description

SoundChart3D is a stand-alone Java application providing a complete tool set for the creation, edition and sonification of three-dimensional data. The purpose of the application is to support the designer through the whole sonification process, from the design and display of 3D graphs to the sonification itself.

7.3 Data management

SoundChart3D provides easy and powerful tools for creating and displaying 3D charts. While the scene panel contains the data management tools such as loading, saving or conceiving a 3D graph base, the edition panel allows the designer to edit and modify any part of an existing 3D chart. Lastly, the view panel supplies means for displaying the chart according to the designer's



Select program : 52 SynthStrings 2
Interval : 200 ms
Min Pitch : 25
Max Pitch : 100
Sonification type : Horizontal Travelling (along X-axis)
Mapping type : [✓ Linear mapping] Chromatic scale mapping
Number of rows 10
Number of columns 10
Initial height 0.0
Coordinates : X 0.5 Y 0.0 Z 3.5

Figure 7.1: The SoundChart3D main window

needs. The main window of SoundChart3D, containing the panels previously named, is shown in figure 7.1.

7.3.1 The scene panel

Once the application is launched, only two actions in the scene panel are available to the designer, which are creating a new 3D scene or loading a previously saved scene from a file.

If the designer wants to create his own 3D graph from scratch, SoundChart3D will automatically build a scene base according to three parameters. Before explaining the creation of a scene itself, let us introduce the most important object handled by SoundChart3D, the `PolySound`. The PolySound is a main structural unit whose members contain all the information needed for the creation, the modification and the sonification of a 3D scene. A single PolySound can be considered as a four-sided polygon, a quadrilateral. As opposite sides are equal and parallel, PolySounds can even be extended to parallelograms. Thus, a PolySound is defined by four points, themselves defined by three coordinates in space: x, y and z. The 3D engine of SoundChart3D has been developed with the Java 3D API provided by Sun. The coordinate system of the Java 3D virtual universe is right-handed.

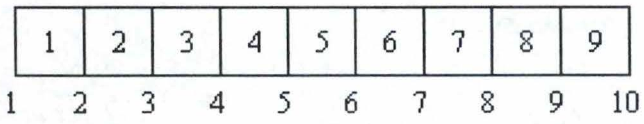


Figure 7.2: A row of 9 PolySounds but 10 points

The X-axis is positive to the right, the Y-axis is positive up, and the Z-axis is positive toward the viewer, with all units in meters [Jav99]. In addition to these parameters providing information about the shape and the location of a PolySound, other visual parameters are available such as the colour and the appearance. Now that the notion of PolySound has been introduced, the three parameters given during the conception of a new 3D scene are described below:

Number of rows The number of rows indicates the number of PolySounds to be created along the Z-axis. As already stated, the main graphical unit in SoundChart3D is the PolySound which contains four points. As adjacent PolySounds share some points, if the designer decides to create a scene with nine PolySounds per row, each row will contain ten points, as shown in figure 7.2. This distinction between PolySounds and points will reveal its importance during the sonification process.

Number of columns The number of columns indicates the number of PolySounds to be created along the X-axis. The same remark about PolySounds and points can be formulated.

Initial height The initial height parameter represents the initial height of the 3D scene base and can take a positive or a negative value. All the points will then be created at this height.

When all the parameters have been set, SoundChart creates a scene base according to the chosen values. The base is composed of a matrix of PolySounds, parallel to the X and Z-axis and placed at the same given height, expressed in meters. The designer can now modify the shape of the chart according to his needs, using the tools provided by the edition panel.

Once a 3D graph is built, the designer has the possibility to save his work to a file and to load it at a later time. SoundChart3D makes use of its own file format which is quite simple and based on text files. Each line in the file represents the height (i.e. the y-value) of the four points composing a single PolySound, separated by a special character "|". An example of a valid SoundChart3D file is available in appendix A.

7.3.2 The edition panel

The edition panel allows the designer to modify any part of a 3D scene. Using the mouse, he can click on a PolySound to select it. Once selected, four pink squares appear on each corner of the PolySound, meaning that the whole parallelogram is selected and the coordinates of its gravity center are displayed. The height of the gravity center is the only variable available for modification, as x and z values have been set with the number of rows and columns. This variable is computed by the following formula:

$$G_y = \frac{(P1_y + P2_y + P3_y + P4_y)}{4}.$$

where G_y is the y coordinate of the gravity center and $P1_y, \ldots, P4_y$ represent the y coordinate of each PolySound's point. If the designer updates the value of this variable, every four points will be increased by the difference between the new value and the old one. Of course, the difference may be negative, resulting in a pulling down of all the points. However, the designer has the opportunity to select a precise point of the PolySound. This is done by clicking with the mouse on one of the four pink squares. This point selection is very handful for fine tuning the scene since the designer can pick each point one by one and give it a specific height according to his needs.

7.3.3 The view panel

The 3D graph is displayed in the view panel. The view contains the X, Y and Z-axis, the maximum values along each axis and finally the graph itself. Several tools are offered for displaying the scene from the desired viewpoint.

Translation The translation tool lets the designer control the translation (on X and Y-axis) of the scene via a mouse drag motion with the right mouse button.

Rotation The rotation tool lets the designer control the rotation of the scene via the left mouse button.

Zoom The zoom tool lets the designer control the Z-axis translation of the scene via a mouse drag motion with the wheel mouse button. If the mouse is not equipped with a wheel, the zoom can be applied by using the left mouse button while pressing the *Alt* key.

Colouring The 3D graph is automatically coloured when modifications occur. The range of colours takes place between light-green and brown and depends on the height of each PolySound. Indeed, just as a topographic map, while lower PolySounds take a green colour, higher PolySounds tend towards a dark-yellow, brown colour.

Shading The shading mode can be chosen by the designer by pushing the $F3$ key. Two modes are available:

- *Flat – shaded*: This shading model does not interpolate colours across the PolySound. Thus, the PolySound is drawn with a single colour and the colour of one point of the PolySound is duplicated across all the other points.
- *Gouraud–shaded*: This shading model smoothly interpolates the colour at each point across the PolySound. Thus, the PolySound is drawn with many different colours and the colour at each point is treated individually.

Rasterization The designer can change the rendering primitive of the scene by pushing the $F4$ key. This tool defines how the PolySounds are drawn: as points, outlines, or filled.

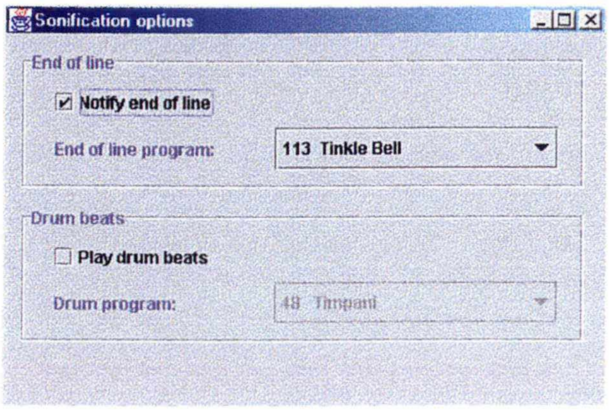
7.4 Sonification

Even if SoundChart3D provides a complete tool set for the creation and visualization of 3D charts, the main goal of the application remains the sonification of the three-dimensional data represented by these charts. Thus, SoundChart3D supplies different tools and sonification techniques to achieve this objective. SoundChart3D uses basically the same pitch-based sonification procedure as SoundChart to translate data into sound. The sonification is made by "cutting into slides" the three-dimensional graph in order to decrease the dimensionality of the data. This way, the initial 3D graph is divided into several 2D graphs which are travelled and auralized one by one. For each 2D graph, the beat drums mapping, inherited from the SoundChart application, can enhance the pitch-based sonification. However, another sonification technique suited to three-dimensional graphs is also implemented. This technique, called the end of line notification, can enhance the sonification by adding a specific sound which indicates the end of a slide. Thus, it becomes easier to identify each slide during the sonification process.

All the tools related to the sonification procedure are placed in two panels. The MIDI settings panel contains all the parameters and settings for the sonification, while the sonification type panel provides a simple sequence player and three sonification techniques for "slide cutting" the 3D graph.

7.4.1 The MIDI settings panel

The MIDI settings panel offers several options that determine how the sequences are constructed. As shown in figure 7.1, the available MIDI settings are:


Sonification options

End of line

[✓] Notify end of line

End of line program: 113 Tinkle Bell

Drum beats

[ ] Play drum beats

Drum program: 48 Timpani

Figure 7.3: Additional sonification options

- The main MIDI program, which represents the instrument that will be used on the first channel.
- The interval between two consecutive notes.
- The minimum and maximum pitch values used for the linear and chromatic scale mappings.

Figure 7.3 shows some additional parameters. The designer can choose whether or not to notify end of lines, and to play drum beats. Moreover, he can change the instruments used for notifying end of lines and for playing drum beats.

7.4.2 The sonification type panel

The sonification type panel allows to choose one of the three sonification techniques proposed in the application. Each of them consists of a travelling way which actually defines the way the 3D chart is cut into 2D charts:

Horizontal travelling The 3D graph is cut horizontally along the X-axis. The sonification starts from the origin of the chart (i.e. the three axis's intersection) and processes each line along the X-axis, one at a time. The parallel purple arrows in figure 7.4 represent the path followed by the sonification in the graph while the orange arrow shows the processing order for each line.

Vertical travelling The 3D graph is cut vertically along the Z-axis. The sonification starts from the origin of the chart and processes each line along the Z-axis, one at a time. According to the shape of the 3D

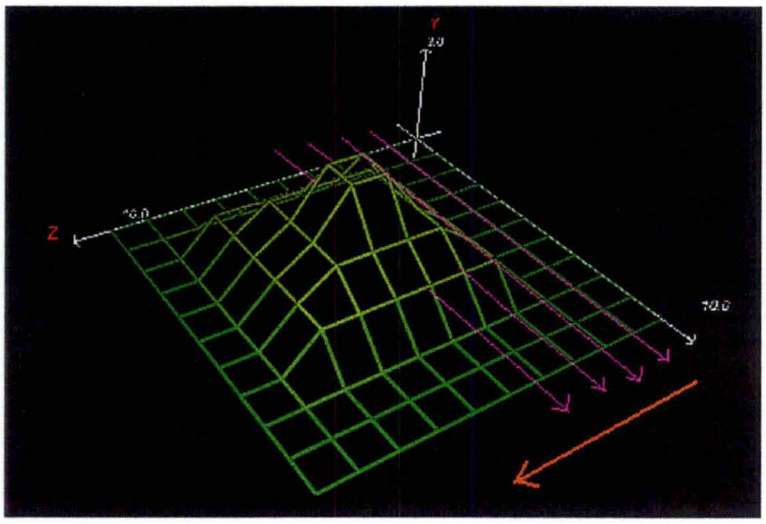


Figure 7.4: Horizontal travelling

graph, the use of horizontal or vertical travellings can radically change the sonification of the graph. Like in the horizontal travelling, the arrows in figure 7.5 show the sonification path for the vertical travelling.

Diagonal travelling The 3D graph is cut diagonally. The diagonal travelling provides information on the length of the auditive signal. The sonification starts from the origin of the chart and processes each line diagonally, from the Z-axis to the X-axis, one at a time. At the beginning of the sonification, the slides are quite short so the sound produced is short as well. But while travelling diagonally through the graph, the slides increase progressively and reach their maximum length at about the middle of the graph. At this point, the auditive signal is the longest. After that, the slides decrease the same way they have increased until the end of the graph. The diagonal travelling is represented by the arrows in figure 7.6.

Furthermore, as in SoundChart, the designer can select between linear or chromatic mapping for the way the data are mapped to frequencies. The rest of the panel contains three command buttons to play, pause or stop the sequence.

7.5 Conclusion

The SoundChart3D application program allows the sonification of three-dimensional time-dependent data series. Like its brother SoundChart, it

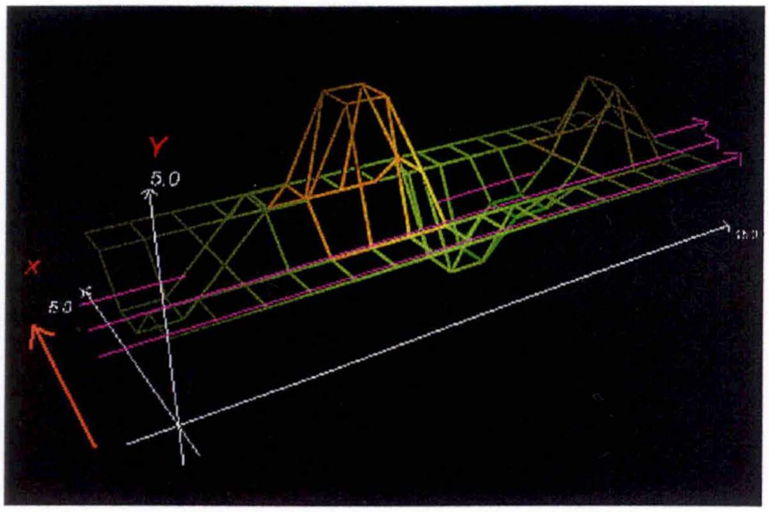


Figure 7.5: Vertical travelling

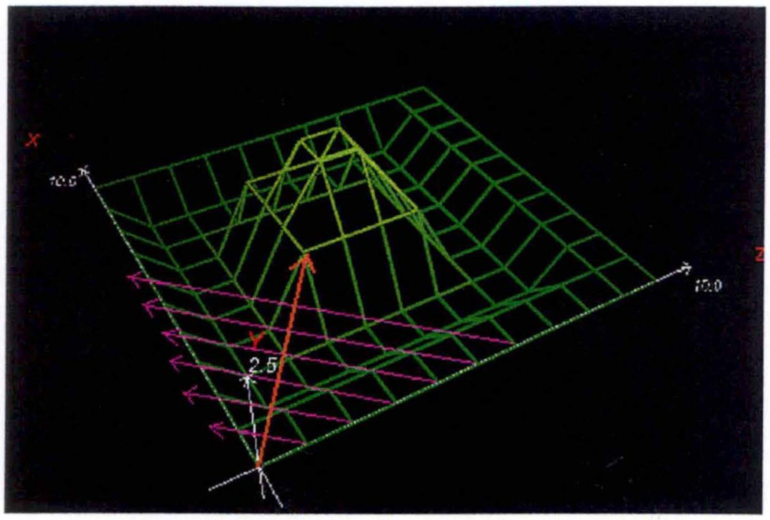


Figure 7.6: Diagonal travelling

can certainly be improved, but it definitely illustrates how 3D data can be translated into sound.

Chapter 8

Experimentation

8.1 Introduction

The purpose of the experimentation is to determine how the sonification of two-dimensional and three-dimensional graphs can support or be an alternative to visually displayed graphs. As the data sonification technique asks a learning effort before getting used to, it becomes interesting to measure the degree of learning needed to handle data sonification as well as the relationship with the musical experience level of the user. Moreover, the sonification is very subjective since there are potentially an infinity of available data-to-sound mappings.

For all these reasons, an Internet Web site was created where SoundChart and SoundChart3D are presented and can be evaluated by the visitor. The site contains two questionnaires, one for each application, which are divided into two parts. The first series of questions forms a skill test where the ability to understand the sonification proposed is measured. The second part contains a series of more subjective questions, asking the subject his preferences between different MIDI instruments and sonification techniques. The site is available from *http://users.skynet.be/sonification.*

8.2 The questionnaire

The object of the questionnaire is to compare the ability of users to answer questions about data presented in an aural form. Two types of questionnaires have been conceived. The first one concerns the SoundChart application which uses 2D data sonification, while the second one involves the SoundChart3D application and 3D data sonification.

The sonified data are only available in the MP3 sound file format which is a compressed WAV file format. Since both applications use MIDI instruments to sonify the data, the use of WAV files may seem odd. Moreover, while MID files are small (about 2 Kb) and can be downloaded quickly, MP3 files are larger (about 150 Kb) and take a longer time to load. Nevertheless, the use of MP3 files instead of MID files presents a major advantage. Indeed, it is important to remember than MID files contain only instructions for the synthesizer of the sound card and thus, the sound produced when these instructions are treated is hardware-dependent. According to the quality of the sound card, the sound heard may be very different from one user to the other, which raises a major issue for the results analysis.

8.2.1 SoundChart

This test is divided into several parts. The first part collects personal information and musical experience of the subject. The second part provides some pre-requirements for the sonification and the last part is the application test itself.

Personal information

First, subjects are asked to give personal information (name, first name, age, gender, title/position, field of activity and e-mail address) which are used to identify the subject and to validate the answers. Next, the musical experience is evaluated with some questions such as the instrument played by the subject and the practicing period. The subject has also to evaluate his own musical level experience, going from "no experience" to "expert". This information allows to receive some indication on the subject's background and training.

Pre-requirements

Before the test begins, a short explanation about the different sonification techniques used in SoundChart is given in order to get the subject used to auditory graphs. The pre-requirements provide a common basis for understanding the sonification used in the application. As described in the chapter on SoundChart, four different sonification techniques are implemented in SoundChart: pitch-based mapping, beat drums mapping, stereo panning and extreme values detection. Each of them is briefly described and at least one example is given. The examples show a graphical representation of the data, followed by the corresponding sound file.

- `Pitch-based mapping`: Two graphs are represented graphically to illustrate the pitch-based mapping. The first graph draws the equation $Y = X$ and shows the subject a positive slope sonification. The sound heard is a series of notes with an increasing pitch. The second graph, representing the equation $Y = -X + a$, shows a negative slope which is perceived as a series of notes with a linearly decreasing slope.

- `Beat drums mapping`: In order to reveal the difference between two positive or negative slopes, the equations $Y = X$ and $Y = X^2$ are represented visually and acoustically. While the beat drums in the first graph appear at a regular rate, their rhythm increases progressively in the second one.
- `Stereo panning`: One example is illustrated to show the relevance of stereo panning. A common graph is displayed and the corresponding sound is progressively played more by the right speaker and less by the left speaker as the graph is covered.
- `Extreme values detection`: The given example shows how an extreme value is sonified. A graph containing an extreme value is drawn and the corresponding sound indicates when this value occurs.

Once the pre-requirements are assimilated and understood, the subject is ready to answer the SoundChart application test.

The application test

The main goal of the application test is to evaluate the efficiency and the understanding of the sonification techniques performed by SoundChart. However, the test also contains subjective questions where the musical preference of the listener is considered. The "skill" part of the test consists of a set of questions related to authentic time series, coming from Rob Hyndman's time series data library [Hyn]. For each question, the subject must specify the number of times he listened to the signal before answering the question. The results of the test will be analysed in a further section. The whole test is divided into 8 questions which are described below:

1. *The sonification represents the annual sheep population in England and Wales between* 1867 *and* 1939.

    - Were there more sheeps in 1867 than in 1939 ?
    - In your opinion, in about which year did the sheep population reach the minimum ?

    `Description`: Beat drums and stereo mapping are added to enhance the pitch-based sonification. The evolution of the annual sheep population during this period is shown in figure 8.1, where the red line represents the original graph and the blue one the chart of the linear

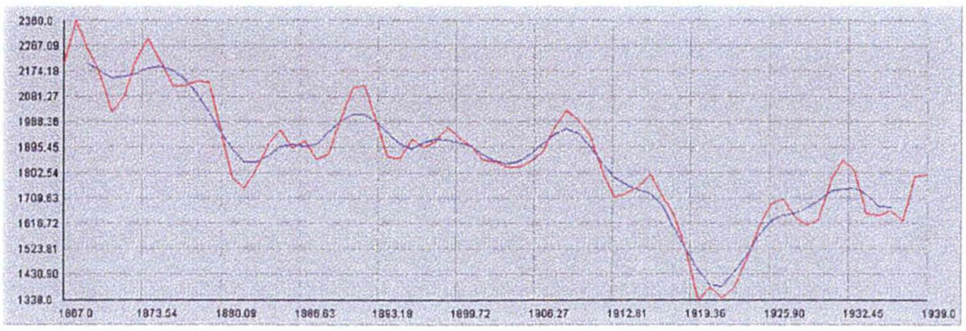


Figure 8.1: The annual sheep population in England and Wales between 1867 and 1939

moving average. The graph shows the correct answers for both questions. It clearly indicates that there were more sheeps in 1867 than in 1939 and the minimum was reached in 1920. While correctly answering these questions with the graph is quite easy, using only sound is not as evident. The results analysis will indicate how the sonification performed by SoundChart allows the perception of a global trend in a graph as well as the localization of a specific point.

2. *The sonification represents the daily morning temperature of an adult woman during two months.*

   - In your opinion, did she have fever during the period ?
   - If yes, during how many days ?

   `Description`: For this question, the sonification includes all the optional sonification techniques: beat drums mapping, stereo panning and extreme values detection. The time series is represented graphically in figure 8.2. The peak in the chart shows clearly that the woman had fever during 2 or 3 days. These days of fever are detected as extreme values during the sonification process and the corresponding alarm sounds are played. The results analysis will show how the extreme values detection is understood by the subject.

3. *The sonification represents the monthly electricity production in Australia between January* 1956 *and August* 1995.

   - Is the electricity production in Australia lower in 1956 than in 1995 ?
   - How would you categorize the evolution of electricity production in Australia ? Linear or exponential ?

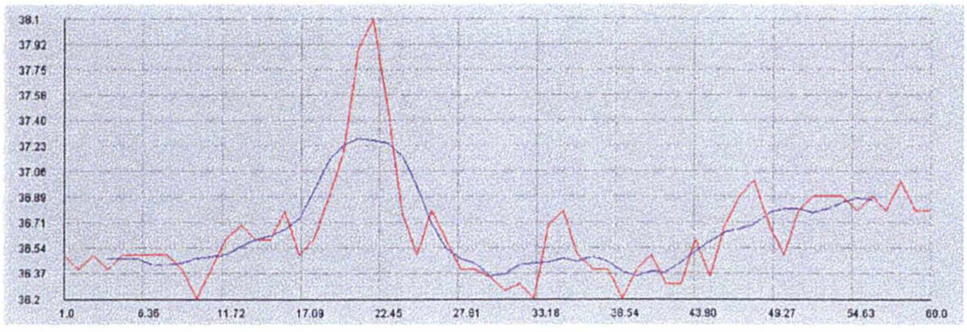


Figure 8.2: The daily morning temperature of an adult woman during two months

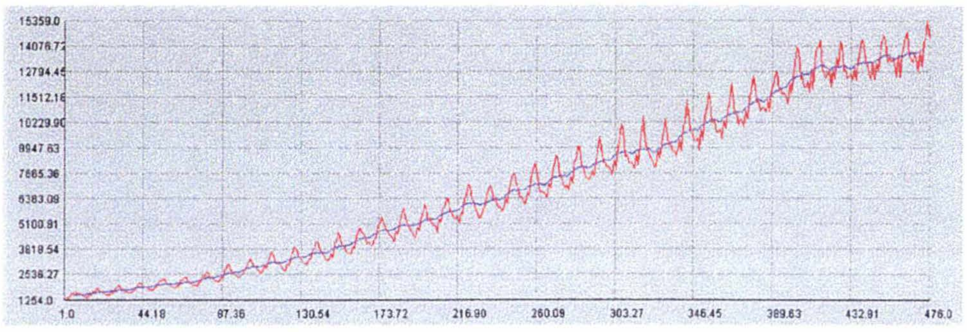


Figure 8.3: The monthly electricity production in Australia between January 1956 and August 1995

- Is the evolution of electricity production in Australia characterized by a seasonal trend ?

`Description`: Only stereo panning is added to the pitch-based sonification. The chart of the time series is represented in figure 8.3. While observing the figure, it appears that the chart is defined by two characteristics. Firstly, as shown by the linear moving average represented by the blue line, the general trend of the chart is linear. Secondly, the evolution of the electricity production is characterized by a seasonal trend, as shown by the peeks situated in a regular way along the chart. The aim of this question is to discover if the seasonal trend is detected by the listener and if the perception of the general trend is affected by this seasonal movement.

4. *The sonification represents the monthly Minneapolis public drunkenness intakes between January* 1966 *and July* 1978 *(*151 *months).*

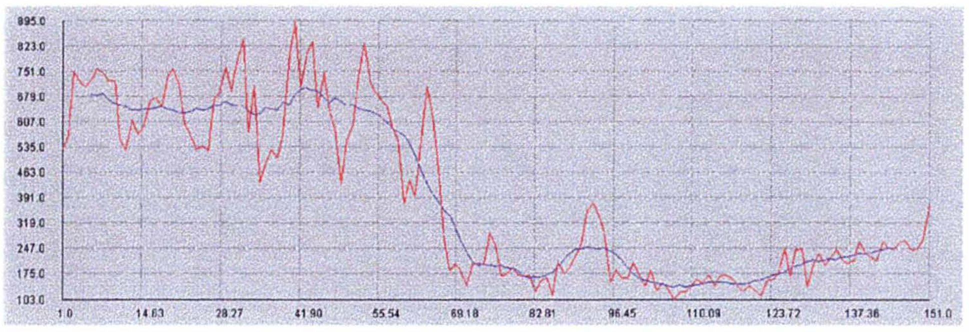


Figure 8.4: The monthly Minneapolis public drunkenness intakes between January 1966 and July 1978 (151 months)

- Were there more intakes in 1966 than in 1978 ?
- Is the evolution of public drunkenness intakes linear ?

`Description`: The time series is only sonified using the pitch-based mapping and is represented by the chart in figure 8.4. The chart illustrates that there were less intakes in 1978 than in 1966. Moreover, the linear moving average shows a break in the evolution of intakes at about the middle of the period. The number of intakes has strongly decreased in a short time, which tends to say that the evolution of public drunkenness intakes in not linear. The result analysis will indicate how the sudden pitch decrease due to the break is perceived and interpreted by the subject.

5. *These sonifications represent the same data but with different MIDI instruments for the pitch mapping. Listen to each of them and give a mark between* 0 *(the worst) and* 10 *(the best). You may give the same mark to different instruments.*

    - Acoustic grand
    - Steel String guitar
    - Violin
    - SynthStrings 2
    - Pan Flute

    `Description`: This subjective question allows to know the subject's preferences about the instrument chosen for the pitch-based sonification. The instruments proposed are very different by nature and belong to a specific MIDI group such as piano, guitar or solo string group.

6. *These sonifications represent the same data but with different MIDI instruments for the beat drums mapping. Listen to each of them and give a mark between* 0 *(the worst) and* 10 *(the best). You may give the same mark to different instruments.*

   - Celesta
   - Slap Bass 1
   - Timpani
   - Tinkle bell
   - Woodblock

   `Description`: This is the same question as the previous one but this time the instruments for the beat drums mapping are evaluated.

7. *Sonification is a mean to convey information using sound. But how would you qualify these different sonification techniques about their utility and their efficiency ?*

   - The pitch mapping
   - The beat drums mapping
   - The stereo mapping
   - The extreme values detection
   - The sonification in general

   `Description`: The four adjectives proposed are: useless, sometimes useful, always useful and essential. For this question, the subject has to give his opinion about these different sonification techniques. The evaluation criteria are the utility of the technique compared to the visual representation as well as its efficiency in relation to the precision of the information transmitted by the sound.

8. *Please tell us what you think about our project, our applications, this web page, or anything else that comes to mind. We welcome any feedback, comments or suggestions.*

   `Descriptionn`: The comments and suggestions collected by the subject may concern the sonification techniques but also some advice or feedback on the application. The most interesting suggestions will be expressed in the results analysis section.

8.2.2 SoundChart3D

The test related to SoundChart3D follows the same layout as the SoundChart test, respectively personal information, pre-requirements and the application test.

Personal information

The same information than in the SoundChart test is collected form the subject: name, first name, age, gender, title/position, field of activity, e-mail address and an evaluation of the musical knowledge.

Pre-requirements

Before the test begins, the subject learns how the sonification of three-dimensional graphs is made in SoundChart3D. The pre-requirements provide a common basis for understanding the sonification used in the application. As described in the chapter on SoundChart3D, the sonification is made by "cutting into slides" the 3D graph in order to decrease the dimensionality of the data. This way, the resulting 2D graphs are travelled and auralized one by one. The pre-requirements introduce and explain the different techniques available in SoundChart3D to perform the sonification: the horizontal, vertical and diagonal travelling, the end of line mapping and the beat drums mapping. Each technique is briefly described and one example is given. The examples show a graphical representation of the data, followed by the corresponding sound file.

- `Horizontal travelling`: A symmetric 3D graph on a 10x10 scene (i.e. a grid of 10 on 10 PolySounds) is represented graphically. The corresponding sonification gets the subject used to the horizontal travelling.

- `Vertical travelling`: The 3D graph illustrated is built on a 15x5 scene. This graph allows to understand the difference between the horizontal and vertical travelling when the graph is not symmetric. Indeed, according to the shape of the 3D graph, the use of horizontal or vertical travellings can radically change the sonification of the graph.

- `Diagonal travelling`: The diagonal travelling is represented by a 3D graph on a 10x10 scene. At the beginning, the sound produced by the sonification of each slide is quite short. But while travelling diagonally through the graph, the length of the auditive signal increases progressively and reaches its maximum at about the middle of the graph. After that, the length of the sound decreases the same way it has increased until the end of the graph.

- `End of line mapping`: One example is given to explain the end of line mapping. A sound is played at the end of each slide during the sonification process, allowing a better distinction of each slide.

- `Beat drums mapping`: The example illustrates the relevance of beat drums mapping. A common 3D graph is displayed and the rhythm of beat drums in the corresponding sonification represents the intensity of the slope in each slide.

Once the pre-requirements are assimilated and understood, the subject is ready to answer the SoundChart3D application test.

The application test

Just like for the SoundChart application, the test contains two parts. The first part evaluates the efficiency and the understanding of the sonification techniques used in SoundChart3D with a series of questions related to fictive 3D data. For each question, the subject must specify the number of times he listened to the sonified data before answering the question. The results of the test will be analysed in a further section. The whole test is divided into 6 questions which are described below:

1. *A 3D graph containing two bumps (i.e. some data with Y values greater than Y values of their neighbourhood and thus looking like a bump) has been sonified. The selected mapping is the vertical travelling and the sonification starts from the bottom right corner. Considering the grid below, that means that the sonification starts from the bottom right case and travels from bottom to top.*

    - If this 3x3 grid represents the graph, where are these two bumps located ?
    - Do they have the same height ?

    `Description`: The 3D graph is sonified using the classical pitch-based sonification and the vertical travelling. The end of line mapping is added to enhance the distinction of each slide. The 3D graph is shown in figure 8.5. The goal of the first question is to correctly place the two bumps by clicking into a grid of 3x3 checkboxes. This grid represents a partition of the scene into 9 parts (north, west, south-west, etc.). The graph quickly shows the correct answers. The two bumps have the same height and are located to the south and the north-west of the scene. While correctly answering these questions with the graph is quite easy, using only sound is pretty difficult and asks a lot of concentration. The results analysis will indicate how the sonification performed by SoundChart3D allows the localization of some specific parts of a graph.

2. *A 3D graph containing two bumps has been sonified. The selected mapping is the horizontal travelling and the sonification starts from*

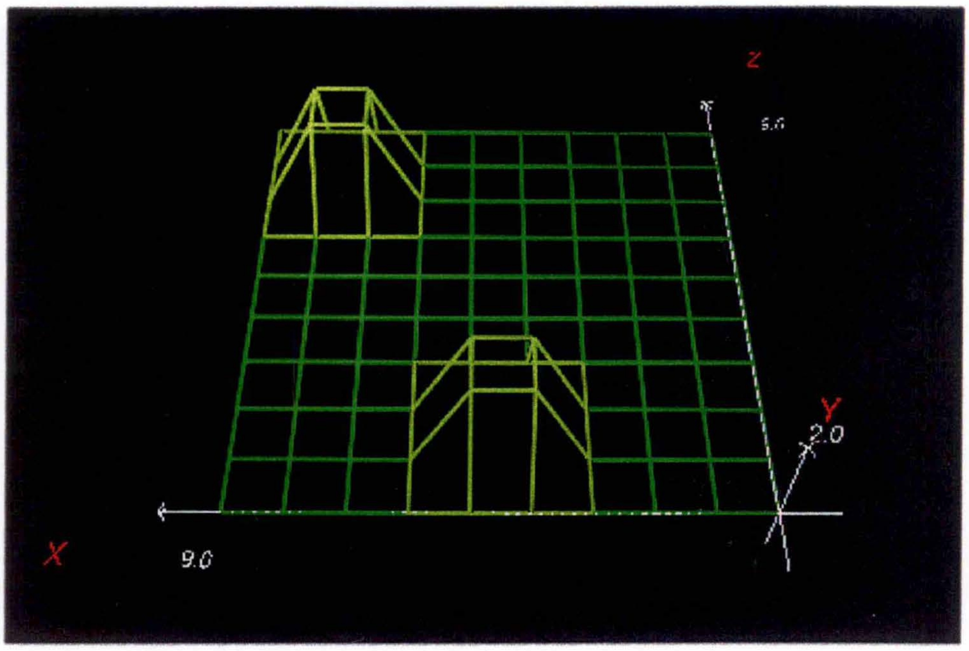


Figure 8.5: The 3D graph related to question 1

the bottom right corner. Considering the grid below, that means that the sonification starts from the bottom right case and travels from right to left.

- If this 3x3 grid represents the graph, where are these two bumps located ?
- Do they have the same height ?

`Description`: Like the previous question, the graph is sonified using the pitch-based sonification and the end of line mapping. However, the sonification is made according to the horizontal travelling. The 3D graph is shown in figure 8.6. The goal is always to find out the location of the two bumps. While observing the figure, it appears that the two bumps have not the same height and are placed to the east and south-east of the scene.

3. *A 3D graph containing two bumps has been sonified. The selected mapping is the diagonal travelling and the sonification starts from the bottom right corner. Considering the grid below, that means that the sonification starts from the bottom right case and travels diagonally from right to left.*

   - If this 3x3 grid represents the graph, where are these two bumps

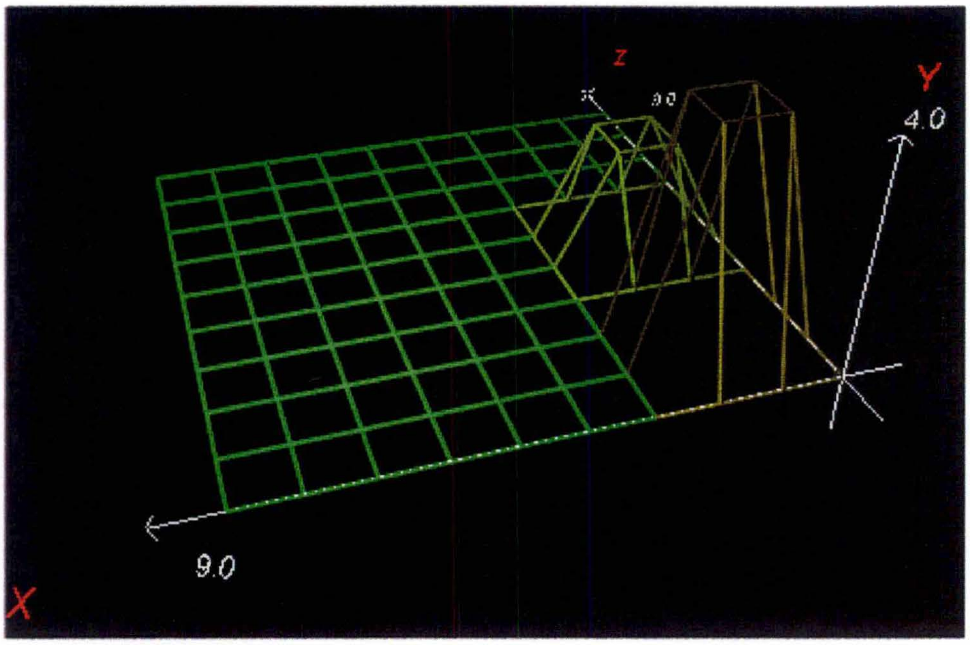


Figure 8.6: The 3D graph related to question 2

located ?

- Do they have the same height ?

`Description`: The pitch-based sonification and the end of line mapping are still used for this question but the graph is covered by the diagonal travelling. The 3D graph is shown in figure 8.7. The two bumps have the same height and are situated to the north-west and south-west of the scene. The results analysis will indicate how the diagonal travelling is perceived and understood by the subject.

4. *Give a mark between* 0 *(the worst) and* 10 *(the best) for each travelling type. You may give the same mark to different travellings.*

   - Vertical travelling
   - Horizontal travelling
   - Diagonal travelling

   `Description`: This subjective question allows to know the subject's preferences about the travelling chosen for the sonification.

5. *Sonification is a mean to convey information using sound. But how would you qualify these different sonification techniques about their utility and their efficiency ?*

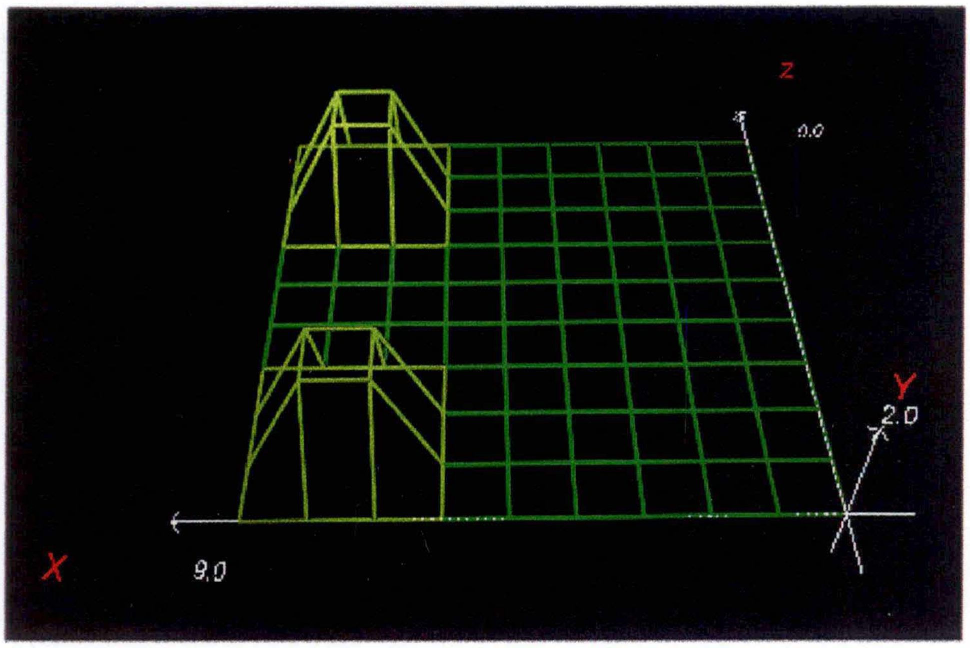


Figure 8.7: The 3D graph related to question 3

- The pitch mapping
- The beat drums mapping
- The end of line notification
- The 3D sonification in general

`Description`: The four adjectives proposed are: useless, sometimes useful, always useful and essential. For this question, the subject has to give his opinion about these different sonification techniques. The evaluation criteria are the utility of the technique compared to the visual representation as well as its efficiency in relation to the precision of the information transmitted by the sound.

6. *Please tell us what you think about our project, our applications, this web page, or anything else that comes to mind. We welcome any feedback, comments or suggestions.*

   `Descriptionn`: The comments and suggestions collected by the subject may concern the sonification techniques but also some advice or feedback on the application. The most interesting suggestions will be expressed in the results analysis section.

8.3 Results analysis

The results analysis will reveal how the sonification techniques proposed by the two applications, SoundChart and SoundChart3D, are perceived and understood. From these results, conclusions will be drawn to determine how the sonification of two-dimensional and three-dimensional data can support or even replace visually displayed graphs.

8.3.1 The sample

The sample represents the people who have answered questionnaires related to SoundChart or SoundChart3D. As some individuals have only answered the SoundChart questionnaire, the number of answers is not the same for both applications. Indeed, 23 answers were collected for the SoundChart test and 18 answers for the test related to SoundChart3D. It is important to note that only data from subjects who answered all questions in each test have been used.

The first stage in the results analysis is to study some characteristics of the sample, namely the field of activity and the musical knowledge of the subjects. For this analysis, data from the 23 persons are taken into account.

Cross table 8.1 below represents, in rows, the proportion of people working in different fields of activity and, in columns, their musical experience level.

	None	Novice	Average	Competent	Expert	Totals
Computer	6 (27 %)	2 (9 %)	1 (4 %)	2 (9 %)	0 (0 %)	11 (49 %)
Economics	4 (18 %)	2 (9 %)	0 (0 %)	0 (0 %)	0 (0 %)	6 (27 %)
Sciences	1 (4 %)	0 (0 %)	1 (4 %)	0 (0 %)	1 (4 %)	3 (12 %)
Health	0 (0 %)	0 (0 %)	1 (4 %)	0 (0 %)	0 (0 %)	1 (4 %)
Logistics	1 (4 %)	0 (0 %)	0 (0 %)	0 (0 %)	0 (0 %)	1 (4 %)
Others	1 (4 %)	0 (0 %)	0 (0 %)	0 (0 %)	0 (0 %)	1 (4 %)
Totals	13 (57 %)	4 (18 %)	3 (12 %)	2 (9 %)	1 (4 %)	23 (100 %)

Table 8.1: The sample population ranged by field of activity and musical experience level

The majority of the sample characterizes people working in the computer science area and having a limited musical experience or no experience at all. Further sections will consider the relationship between these two factors and the final results.

8.3.2 SoundChart

For each question, the proportion of correct and wrong answers is given, as well as the average number of hearings before answering. A short analysis comments the results.

1. *The sonification represents the annual sheep population in England and Wales between* 1867 *and* 1939.

   - Were there more sheeps in 1867 than in 1939 ? **Answer : Yes.**

Correct answers	Wrong answers	No idea
17 (74 %)	5 (22 %)	1 (4 %)

   - In your opinion, in about which year did the sheep population reach the minimum ? **Answer : 1920.**

Answers	Number of answers
Before 1909	2 (8 %)
Between 1909 and 1919	5 (22 %)
1920	3 (13 %)
Between 1921 and 1931	9 (39 %)
After 1931	4 (18 %)

   - Average number of hearings before answering : 3.

`Analysis:` The 74 per cent of correct answers for the first question reveal that the sonification performed by SoundChart allows quite efficiently to perceive a global trend in the series. The second question is a little more tricky as it asks to identify a precise point in the time series. The impossibility to clearly isolate temporal units makes it very difficult to locate a precise year in the series. While the correct answer is 1920, answers given in a 10 years bracket remain satisfactory, which leads to another 74 per cent of more or less correct answers. In average, subjects have listened 3 times to the sonification before answering, which seems quite normal.

2. *The sonification represents the daily morning temperature of an adult woman during two months.*

   - In your opinion, did she have fever during the period ? **Answer : Yes.**

Correct answers	Wrong answers	No idea
23 (100 %)	0 (0 %)	0 (0 %)

- If yes, during how many days ? **Answer : 3**.

Answers	Number of answers
Less than 2	2 (8 %)
2	6 (27 %)
3	5 (22 %)
4	2 (8 %)
More than 4	8 (35 %)

- Average number of hearings before answering : 2.

`Analysis`: This question allows to know how extreme values detection is understood by the subjects. The perfect score for the first question shows that the sound played to sonify extreme values is clearly recognized and interpreted. However, the results for the second question are quite surprising. While 2, 3, or 4 days of fever may be considered as correct answers and represent about half of the answers, the 35 per cent of answers for more than 4 days remain astonishing. It seems that some subjects do not associate each alarm sound to one day of fever, and pay more attention to the pitch mapping when answering the question. The average number of hearings before answering is 2, probably one audition for each question.

3. *The sonification represents the monthly electricity production in Australia between January* 1956 *and August* 1995.

   - Is the electricity production in Australia lower in 1956 than in 1995 ? **Answer : Yes**.

Correct answers	Wrong answers	No idea
22 (96 %)	1 (4 %)	0 (0 %)

   - How would you categorize the evolution of electricity production in Australia ? Linear or exponential ? **Answer : Linear**.

Correct answers	Wrong answers	No idea
12 (52 %)	11 (48 %)	0 (0 %)

   - Is the evolution of electricity production in Australia characterized by a seasonal trend ? **Answer : Yes**.

Correct answers	Wrong answers	No idea
16 (70 %)	6 (26 %)	1 (4 %)

- Average number of hearings before answering : 2.

`Analysis`: The aim of this question is to discover if the seasonal trend is detected by the listener and if the perception of the general trend is affected by this seasonal movement. The excellent results for the first question confirm that the global evolution of the time series is correctly perceived by the subjects during the sonification. However, the second question presents mixed results, where there are almost as much correct answers as wrong answers. Thus, it seems that the only use of pitch-based mapping is sufficient to transmit the evolution of the series, but not enough to correctly characterize this evolution. This conclusion shows the relevance of the beat drums mapping to transmit acoustical changes in the slope. Indeed, only the stereo panning was added to the pitch-based sonification and it can be stated that the use of the beat drums mapping would most probably lead to better results for this question. However, this lack of beat drums mapping allows to clearly perceive the pitch-based mapping, without being overloaded by additional sounds. Probably due to this fact, results for the third question are quite good, even if the seasonal trend was hard to detect. In average, subjects have listened 2 times to the sonification before answering, which is quite low considering the three questions proposed.

4. *The sonification represents the monthly Minneapolis public drunkenness intakes between January* 1966 *and July* 1978 *(*151 *months).*

   - Were there more intakes in 1966 than in 1978 ? **Answer : Yes.**

Correct answers	Wrong answers	No idea
20 (87 %)	2 (9 %)	1 (4 %)

   - Is the evolution of public drunkenness intakes linear ? **Answer : No.**

Correct answers	Wrong answers	No idea
19 (83 %)	3 (13 %)	1 (4 %)

   - Average number of hearings before answering : 2.

`Analysis`: The results for the first question are not surprising and meet the previous results about the detection of a global trend in a time series. 87 per cent of the subjects have correctly answered the question. For the second question, the results show that 83 per cent of the sample have detected the sudden pitch decrease due to the break in the evolution of intakes at about the middle of the period. The average number of hearings before answering is 2, probably one audition for each question.

5. *These sonifications represent the same data but with different MIDI instruments for the pitch mapping. Listen to each of them and give a mark between* 0 *(the worst) and* 10 *(the best). You may give the same mark to different instruments.*

    - The table below represents the average score (on 10) given by the sample population to each MIDI instrument.

MIDI Instrument	Average score
Acoustic grand	7.5
Steel String guitar	7.0
Violin	7.3
SynthStrings 2	5.5
Pan Flute	6.5

Analysis: The results give the subject's preferences about the instrument chosen for the pitch-based sonification. The Acoustic grand, Steel String guitar and Violin are commonly preferred by the sample. Surprisingly, the SynthStrings 2 has received the worst score despite the fact it was employed in every sonification during the test.

6. *These sonifications represent the same data but with different MIDI instruments for the beat drums mapping. Listen to each of them and give a mark between* 0 *(the worst) and* 10 *(the best). You may give the same mark to different instruments.*

    - The table below represents the average score (on 10) given by the sample population to each MIDI instrument.

MIDI Instrument	Average score
Celesta	5.0
Slap Bass 1	5.5
Timpani	6.1
Tinkle bell 2	6.3
Woodblock	6.5

`Analysis`: The results give the subject's preferences about the instrument chosen for the beat drums sonification. The Woodblock instrument retains the preference of the sample, followed by the Tinkle bell and Timpani instrument. This last one was used in every beat drums mapping during the questionnaires.

7. *Sonification is a mean to convey information using sound. But how would you qualify these different sonification techniques about their utility and their efficiency ?*

   Figure 8.8 below represents the global appreciation of each sonification technique.

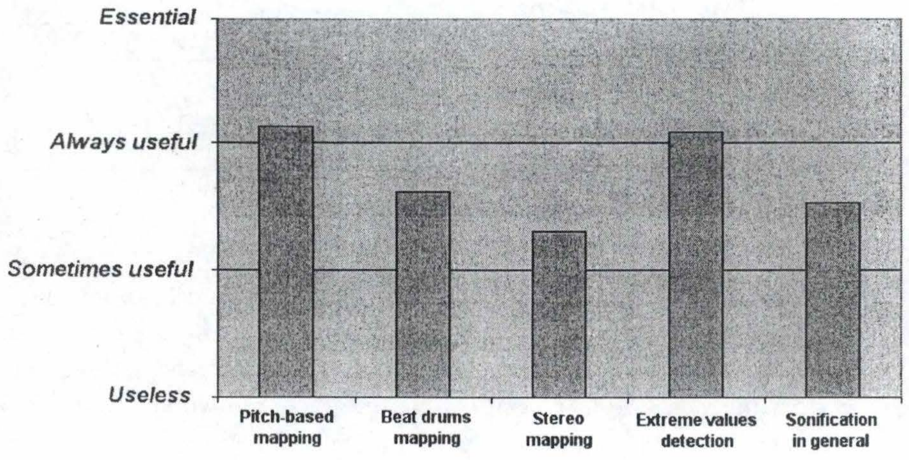


Figure 8.8: Average appreciation of each 2D sonification technique

   As pitch-based mapping is the heart of time-dependent data sonification, it is normal that subjects consider this technique as always useful. Beat drums mapping and stereo mapping remain additional features to enhance the pitch-based mapping and are considered as sometimes useful. However, most of the subjects regard the extreme values detection as an important and always useful feature of the sonification.

8. *Please tell us what you think about our project, our applications, this web page, or anything else that comes to mind. We welcome any feedback, comments or suggestions.*

   The most interesting suggestions collected by the subjects are picked up below with a brief description of their author:

- Great job! It's clear and pretty well explained. The idea of the stereo mapping according to the length of the graph is very useful. I was wondering : instead of using the stereo mapping to represent the length, maybe it could be possible to use the stereo to represent two different sets of data at the same time ? [*Nicolas Vanderavero, student in computer science*]
- Interesting study. Might be useful in the future or for less developed people (blinds) or high tech developments (Formula One). [*Pierre de Somer, cotton trading employee*]
- It might be useful for people who would like to use it frequently and for evaluating the same kind of data. Otherwise you could never use it as sole guidance because you wouldn't know if the data are important with big changes or small with only little changes. But I think it works quite well. [*Nicolas Schöller, economist*]
- What is possible to do with sounds, especially as a translation of visual information, has astonished me. [*Aurelia De Pauw, student in biology*]
- These tests allowed me to confirm that my memory is more visual than auditive. Indeed, while a graph allows me to quickly answer a question, the only use of sonification asks me 2 or 3 hearings before answering. [*Quentin Dallons, student in computer science*]
- I find instruments with marked variations (like the piano), where each note is clearly perceived, more suited to determine values variations than instruments with continue variations (like the violin), where the passage between notes is less marked. In my opinion, these marked variations allow to better perceive the variations in sound frequency, and thus to determine easier the general trend of the symbolized curves. [*Alain Dallons, pharmacist*]

8.3.3 SoundChart3D

1. *A 3D graph containing two bumps (i.e. some data with Y values greater than Y values of their neighbourhood and thus looking like a bump) has been sonified. The mapping selected is the vertical travelling and the sonification starts from the bottom right corner. Considering the grid below, that means that the sonification starts from the bottom right case and travels from bottom to top.*

    - If this 3x3 grid represents the graph, where are these two bumps located ? **Answer : North-West and South.**

Answers	Number of answers
Both bumps correctly placed	4 (22 %)
One bump correctly placed	6 (33 %)
No bump correctly placed	8 (45 %)

- Do they have the same height ? **Answer : Yes**.

Correct answers	Wrong answers	No idea
11 (62 %)	4 (22 %)	3 (16 %)

- Average number of hearings before answering : 4.

`Analysis`: The results for the first question clearly indicate that 3D sonification performed by SoundChart3D is not as evident as 2D sonification. The results contain a majority of wrong answers, where no single bump has been placed correctly. Only 22 per cent of the sample have succeeded in correctly placing the two bumps on the grid, and one third of the sample has found one of both bumps. 3D sonification requests a lot of concentration and numerous playbacks of the sound file become necessary in order to detect the bumps. However, SoundChart3D seems quite efficient to communicate height values, since 62 per cent of the sample have correctly compared the height of the two bumps. In average, subjects has listened 4 times to the sonification before answering, which seems to be the minimum to answer all questions.

2. *A 3D graph containing two bumps has been sonified. The mapping selected is the horizontal travelling and the sonification starts from the bottom right corner. Considering the grid below, that means that the sonification starts from the bottom right case and travels from right to left.*

   - If this 3x3 grid represents the graph, where are these two bumps located ? **Answer : East and South-East**.

Answers	Number of answers
Both bumps correctly placed	4 (22 %)
One bump correctly placed	6 (33 %)
No bump correctly placed	8 (45 %)

   - Do they have the same height ? **Answer : No**.

Correct answers	Wrong answers	No idea
12 (67 %)	33 (0 %)	0 (0 %)

- Average number of hearings before answering : 3.

`Analysis:` The results are almost the same as for the previous question and always contain a majority of wrong answers for the first question. The vertical travelling is as tricky as the horizontal travelling. Once again, comparing the bump's heights gives better results with 67 per cent of correct answers. The average number of hearings before answering is 3, probably two auditions to detect both bumps and one audition for the second question.

3. *A 3D graph containing two bumps has been sonified. The mapping selected is the diagonal travelling and the sonification starts from the bottom right corner. Considering the grid below, that means that the sonification starts from the bottom right case and travels diagonally from right to left.*

   - If this 3x3 grid represents the graph, where are these two bumps located ? **Answer : North-West and South-West.**

Answers	Number of answers
Both bumps correctly placed	1 (6 %)
One bump correctly placed	11 (61 %)
No bump correctly placed	6 (33 %)

   - Do they have the same height ? **Answer : Yes.**

Correct answers	Wrong answers	No idea
10 (56 %)	6 (33 %)	2 (11 %)

   - Average number of hearings before answering : 3.

`Analysis:` The diagonal travelling used for this question radically changes the sonification of the 3D chart. This difference is found in the results where this time 61 per cent of the sample has successfully placed one bump on the grid. However, only 1 subject has placed both bumps correctly, which indicates the difficulty of the diagonal travelling. Indeed, this travelling method is less natural than the horizontal and vertical travelling and asks more concentration and training before getting used to. As usual, the majority of the subjects, 56 per cent, have correctly compared the height of the two proposed bumps.

4. *Give a mark between* 0 *(the worst) and* 10 *(the best) for each travelling type. You may give the same mark to different travellings.*

- The table below represents the average score (on 10) given by the sample population to each travelling type.

Travelling type	Average score
Vertical travelling	5.9
Horizontal travelling	5.6
Diagonal travelling	4.6

`Analysis`: The results give the subject's preferences about the travelling type chosen for the sonification. The scores are not very high and reflect the difficulty of 3D sonification in general. The diagonal travelling is considered by the sample to be the least efficient of the three methods.

5. *Sonification is a mean to convey information using sound. But how would you qualify these different sonification techniques about their utility and their efficiency ?*

   Figure 8.9 represents the global appreciation of each sonification technique.

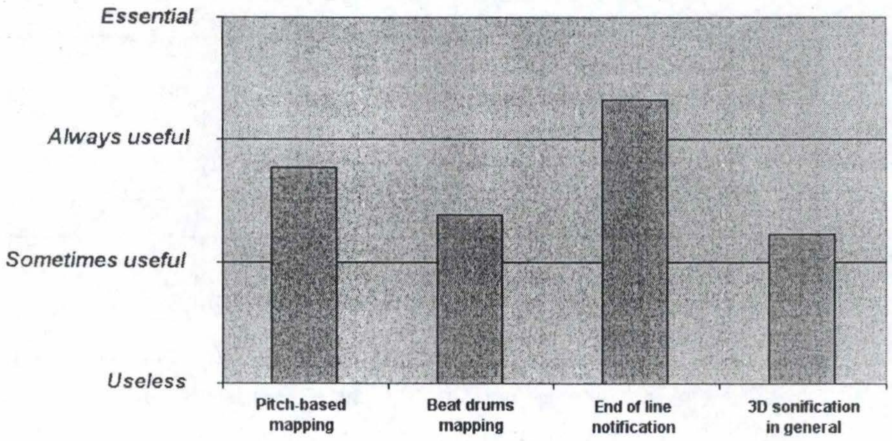


Figure 8.9: Average appreciation of each 3D sonification technique

Like in the 2D sonification, the pitch-based mapping is the main technique of 3D data sonification. However, subjects consider this technique as less useful than 2D sonification, which reflects the difficulty and relative inefficiency of the method. Beat drums mapping is still considered as sometimes useful. However, most of the subjects regard the end of line notification as the most important technique in 3D sonification. This result comes from the fact that end of line sonification is an essential feature to identify each slide during the travelling.

6. *Please tell us what you think about our project, our applications, this web page, or anything else that comes to mind. We welcome any feedback, comments or suggestions.*

   The most interesting suggestions collected by the subjects are picked up below with a brief description of their author:

   - It is very difficult to map the data trends to the sound. One needs a large concentration to be able to determine the tendency of the data. [*Adolphe Nahimana, researcher in computer science*]
   - Well, 3D sonification is a little more tricky than 2D sonification. The end of line notification is an essential feature, especially in the diagonal travelling. Without this, I would be totally lost in the graph. 3D requires more attention to hear than 2D sonification. When I listen to a 2D sonification, I don't need to do any effort to know what the graph looks like. But with 3D sonification I must concentrate only on the sound and I try to "recreate" the original graph in my mind. So, I think that 3D sonification needs too much concentration to be used when the user is doing another task. It is not something you "just listen to" without paying too much attention to it. For this reason, I don't like the beat drums mapping : it's too complex to handle so many information at the same time. [*Nicolas Vanderavero, student in computer science*]
   - The use of 3D is more difficult to represent mentally. It would require a longer time of learning. The choice of one instrument for the X, another one for the Y and a third for the Z could be considered. [*Alain Dallons, pharmacist*]
   - I don't think that even for trained users of the sonification, it is possible to concentrate on three different types of sound mixed together. Certainly not when you use the sonification as a tool for gaining time. [*Nicolas Schöller, economist*]
   - It could be useful to choose a different sound to represent the "end-of-line" of the middle diagonal to locate it better. [*Xavier Martin, student in computer science*]

8.3.4 Musical skill influence

An interesting relationship to study is the relation between results performed by the subjects and their musical experience level. It would be quite natural to think that users with some musical background have some facilities to detect variations in music, and thus are able to handle more quickly and easily the sonification techniques proposed in the test. In order to find out the existence of such a relationship, results from both questionnaires will be divided into two different groups, namely the subjects with no musical experience and subjects with musical background.

SoundChart

The table below represents the average results for the questionnaire related to SoundChart according to the musical level. The musical background sample contains subjects with a novice, competent, average and expert musical experience level. The SoundChart test contained 9 questions and each question has the same importance. To ease the interpretation, average results have been written in percentages.

Musical level of the subjects	Average score
Musical background	76 %
No experience	66 %

As it was supposed, subjects with musical experience perform, in average, better than subjects with no experience at all. However, an average score of 66 per cent, instead of the 76 per cent for trained subjects, proves that the sonification of time-dependent data remains a valuable tool accessible to any individual, with or without preliminary musical experience.

SoundChart3D

The SoundChart3D test also contained 9 questions with the same importance. The table below shows the results in percentages.

Musical level of the subjects	Average score
Musical background	40 %
No experience	42 %

As sonification of three-dimensional data is quite difficult and requires a lot of concentration, the musical experience of subjects is no longer an advantage. Both scores are almost equal and reveal the inefficiency of the techniques performed by SoundChart3D.

8.3.5 Field of activity influence

A second factor which is worth to study is the influence of the field of activity on the results performed by the subjects. Notwithstanding the main part of the subjects working in the computer science area, it would be quite natural to consider that it is not significant for the results as answering both questionnaires is not eased by a specific knowledge in this area. In order to find out the existence of such a relationship, results from both questionnaires will be divided into two different groups, namely the subjects related to the computer science field of activity and subjects from others areas.

SoundChart

The table below represents the average results for the questionnaire related to SoundChart according to the field of activity. Each question has the same importance and average results have been written in percentages.

Field of activity of the subjects	Average score
Computer science	72 %
Others	69 %

As it was supposed, subjects from the computer science field of activity do not perform better than subjects from others fields. Both scores are quite good and prove that the sonification of time-dependent data is accessible to individuals from any field of activity.

SoundChart3D

The table below shows the results in percentages for the SoundChart3D test.

Field of activity of the subjects	Average score
Computer science	41 %
Others	40 %

Understanding the sonification techniques of SoundChart3D is not eased by some experience or knowledge in the computer science area. Both scores show once again the difficulty to handle the techniques performed by the application.

8.4 Conclusion

Observational studies cannot provide conclusive answers about the efficiency of data sonification since the population can differ in significant ways. Moreover, the present sample was too small to draw any decisive conclusions about the experimented sonification. However, this study provides important clues which are worthy of analysis.

Basically, the results obtained during the experimentation shows that sound can be very effective as a medium for presenting information in addition to visually displayed data.

The SoundChart test revealed that the sonification of two-dimensional time-dependent data can give excellent results, whatever the musical training of the user. The auditory feedback provided by SoundChart is a valuable tool for supporting or even replacing the visual chart when analysing time series.

Considering the results from the SoundChart3D test, it can be stated that handling the sonification of three-dimensional data performed by SoundChart3D is a difficult task, maybe because even graphically displayed 3D graphs are harder to understand than 2D ones. The evaluation of this application suggests that the approach is viable, but that it is difficult to use and requires some intensive concentration, preventing the user from doing another task while listening to the sound. Due to this fact, SoundChart3D can not efficiently support the visualization and is certainly not an alternative to it. However, 3D sonification could remain a useful technique for visually impaired individuals.

Chapter 9

Conclusion

The objective of the dissertation, as stated in the introduction, was to study in which way sound could represent time-dependent data. In this recent and relatively unexplored field of research called sonification, we have found, implemented and evaluated several techniques throughout this report to translate time-dependent data into sound.

In the first chapter, we showed how sound properties could offer some new and interesting ways of communicating and interacting with the computer. The numerous advantages proper to sound, despite some drawbacks, yielded to consider the sonification as a first-class technique when presenting the information to a different modality than the usual visualization. Moreover, the plurality and the diversity of existing implementations presented in the chapter allowed to realize the many possibilities offered by sonification.

However, while sonification could be applied in a quite straightforward way to any data set, the particular context of time-dependent data makes some preliminary preprocessing of the data necessary. Thus, we described how statistical methods could pave the way for further sonification, by filtering undesired values from the data set and removing the short term fluctuations that would obscure the sonification.

Once the time-dependent data ready for the sonification process, we got to the heart of the matter by presenting the most widely used sonification technique for time-dependent data, namely the pitch-based mapping. Starting form this technique, we searched and found more complex and original mappings to improve the sonification, such as the slope indicator and the stereo panning.

Before starting to implement our findings about sonification techniques, the choice of the right sound format was an important issue. While sampled sounds, such as WAV files, fit better to sonifications that require natural or special sound effects, the MIDI sound format is more suited to time-dependent data sonification, notably because of the ease to create or modify MID files and the support provided by the Java Sound API.

Two Java applications were developed to illustrate the sonification of time-dependent data. Our first application, SoundChart, provides tools for the sonification of two-dimensional time-dependent data while our second, SoundChart3D, is able to perform a sonic representation of three-dimensional data. Thanks to the numerous tools available, both applications allowed us to search and find out the right techniques and parameters in order to sonify time-dependent data in the best possible way.

Developing two sonification applications was interesting but not sufficient. Indeed, sonification is not necessarily straightforward or intuitive and asks a learning effort before getting used to. Moreover, as sonification makes use of music features, the choice of the right parameters or instruments remains very subjective and depends on musical preferences of the user. For all these reasons, we created an Internet Web site where SoundChart and SoundChart3D were presented and can be evaluated by the visitor. The results analysis allowed us to measure the degree of learning needed to handle data sonification as well as the relationship with the musical experience level of the user.

The results obtained during the experimentation support the conclusions drawn form previous studies, stating that sound can be very effective as a medium for presenting information in addition to visually displayed data.

In the particular context of time-dependent data sonification, while the results are encouraging and suggest that auditory feedback of two-dimensional data may improve or even substitute the information provided by a visual chart, the results collected from the SoundChart3D questionnaire reveal that three-dimensional data sonification is not an evident thing. 3D sonification is more tricky and requires more attention than 2D sonification. Consequently, it becomes impossible to do another task while listening to the sound, which suppresses one of the major benefits of data sonification. Due to the high level of concentration needed, it can be stated that the 3D sonification, as performed by SoundChart3D, can not efficiently support the visualization and certainly not replace it, but it could be useful for visually impaired individuals.

The sonification of time-dependent data, especially two-dimensional data, finds many application fields. Globally, sound can be a used as a graphics replacement wherever visual information cannot be used. For example, the automobile sector could use sonification to inform drivers about the status of their car. However, even if a display unit is present, sound can represent a rich addition. For instance, whenever an interface suffers from graphical overload, sound could be added to diminish its complexity. The mobile phones sector represents a good example, since the screen on a mobile phone is usually too small to display large amounts of information.

Naturally, there are some purposes where our sonification techniques are not totally adequate. Users can acquire and understand the general trend of a time series but if they need to know a precise value at a given time, non-speech audio may not provide the information they need.

In conclusion, sonification remains an interesting and promising field of research. The sonification applied to time-dependent data offers a new perspective on how to treat and analyse data, and the possibilities of mapping techniques seem only limited by the creativity.

Bibliography

[Anr99] Koen Anrijs, *The use of sound in 3d representations of symbolic objects*, Master's thesis, Facultés Universitaires Notre-Dame de la Paix, Namur, 1999.

[ARV97] James L. Alty, Dimitrios Rigas, and Paul Vickers, *Using music as a communication medium*, Proceedings of CHI'97, 1997.

[BCS$^+$00] Maria Barra, Tania Cillo, Antonio De Santis, Umberto Ferraro Petrillo, Alberto Negro, and Vittorio Scarano, *Webmelody: Sonification of web servers*, Dipartimento di Informatica ed Applicazioni, Università di Salerno, Baronissi (Salerno) 84081 - Italy, 2000, Available from http://isis.dia.unisa.it/SONIFICATION/.

[Bur98] Shaun Burke, *Missing values, outliers, robust statistics and non-parametric methods*, VAM Bulletin (1998), no. 19, 22–27.

[BWE93] Stephen A. Brewster, Peter C. Wright, and Alistair D. N. Edwards, *An evaluation of earcons for use in auditory human-computer interfaces*, Proceedings of InterCHI'93, 1993.

[CBL95] Stéphane Conversy and Michel Beaudouin-Lafon, *Le son dans les applications interactives*, Laboratoire de Recherche en Informatique, Université de Paris-Sud, 1995.

[Com98] *Compton's encyclopedia online v3.0*, The Learning Company, Inc., 1998, Available from http://www.comptons.com/encyclopedia.

[Edw89] Alistair D. N. Edwards, *Soundtrack: An auditory interface for blind users*, 1989.

[EM97] Valerie J. Easton and John H. McColl, *Statistics glossary*, 1997, Available from http://www.stats.gla.ac.uk/steps/glossary/time_series.html.

[FBT96] John H. Flowers, Dion C. Buhman, and Kimberly D. Turnage, *Data sonification from the desktop: Should sound be part of standard data analysis software?*, The Proceedings of ICAD'96 International Conference on Auditory Display, Department of Psychology, University of Nebraska, Lincoln, 1996.

[Gav89] William W. Gaver, *The sonicfinder: An interface that uses auditory icons*, In Proceedings of CHI'89 Conference on Human Factors in Computing Systems, volume 4, pages 67-94, 1989.

[Hen01] Tom Henderson, *The physics classroom, lesson 2: Sound properties and their perception*, 1996-2001, Available from http://www.glenbrook.k12.il.us/gbssci/phys/Class/BBoard.html.

[Her99] Thomas Hermann, *Data exploration by sonification*, 1999, Available from http://www.techfak.uni- bielefeld.de/techfak/ags/ni/projects/datamining/datamin_e.htm.

[HHR01] T. Hermann, M. H. Hansen, and H. Ritter, *Sonification of markov chain monte carlo simulations*, Proceedings of the ICAD 2001 International Conference on Auditory Display, 2001, Available from http://cm.bell-labs.com/who/cocteau/papers/pdf/mcmc.pdf.

[Hob00] John Maxwell Hobbs, *Introduction to the java sound api*, EarthWeb Networking and Communication, 1999-2000, Available from http://softwaredev.earthweb.com/java/sdjop/archives/.

[Hyn] Rob Hyndman, *Time series data library*, Available from http://www-personal.buseco.monash.edu.au/ hyndman/TSDL/index.htm.

[Jav99] *Getting started with the java 3d api*, Sun Microsystems, 1999.

[Jav00a] *Java 2 sdk, standard edition documentation*, Sun Microsystems, 1995-2000.

[Jav00b] *Javasound api programmer's guide*, Sun Microsystems, 2000.

[Kra94] Gregory Kramer, *An introduction to auditory display*, Auditory Display: Sonification, Audification, and Auditory Interfaces, Santa Fe Institute Studies in the Sciences of Complexity, 1994.

[Lip89] Eric Lipscomb, *How much for just the midi*, North Texas Computing Center Newsletter (1989), Available from http://www.harmony-central.com/MIDI/Doc/intro.html.

[MR95] Tara M. Madhyastha and Daniel A. Reed, *Data sonification: Do you see what I hear?*, IEEE Software **12** (1995), no. 2, 45–56, Available from http://citeseer.nj.nec.com/madhyastha95data.html.

[NF00] Monique Noirhomme-Fraiture, *Multimedia support for complex multidimensional data mining*, Proceedings of the International Workshop on Multimedia Data Mining (MDM/KDD'2000), in conjunction with ACM SIGKDD conference, Boston, USA, 2000.

[PB01] Bryan Pardo and William P. Birmingham, *The chordal analysis of tonal music*, Electrical Engineering and Computer Science Department, University of Michigan, 2001, Available from http://www-personal.umich.edu/ bryanp/.

[Sah98] Steven C. Sahyun, *Auditory vs. visual graph test*, Physics Department, Oregon State University, 1998, Available from http://www.physics.orst.edu/ sahyun/survey/.

[Sah99] ———, *A comparison of auditory and visual graphs for use in physics and mathematics*, Ph.D. thesis, Oregon State University, 1999, Available from http://www.physics.orst.edu/ sahyun/thesis/.

[Sch97] Wolfgang W. Scherer, *Mid versus wav & ra, which sound files to use in web pages*, midi-LOOPs, W-Music & Arts, 1997, Available from http://www.midiloops.com/midvswav.htm.

[SSC99] Russell D. Shilling and Barbara Shinn-Cunningham, *Virtual auditory displays*, Handbook of Virtual Environment Technology, 1999, Available from http://vehand.engr.ucf.edu/handbook/Chapters/VECHAPTER4a.html.

[Sta01] *Electronic statistics textbook*, 2001, Available from http://www.statsoft.com/textbook/stathome.html.

[Une00] *Unesco training of computer specialists, trainers and users*, 1999-2000, Available from http://203.162.7.85/unescocourse/index.htm.

[VA96] Paul Vickers and James L. Alty, *Caitlin: A musical program auralisation tool to assist novice programmers with debugging*, Proceedings of ICAD'96, 1996.

[Vic99] Paul Vickers, *Caitlin: Implementation of a musical program auralisation system to study the effects on debugging tasks as performed by novice pascal programmers*, Ph.D. thesis, Loughborough University, Dept. of Computer Science, 1999, Available from http://www.cms.livjm.ac.uk/caitlin/.

[Wan00] Yong Wang, *Physiology time series analysis*, Bioinformatics Research Center, Medical College of Wisconsin, 2000, Available from http://brc.mcw.edu/PTs/.

[Wil96] Catherine M. Wilson, *Listen: A data sonification toolkit*, Master's thesis, University of California, Santa Cruz, 1996.

Appendix A

SoundChart3D file format

A.1 File structure

SoundChart3D uses its own file format for storing three-dimensional graphs. No specific extension is needed, as long as the file structure is valid. A valid SoundChart3D file contains some header information before the actual 3D values. The first information is the text string "SOUNDCHART 3D file", followed by the "BEGIN" string. The end of a SoundChart3D file is marked by a corresponding "END" string. The header also contains the information needed to construct the graph base, namely the number of rows, the number of columns and the initial height (in that order). The number of rows and columns must be integer values, but the initial height can be a floating point number.

Next follow the actual 3D data. Each line in the file represents the height (i.e. the y-value) of the four points composing a single PolySound, separated by a special character "|". The end of a line is marked with the same special character. The four values are floating point numbers. Knowing the number of rows and columns, SoundChart3D can compute the number of PolySounds that will compose the graph, thus knowing how many lines the file should contain. For example, a graph containing 10 rows and 5 columns requires (5 x 10) = 50 PolySounds.

A.2 Example

The following file represents a valid SoundChart3D file. The graph contains 15 rows, 5 columns and has an initial height of 2. Thus the file has (15 x 5) = 75 lines of data, each one representing a single PolySound. Since most of the PolySounds share the same extremities, several lines contain the same data (i.e. they represent the same three-dimensional value).

```
SOUNDCHART 3D file
BEGIN
15
5
2
2.0|2.0|2.0|2.0|
1.0|1.0|2.0|2.0|
1.0|1.0|1.0|1.0|
2.0|2.0|1.0|1.0|
2.0|2.0|2.0|2.0|
2.0|2.0|2.0|2.0|
1.0|2.0|2.0|2.0|
1.0|2.0|2.0|1.0|
2.0|2.0|2.0|1.0|
2.0|2.0|2.0|2.0|
2.0|2.0|2.0|2.0|
2.0|3.0|2.0|2.0|
2.0|3.0|3.0|2.0|
2.0|2.0|3.0|2.0|
2.0|2.0|2.0|2.0|
2.0|2.0|2.0|2.0|
3.0|3.0|2.0|2.0|
3.0|3.0|3.0|3.0|
2.0|2.0|3.0|3.0|
2.0|2.0|2.0|2.0|
2.0|2.0|2.0|2.0|
3.0|5.0|2.0|2.0|
3.0|5.0|5.0|3.0|
2.0|2.0|5.0|3.0|
2.0|2.0|2.0|2.0|
2.0|2.0|2.0|2.0|
5.0|5.0|2.0|2.0|
5.0|5.0|5.0|5.0|
2.0|2.0|5.0|5.0|
2.0|2.0|2.0|2.0|
2.0|2.0|2.0|2.0|
5.0|3.0|2.0|2.0|
5.0|3.0|3.0|5.0|
2.0|2.0|3.0|5.0|
2.0|2.0|2.0|2.0|
2.0|2.0|2.0|2.0|
3.0|0.0|2.0|2.0|
3.0|0.0|0.0|3.0|
2.0|2.0|0.0|3.0|
2.0|2.0|2.0|2.0|
2.0|2.0|2.0|2.0|
0.0|0.0|2.0|2.0|
0.0|0.0|0.0|0.0|
2.0|2.0|0.0|0.0|
2.0|2.0|2.0|2.0|
2.0|2.0|2.0|2.0|
0.0|1.0|2.0|2.0|
0.0|1.0|1.0|0.0|
2.0|2.0|1.0|0.0|
2.0|2.0|2.0|2.0|
2.0|2.0|2.0|2.0|
1.0|1.0|2.0|2.0|
1.0|1.0|1.0|1.0|
2.0|2.0|1.0|1.0|
2.0|2.0|2.0|2.0|
2.0|2.0|2.0|2.0|
1.0|2.0|2.0|2.0|
1.0|2.0|2.0|1.0|
2.0|2.0|2.0|1.0|
2.0|2.0|2.0|2.0|
2.0|2.0|2.0|2.0|
2.0|4.0|2.0|2.0|
2.0|4.0|4.0|2.0|
2.0|2.0|4.0|2.0|
2.0|2.0|2.0|2.0|
2.0|2.0|2.0|2.0|
4.0|3.0|2.0|2.0|
4.0|3.0|3.0|4.0|
2.0|2.0|3.0|4.0|
2.0|2.0|2.0|2.0|
2.0|2.0|2.0|2.0|
3.0|2.0|2.0|2.0|
3.0|2.0|2.0|3.0|
2.0|2.0|2.0|3.0|
2.0|2.0|2.0|2.0|
END
```

Appendix B

Questionnaire answers

The following pages show the charts containing the answers to the SoundChart and SoundChart3D questionnaires. As stated in chapter 8, we received 23 answers for the SoundChart test and 18 answers for the SoundChart3D test.

SoundChart Test		
Name	Noirhomme	Martin
First name	Monique	Xavier
Age		22
Gender	Female	Male
Title/position	Doctor	student
Field of activity	Statistics	computer science
e-mail	mno@info.fundp.ac.be	xmartin@info.fundp.ac.be
instrument played ?	NO	NO
which one ?		
For how many years ?		
Musical experience level	Competent	No experience
Question 1a - Yes	Yes	Yes
Question 1b - 1920	1930	1925
Question 1c	3	5
Question 2a - Yes	Yes	Yes
Question 2b - 3	2	1
Question 2c	2	1
Question 3a - True	True	True
Question 3b - Linear	Exponential	Exponential
Question 3c - True	True	No idea
Question 3d	2	1
Question 4a - Yes	Yes	Yes
Question 4b - False	True	False
Question 4c	2	1
Acoustic Grand	8	5
Steel String guitar	7	4
Violin	7	6
SynthStrings 2	6	9
Pan Flute	5	7
Drum Celesta	8	6
Slap Bass 1	5	3
Timpani	5	7
Tinkle bell	5	6
Woodblock	8	9
Pitch mapping	Always useful	Essential
Beat drums mapping	Sometimes useful	Sometimes useful
Stereo mapping	Useless	Sometimes useful
Extreme values detection	Essential	Essential
Sonification in general	Always useful	Always useful
Results : / 9	7	6

Vanderavero	Stephane
Nicolas	Nicoll
22	23
Male	Male
Student	Master in computer science
Computer Science (Intrusion Detection)	Telecom
nvandera@info.fundp.ac.be	stephane.nicoll@mail.be
YES	NO
Piano	
10	
Novice	No experience
Yes	Yes
1930	1930
3	1
Yes	Yes
4	2
2	2
True	True
Linear	Linear
True	False
1	1
Yes	Yes
False	False
3	1
9	10
6	10
4	8
10	6
5	8
8	6
2	5
4	8
8	6
10	7
Essential	Essential
Sometimes useful	Sometimes useful
Essential	Sometimes useful
Always useful	Essential
Always useful	Sometimes useful
9	8

Bontemps	HO	DE SOMER
Yves	Kwai Sung	pierre
23	36	25
Male	Male	Male
Master in computer science	employee	EMPLOYEE
	computing	COTTON TRADING
ybontemp@info.fundp.ac.be	ksh@bvdep.com	beeboske@yahoo.com
NO	NO	NO
No experience	No experience	Novice
Yes	Yes	No
1930	1930	1920
1	2	2
Yes	Yes	Yes
3	2	2
1	4	2
True	True	True
Exponential	Exponential	Linear
True	True	True
1	3	1
Yes	Yes	Yes
False	No idea	False
1	3	1
7	10	8
6	10	3
8	8	5
9	9	6
0	10	7
0	10	6
0	10	3
0	9	4
4	9	2
6	10	8
Sometimes useful	Always useful	Always useful
Sometimes useful	Always useful	Always useful
Useless	Essential	Essential
Always useful	Always useful	Essential
Sometimes useful	Essential	Always useful
8	7	8

De Pauw	Dallons	Panneels
Aurélia	Quentin	Pascal
21	23	28
Female	Male	Male
Miss	Mr.	
student	Student in Computer Sciences	computer science
aurelia_depauw@yahoo.fr	qdallons@info.fundp.ac.be	pepouille@skynet.be
NO	NO	NO
No experience	Novice	No experience
Yes	Yes	Yes
1930	1867	1920
4	4	2
Yes	Yes	Yes
6	3	5
5	5	2
True	True	False
Exponential	Exponential	Linear
True	True	True
3	3	2
Yes	Yes	Yes
False	False	True
4	3	2
8	7	5
5	10	8
4	8	10
7	1	0
7	5	4
	5	5
5	8	8
4	7	6
3	6	4
4	10	2
Always useful	Essential	Sometimes useful
Sometimes useful	Essential	Useless
Always useful	Sometimes useful	Useless
Sometimes useful	Always useful	Sometimes useful
Useless	Sometimes useful	Sometimes useful
7	7	7

Dallons	DALLONS	Weltens
Roxane	ALAIN	Michel
21	48	41
Female	Male	Male
student	PHARMACIEN	operator
economy	SANTE	logistic
e980313@fundp.ac.be	alain.dallons@skynet.be	michel.weltens@brutele.be
NO	YES	NO
	piano	
	10	
Novice	Average abilities	No experience
Yes	Yes	No idea
1920	1933	1903
2	4	4
Yes	Yes	Yes
2	5	7
2	3	0
True	True	True
Exponential	Exponential	Exponential
True	True	True
4	3	8
Yes	Yes	No idea
False	False	False
2	2	1
10	10	6
7	9	8
10	7	8
7	5	1
6	9	10
7	7	3
6	9	6
9	8	6
9	9	8
6		8
Sometimes useful	Sometimes useful	Sometimes useful
Sometimes useful	Sometimes useful	Sometimes useful
Sometimes useful	Essential	Sometimes useful
Sometimes useful	Always useful	Always useful
Useless	Sometimes useful	Sometimes useful
8	8	4

Schöller Nicolas 21 Male Licencié en sciences économiques economics nicscholler@hotmail.com	Mairiaux Aubry 32 Male accountant finances aubry.mairiaux@tiscalinet.be
NO No experience	NO No experience
No 1934 5	Yes 1917 12
Yes 7 6	Yes 12 6
True Linear True 3	True Linear False 5
Yes False 3	Yes True 4
4 6 8 7 6	4 5 9 8 9
5 7 5 6 8	5 6 7 9 7
Essential Always useful Useless Always useful Sometimes useful	Essential Essential Sometimes useful Sometimes useful Sometimes useful
7	6

Delannay	DEMOULIN	Pinera-Gonzalez
Gaetan	Jean-Pol	Mathieu
25	50	21
Male	Male	Male
Chercheur	M.	student
aux Facs	logiciels	Engineering
gdy@info.fundp.ac.be	generationimage@skynet.be	mpinerag@student.fsa.ucl.ac.be
YES	NO	NO
Violon et sampler		
20		
Competent	No experience	Expert
Yes	No	No
1930	1918	1915
2	3	4
Yes	Yes	Yes
10	3	1
3	2	2
True	True	True
Exponential	Linear	Linear
True	False	True
1	2	2
Yes	No	No
False	False	False
2	2	1
9	4	9
8	6	9
5	7	8,5
3	5	6,5
7	8	4
8	7	3
7	6	6
8	8	5
8	4	5
9	8	7,5
Essential	Always useful	Sometimes useful
Essential	Essential	Sometimes useful
Sometimes useful	Sometimes useful	Sometimes useful
Sometimes useful	Essential	Essential
Sometimes useful	Always useful	Sometimes useful
7	6	6

scholler	Schöller	van passel
martine	Stéphanie	guy
48	19	49
Female	Female	Male
mummy of the genius	student	sales representative
housewife	vet	commercial
NO	YES	NO
	flute	
	12	
No experience	Average abilities	No experience
Yes	Yes	No
1937	1923	1910
5	2	5
Yes	Yes	Yes
2	3	4
3	2	3
True	True	True
Linear	Linear	Linear
False	False	False
2	1	4
Yes	Yes	Yes
False	False	False
2	1	3
7	7	9
9	8	7
5	8	9
4	3	5
9	5	9
6	4	6
7	2	5
9	3	7
8	5	9
5	4	8
Always useful	Always useful	Always useful
Sometimes useful	Sometimes useful	Essential
Sometimes useful	Sometimes useful	Essential
Always useful	Sometimes useful	Always useful
Essential	Essential	Essential
7	8	7

SCHOLLER	Lecerf
Michel	Audrey
54	23
Male	Female
MANAGING Director	student
Trading	computer science
	alecerf@info.fundp.ac.be
NO	NO
No experience	Average abilities
Yes	Yes
1939	1910
3	3
Yes	Yes
12	3
2	3
True	True
Linear	Exponential
True	True
2	3
Yes	Yes
False	False
2	3
10	6
4	5
8	7
2	8
6	4
2	0
10	2
6	6
8	4
4	1
Essential	Essential
Sometimes useful	Essential
Always useful	Useless
Essential	Essential
Always useful	Sometimes useful
7	8

SoundChart3D Test		
Name	Martin	Nahimana
First name	Xavier	Adolphe
Age	22	
Gender	Male	Male
Title/position	student	
Field of activity	computer science	computer science
e-mail	xmartin@info.fundp.ac.be	anahimana@hotmail.com
instrument played ?	NO	NO
which one ?		
For how many years ?		
Musical experience level	No experience	No experience
Question 1a - NW	W	C
Question 1a - S	S	SW
Question 1b - Yes	Yes	Yes
Question 1c	2	4
Question 2a - E	E	NE
Question 2a - SE	SE	C
Question 2b - No	No	No
Question 2c	2	3
Question 3a - NW	NW	C
Question 3a - SW	C	SW
Question 3b - Yes	Yes	No
Question 3c	1	3
Vertical travelling	9	6
Horizontal travelling	9	5
Diagonal travelling	8	7
Pitch mapping	Essential	Sometimes useful
Beat drums mapping	Sometimes useful	Essential
End of line notification	Essential	Essential
3D Sonification in general	Always useful	Sometimes useful
Results : / 9	7	3

Vanderavero	Stephane
Nicolas	Nicoll
22	23
Male	Male
Student	Master in computer science
Computer Science (Intrusion Detection)	Telecom
nvandera@info.fundp.ac.be	stephane.nicoll@mail.be
YES	NO
Piano	
10	
Novice	No experience
NW	N
E	W
Yes	Yes
3	3
S	SW
SE	S
No	Yes
2	2
NW	NW
C	C
Yes	No
2	1
10	3
8	0
2	4
Essential	Always useful
Sometimes useful	Sometimes useful
Essential	Always useful
Sometimes useful	Sometimes useful
6	2

Bontemps	de somer	Dallons
Yves	pierre	Quentin
22	25	23
Male	Male	Male
PhD Student	EMPLOYEE	Mr.
Computer Science	COTTON TRADING	Student in Computer Sciences
ybontemp@info.fundp.ac.be	beeboske@yahoo.com	qdallons@info.fundp.ac.be
NO	NO	NO
No experience	Novice	Novice
NE	NE	W
C	C	C
No idea	Yes	Yes
1	2	3
C	E	C
S	C	S
No	Yes	No
2	1	5
No idea		W
		SW
	No idea	Yes
3		6
0	8	10
0	7	5
0	5	1
Useless	Sometimes useful	Essential
Sometimes useful	Always useful	Sometimes useful
Sometimes useful	Always useful	Sometimes useful
Useless	Always useful	Sometimes useful
1	2	4

De Pauw	Dallons	DALLONS
Aurélia	Roxane	ALAIN
21	21	48
Female	Female	Male
Miss	Student	PHARMACIEN
Student	economy	SANTE
aurelia_depauw@yahoo.fr	e980313@fundp.ac.be	alain.dallons@skynet.be
NO	NO	YES
		piano
		10
No experience	Novice	Average abilities
W	N	NW
C	S	C
No idea	No	Yes
7	7	3
E	C	W
SE	SW	C
Yes	No	No
7	2	2
NW	W	NW
C	C	N
Yes	Yes	No
5	4	3
7	2	5
7	8	5
8	5	2
Always useful	Sometimes useful	Sometimes useful
Sometimes useful	Sometimes useful	Always useful
Always useful	Always useful	Always useful
Sometimes useful	Useless	Useless
4	3	4

Schöller	Mairiaux
Nicolas	Aubry
21	32
Male	Male
Junior executive	accountant
economics	finances
nicscholler@hotmail.com	aubry.mairiaux@tiscalinet.be
NO	NO
No experience	No experience
NW	
S	
Yes	No idea
4	5
E	W
SE	C
No	Yes
4	5
NW	W
C	C
Yes	No
2	6
8	0
8	5
5	5
Always useful	Sometimes useful
Useless	Sometimes useful
Always useful	Sometimes useful
Sometimes useful	Sometimes useful
8	0

Pinera-Gonzalez	scholler	Schöller
Mathieu	martine	Stephanie
21	48	19
Male	Female	Female
Student	mam	student
Engineering		vet
mpinerag@student.fsa.ucl.ac.be		
NO	NO	YES
		flute
		12
Novice	No experience	Average abilities
NW	NW	N
S	C	W
No	Yes	No
6	5	2
C	E	C
SE	SE	SE
No	Yes	Yes
4	5	2
NW	NW	W
NE	W	C
Yes	No	No
3	3	1
4	8	8
4	6	8
5	7	3
Sometimes useful	Sometimes useful	Always useful
Sometimes useful	Always useful	Sometimes useful
Essential	Essential	Essential
Sometimes useful	Always useful	Always useful
6	5	1

van passel	SCHOLLER	Lecerf
guy	Michel	Audrey
49	54	23
Male	Male	Female
sales representative	Managing Director	student
commercial	Trading	computer science
		alecerf@info.fundp.ac.be
NO	NO	NO
No experience	No experience	Average abilities
NW	NW	NW
S	S	C
Yes	No	Yes
6	6	7
E	NE	E
NE	C	S
No	No	No
3	5	5
NE	NW	NW
W	SW	W
Yes	Yes	Yes
5	5	4
6	8	5
8	4	5
4	6	6
Always useful	Essential	Essential
Essential	Always useful	Sometimes useful
Essential	Essential	Essential
Essential	Always useful	Sometimes useful
6	6	6

Appendix C

Source code

C.1 SoundChart hierarchy

The SoundChart application program can be subdivided into five distinct parts. While the initialization classes are used only once (when the program starts), the global classes are used all along the program execution. The main chart classes are used for creating the main data values and drawing the chart. The algorithm classes contain methods for smoothing the data values. The sonification classes map the selected chart to sound using the MIDI sound format.

The SoundChart class hierarchy is shown in figure C.1. An arrow represents a dependence between two classes (i.e. the class from where the arrow starts uses methods from the other class).

C.2 SoundChart3D hierarchy

The SoundChart3D application program can be subdivided into four distinct parts. Like in the SoundChart program, the initialization classes are used only once and the global classes are used all along the program execution. The 3D scene group handles the 3D values and the corresponding 3D scene. The sonification classes map the 3D scene to sound using the selected travelling method.

The SoundChart3D class hierarchy is shown in figure C.2.

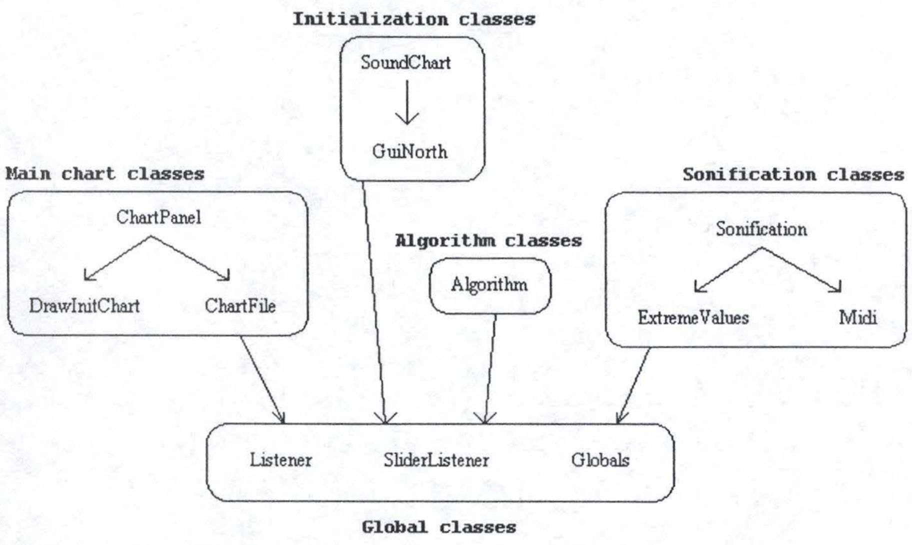


Figure C.1: SoundChart class hierarchy

C.3 Code listings

The following pages contain listings of SoundChart's and SoundChart3D's classes. Each class is briefly explained in a short introduction, and is then detailed along with plenty of comments.

The `Midi` class appears only once, but is used by both application programs.

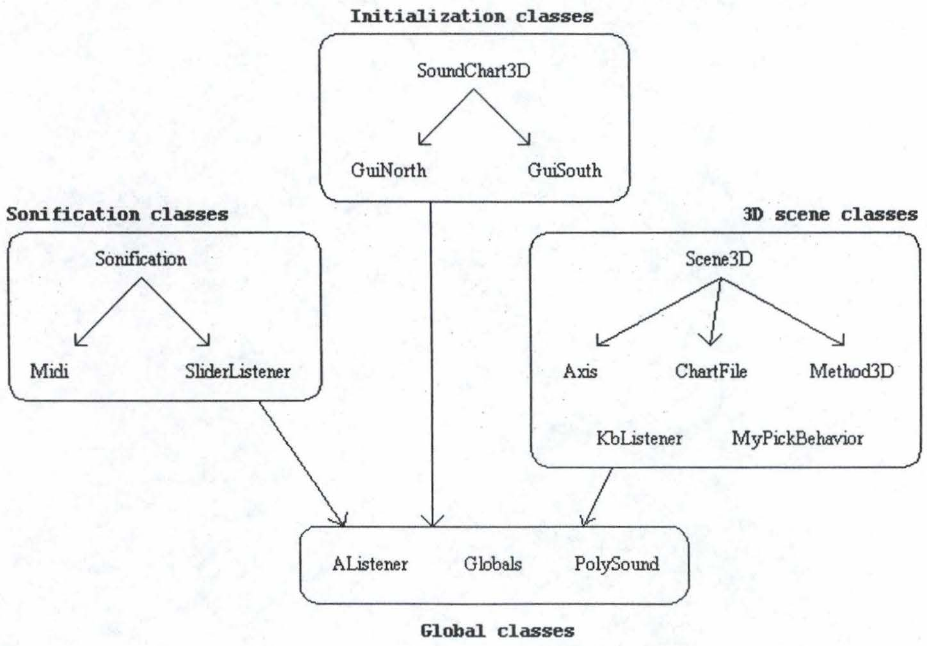


Figure C.2: SoundChart3D class hierarchy

SoundChart.java

The SoundChart class is the main class of the application. The first action is to initialize the two windows used in the application. SCFrame represents the main window, where the "Main Chart", "Algorithms" and "Sonification" panels are located (among other objects). SOFrame represents the window containing additional sonification options. The other main operation of the SoundChart class is to initialize the MIDI objects used throughout the application. The same objects are closed when the program terminates (i.e. when the main window is closed).

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.sound.midi.*;
import java.io.*;
import java.io.IOException;
import java.text.DecimalFormat;
import java.text.ParseException;
import java.applet.*;


class MainSplitPane
{
    // the main window is split in two parts
    private JSplitPane splitPane;
    private ChartPanel pSouth;
    private JTabbedPane tpNorth;

    public MainSplitPane()
    {
        tpNorth = new JTabbedPane();
        pSouth = new ChartPanel();

        // save tpNorth (used to disable tabs)
        Globals.tabbedpane = tpNorth;

        pSouth.setLayout(null);

        Component panel1 = GuiNorth.makePanelValue();
        tpNorth.addTab("   Main Chart    ", null, panel1, "Main chart settings");

        Component panel2 = GuiNorth.makePanelAlgorithm();
        tpNorth.addTab("   Algorithms    ", null, panel2, "Algorithm settings");

        Component panel3 = GuiNorth.makePanelSonification();
        tpNorth.addTab("   Sonification    ", null, panel3, "Sonification settings");

        Component panel4 = GuiNorth.makePanelStatus();
        tpNorth.addTab("   Status    ", null, panel4, "Status view");

        tpNorth.setSelectedIndex(0);
        splitPane = new JSplitPane(JSplitPane.VERTICAL_SPLIT, tpNorth, pSouth);
        splitPane.setOneTouchExpandable(false);
        splitPane.setDividerLocation(Globals.screenHeight / 2 - 30);
        splitPane.setContinuousLayout(false);

        // provide minimum sizes for the two components in the split pane
        Dimension minimumSize = new Dimension(100, 50);
        tpNorth.setMinimumSize(minimumSize);
        pSouth.setMinimumSize(minimumSize);

        // provide a preferred size for the split pane
        splitPane.setPreferredSize(new Dimension(400, 200));
    }


    public JSplitPane getSplitPane()
    {
        return splitPane;
    }
}


class SCFrame extends JFrame
{
    public SCFrame()
    {
        setTitle("SoundChart");

        addWindowListener(new WindowAdapter()
        {
            public void windowClosing(WindowEvent e)
            {
                SoundChart.closeMidiObjects();
                System.exit(0);
            }
        } );

        setSize(Globals.screenWidth, Globals.screenHeight);

        Container contentPane = getContentPane();

        MainSplitPane sp = new MainSplitPane();

        contentPane.add(sp.getSplitPane());
    }
}


class SOFrame extends JFrame
{
    public SOFrame()
    {
        setTitle("Sonification options");
        setBounds(300,220,430,290);

        addWindowListener(new WindowAdapter()
        {
            public void windowClosing(WindowEvent e)
            {
                Globals.bOptions.setEnabled(true);
            }
        } );

        getContentPane().add(GuiNorth.makePanelSoundOptions());
    }
}


public class SoundChart
{
    // retrieves the number pressed by the user
    private static int getKbdInt()
    {
        int val = 0;

        try
        {
            InputStreamReader isr = new InputStreamReader(System.in);
            BufferedReader br = new BufferedReader(isr);

            String s = br.readLine();
            DecimalFormat df = new DecimalFormat();

            val = (df.parse(s)).intValue();
        }
        catch (IOException ex)
        {
            System.out.println("\nERROR in getKbdInt(): " + ex + "\n");
            System.exit(1);
        }
        catch (ParseException ex)
        {
            System.out.println("\nERROR in getKbdInt(): " + ex + "\n");
            System.exit(1);
        }

        return val;
    }


    static void initMidiObjects()
    {
        MidiDevice.Info[] midiDevices = MidiSystem.getMidiDeviceInfo();

        if (midiDevices.length == 0)
        {
            System.out.println("No MIDI devices found on this system!\n");
            System.exit(0);
        }

        // print available MIDI devices
        System.out.println("Available MIDI devices on this system:\n");
        for (int i = 0; i < midiDevices.length; i++)
            System.out.println(i + ": " + midiDevices[i]);

        // the user must choose a MIDI device
        System.out.print("\nUse device number: ");
        int iDevice = getKbdInt();
        try
        {
            System.out.print("\nGetting MIDI device...");
            Globals.midiDevice = MidiSystem.getMidiDevice(midiDevices[iDevice]);
            System.out.println(" ok");

            System.out.print("Getting synthesizer...");
            Globals.midiSynthesizer = MidiSystem.getSynthesizer();
            System.out.println(" ok");

            System.out.print("Getting sequencer...");
            Globals.midiSequencer = MidiSystem.getSequencer();
            System.out.println(" ok\n");
        }
        catch (MidiUnavailableException ex)
        {
            System.out.println("\nERROR in initMidiDevice(): " + ex + "\n");
            System.exit(1);
        }
        catch (SecurityException ex)
        {
            System.out.println("\nERROR in initMidiDevice(): " + ex + "\n");
            System.exit(1);
        }
    }


    static void closeMidiObjects()
    {
        System.out.print("Closing MIDI objects...");
        Globals.midiSynthesizer.close();
        Globals.midiSequencer.close();
        Globals.midiDevice.close();
        System.out.println(" ok\n");
    }


    public static void main(String[] args)
    {
        // create windows
        Globals.mainFrame = new SCFrame();
        Globals.optionsFrame = new SOFrame();

        System.out.println(" ___________________________");
        System.out.println("|                           |");
        System.out.println("|        SOUNDCHART         |");
        System.out.println("|___________________________|\n\n");

        // initialize MIDI objects
        initMidiObjects();

        // show main window
        Globals.mainFrame.show();
    }
}
```

GuiNorth.java

The `GuiNorth` class initializes the upper part of SoundChart's main window. Four panels can be distinguished: the main chart panel, the sonification panel, the algorithms panel and the status panel. Furthermore, the sonification panel contains the information to initialize the sound options panel, contained by the "Sonification Options" window. Two extra classes are defined to specify the `MyTable` class, which is used by the main chart panel. This class is merely a simple `JTable` with special attributes (e.g. the cells are not editable).

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.border.*;
import java.util.Vector;
import javax.swing.JSlider;
import javax.swing.table.*;
import javax.swing.event.ListSelectionEvent;


public class GuiNorth
{
    |          MAIN CHART PANEL          |

    protected static Component makePanelValue()
    {
        JPanel panel = new JPanel();
        panel.setLayout(null);

        /** Table panel **/

        // create the main chart table panel
        JPanel pTable = new JPanel();
        pTable.setLayout(null);
        pTable.setBounds(25, 10, 425, 260);
        Border bEtched1 = BorderFactory.createEtchedBorder();
        Border bTitle1 = BorderFactory.createTitledBorder(bEtched1, "Chart Table");
        pTable.setBorder(bTitle1);

        // create the main chart table
        Globals.vCoord = new Vector();
        Vector vNames = new Vector();
        vNames.add("X");
        vNames.add("Y");
        Globals.tCoord = new MyTable(Globals.vCoord, vNames);

        // add a scrollpane to the table
        JScrollPane spCoord = new JScrollPane(Globals.tCoord);
        spCoord.setBounds(25, 30, 170, 210);
        pTable.add(spCoord);

        // create textfields to manually add a value to the table
```

```
JLabel ltfx = new JLabel("X value :");
JLabel ltfy = new JLabel("Y value :");
ltfx.setBounds(225, 25, 50, 20);
ltfy.setBounds(225, 50, 50, 20);
Globals.tfx = new JTextField();
Globals.tfy = new JTextField();
Globals.tfx.setBounds(295, 25, 100, 20);
Globals.tfy.setBounds(295, 50, 100, 20);

/** create main chart buttons **/

Globals.bInsert = new JButton("Insert point");
Globals.bInsert.setBounds(225, 90, 170, 30);
Globals.bInsert.addActionListener(new Listener());

Globals.bDelete = new JButton("Delete selected line");
Globals.bDelete.setBounds(225, 130, 170, 30);
Globals.bDelete.addActionListener(new Listener());
Globals.bDelete.setEnabled(false);

Globals.bImport = new JButton("Load from file...");
Globals.bImport.setBounds(225, 170, 170, 30);
Globals.bImport.addActionListener(new Listener());

Globals.bClearTable = new JButton("Clear table");
Globals.bClearTable.setBounds(225, 210, 170, 30);
Globals.bClearTable.addActionListener(new Listener());
Globals.bClearTable.setEnabled(false);

// add all these elements to the table panel
pTable.add(ltfx);
pTable.add(ltfy);
pTable.add(Globals.tfx);
pTable.add(Globals.tfy);
pTable.add(Globals.bInsert);
pTable.add(Globals.bDelete);
pTable.add(Globals.bImport);
pTable.add(Globals.bClearTable);

/** Chart Settings panel **/

// create chart settings panel
JPanel pChartSettings = new JPanel();
pChartSettings.setLayout(null);
pChartSettings.setBounds(475,10,200,260);
Border bEtched3 = BorderFactory.createEtchedBorder();
Border bTitle3 = BorderFactory.createTitledBorder(bEtched3, "Chart Settings");
pChartSettings.setBorder(bTitle3);

// create slider for number of values on X axis
JLabel lNumValX = new JLabel("# Absciss values :");
lNumValX.setBounds(20,20,160,30);
Globals.slNumValuesX = new JSlider(JSlider.HORIZONTAL, 2, 12, 8);
Globals.slNumValuesX.setBounds(20,50,160,50);
Globals.slNumValuesX.addChangeListener(new SliderListener());
Globals.slNumValuesX.setMajorTickSpacing(2);
Globals.slNumValuesX.setMinorTickSpacing(1);
Globals.slNumValuesX.setPaintTicks(true);
Globals.slNumValuesX.setPaintLabels(true);
```

```
        Globals.slNumValuesX.setSnapToTicks(true);
        Globals.slNumValuesX.setBorder(BorderFactory.createEmptyBorder(0,0,10,0));

        // create slider for number of values on Y axis
        JLabel lNumValY = new JLabel("# Ordinate values :");
        lNumValY.setBounds(20,105,160,30);
        Globals.slNumValuesY = new JSlider(JSlider.HORIZONTAL, 2, 12, 8);
        Globals.slNumValuesY.setBounds(20,135,160,50);
        Globals.slNumValuesY.addChangeListener(new SliderListener());
        Globals.slNumValuesY.setMajorTickSpacing(2);
        Globals.slNumValuesY.setMinorTickSpacing(1);
        Globals.slNumValuesY.setPaintTicks(true);
        Globals.slNumValuesY.setPaintLabels(true);
        Globals.slNumValuesY.setSnapToTicks(true);
        Globals.slNumValuesY.setBorder(BorderFactory.createEmptyBorder(0,0,10,0));

        // create textfield for number of decimals
        JLabel lNumDecimals = new JLabel("# decimals :");
        lNumDecimals.setBounds(20,210,100,20);
        Globals.tfNumDecimals = new JTextField("2");
        Globals.tfNumDecimals.setBounds(125,210,50,20);

        // add all these elements to the chart settings panel
        pChartSettings.add(lNumValX);
        pChartSettings.add(Globals.slNumValuesX);
        pChartSettings.add(lNumValY);
        pChartSettings.add(Globals.slNumValuesY);
        pChartSettings.add(lNumDecimals);
        pChartSettings.add(Globals.tfNumDecimals);

        // create button to draw main chart
        Globals.bDrawMainChart = new JButton("Draw main chart");
        Globals.bDrawMainChart.setBounds(710,198,200,30);
        Globals.bDrawMainChart.addActionListener(new Listener());
        Globals.bDrawMainChart.setEnabled(false);

        // create button to hide main chart
        Globals.bHideMainChart = new JButton("Hide main chart");
        Globals.bHideMainChart.setBounds(710,238,200,30);
        Globals.bHideMainChart.addActionListener(new Listener());
        Globals.bHideMainChart.setEnabled(false);

        // add them to the main panel
        panel.add(Globals.bDrawMainChart);
        panel.add(Globals.bHideMainChart);

        // add table panel and chart settings panel to main panel
        panel.add(pTable);
        panel.add(pChartSettings);

        return panel;
    }
```

ALGORITHM PANEL

```
protected static Component makePanelAlgorithm()
{
    JPanel panel = new JPanel();
```

```
panel.setLayout(null);

// create algorithm panel
JPanel pAlgorithm = new JPanel();
pAlgorithm.setLayout(null);
pAlgorithm.setBounds(25,10,425,260);
Border bEtched = BorderFactory.createEtchedBorder();
Border bTitle = BorderFactory.createTitledBorder(bEtched, "Algorithm settings");
pAlgorithm.setBorder(bTitle);

// create select algorithm combobox
JLabel lAlgorithm = new JLabel("Select algorithm :");
lAlgorithm.setBounds(25,40,180,20);
Globals.cbAlgorithm = new JComboBox();
Globals.cbAlgorithm.setEditable(false);
Globals.cbAlgorithm.setBounds(180,35,225,30);
Globals.cbAlgorithm.addItem("Linear Moving Average");
Globals.cbAlgorithm.addItem("Exponential Moving Average");
Globals.cbAlgorithm.addItem("Weighted Moving Average");
Globals.cbAlgorithm.addActionListener(new Listener());
Globals.cbAlgorithm.setEnabled(false);

// create slider for Linear MA order
Globals.lOrder = new JLabel("Order :");
Globals.lOrder.setBounds(25,90,180,20);
Globals.slOrder = new JSlider(JSlider.HORIZONTAL, 0, 0, 0);
Globals.slOrder.setBounds(180,94,155,50);
Globals.slOrder.addChangeListener(new SliderListener());
Globals.slOrder.setMajorTickSpacing(2);
Globals.slOrder.setMinorTickSpacing(1);
Globals.slOrder.setBorder(BorderFactory.createEmptyBorder(0,0,10,0));
Globals.slOrder.setEnabled(false);
Globals.lValOrder = new JLabel("0");
Globals.lValOrder.setBounds(355,90,50,20);

// create slider for Exponential MA percentage
Globals.lPercentage = new JLabel("Percentage :");
Globals.lPercentage.setBounds(25,90,180,20);
Globals.slPercentage = new JSlider(JSlider.HORIZONTAL, 1, 100, 10);
Globals.slPercentage.setBounds(180,94,155,50);
Globals.slPercentage.addChangeListener(new SliderListener());
Globals.slPercentage.setMajorTickSpacing(5);
Globals.slPercentage.setMinorTickSpacing(1);
Globals.slPercentage.setBorder(BorderFactory.createEmptyBorder(0,0,10,0));
Globals.slPercentage.setEnabled(false);
Globals.lValPercentage = new JLabel("10");
Globals.lValPercentage.setBounds(355,90,50,20);
Globals.lPercentage.setVisible(false);
Globals.slPercentage.setVisible(false);
Globals.lValPercentage.setVisible(false);

// create slider for Weighted MA coefficient
Globals.lCoeff = new JLabel("Coefficient :");
Globals.lCoeff.setBounds(25,90,180,20);
Globals.slCoeff = new JSlider(JSlider.HORIZONTAL, 1, 1, 1);
Globals.slCoeff.setBounds(180,94,155,50);
Globals.slCoeff.addChangeListener(new SliderListener());
Globals.slCoeff.setMajorTickSpacing(2);
Globals.slCoeff.setMinorTickSpacing(1);
```

```
        Globals.slCoeff.setBorder(BorderFactory.createEmptyBorder(0,0,10,0));
        Globals.slCoeff.setEnabled(false);
        Globals.lValCoeff = new JLabel("1");
        Globals.lValCoeff.setBounds(355,90,50,20);
        Globals.lCoeff.setVisible(false);
        Globals.slCoeff.setVisible(false);
        Globals.lValCoeff.setVisible(false);

        // create button to draw selected algorithm
        Globals.bDrawSelAlg = new JButton("Draw chart using selected algorithm");
        Globals.bDrawSelAlg.setBounds(70,200,278,30);
        Globals.bDrawSelAlg.addActionListener(new Listener());
        Globals.bDrawSelAlg.setEnabled(false);

        // add all these elements to algorithm panel
        pAlgorithm.add(lAlgorithm);
        pAlgorithm.add(Globals.cbAlgorithm);
        pAlgorithm.add(Globals.lOrder);
        pAlgorithm.add(Globals.slOrder);
        pAlgorithm.add(Globals.lValOrder);
        pAlgorithm.add(Globals.lPercentage);
        pAlgorithm.add(Globals.slPercentage);
        pAlgorithm.add(Globals.lValPercentage);
        pAlgorithm.add(Globals.lCoeff);
        pAlgorithm.add(Globals.slCoeff);
        pAlgorithm.add(Globals.lValCoeff);
        pAlgorithm.add(Globals.bDrawSelAlg);

        /** create hide chart buttons **/

        Globals.bHideAlgMA = new JButton("Hide Linear Moving Average chart");
        Globals.bHideAlgMA.setBounds(650,158,260,30);
        Globals.bHideAlgMA.addActionListener(new Listener());
        Globals.bHideAlgMA.setEnabled(false);

        Globals.bHideAlgEMA = new JButton("Hide Exponential Moving Average chart");
        Globals.bHideAlgEMA.setBounds(650,198,260,30);
        Globals.bHideAlgEMA.addActionListener(new Listener());
        Globals.bHideAlgEMA.setEnabled(false);

        Globals.bHideAlgWMA = new JButton("Hide Weighted Moving Average chart");
        Globals.bHideAlgWMA.setBounds(650,238,260,30);
        Globals.bHideAlgWMA.addActionListener(new Listener());
        Globals.bHideAlgWMA.setEnabled(false);

        // add these buttons and algorithm panel to main algorithm panel
        panel.add(pAlgorithm);
        panel.add(Globals.bHideAlgMA);
        panel.add(Globals.bHideAlgEMA);
        panel.add(Globals.bHideAlgWMA);

        return panel;
}
```

SONIFICATION PANEL

```
protected static Component makePanelSonification()
{
```

```
JPanel panel = new JPanel();
panel.setLayout(null);

// create sonification settings (sonset) panel
JPanel pSonification = new JPanel();
pSonification.setLayout(null);
pSonification.setBounds(25,10,425,260);
Border bEtched = BorderFactory.createEtchedBorder();
Border bTitle = BorderFactory.createTitledBorder(bEtched, "MIDI settings");
pSonification.setBorder(bTitle);

// create program combo box
JLabel lProgram = new JLabel("Select program :");
lProgram.setBounds(25,40,180,20);
// fill vProgramCB vector with Midi programs
Midi.fillProgramComboBox();
Globals.cbProgram = new JComboBox(Globals.vProgramCB);
Globals.cbProgram.setSelectedItem("52    SynthStrings 2");
Globals.cbProgram.setEditable(false);
Globals.cbProgram.setBounds(180,35,225,30);
Globals.cbProgram.addActionListener(new Listener());
Globals.cbProgram.setEnabled(false);

// create interval slider
JLabel lInterval = new JLabel("Interval :");
lInterval.setBounds(25,90,180,20);
Globals.slTime = new JSlider(JSlider.HORIZONTAL, 0, 500, 200);
Globals.slTime.setBounds(180,90,155,25);
Globals.slTime.addChangeListener(new SliderListener());
Globals.slTime.setMajorTickSpacing(5);
Globals.slTime.setMinorTickSpacing(1);
Globals.slTime.setEnabled(false);
Globals.lTime = new JLabel("200 ms" );
Globals.lTime.setBounds(355,86,50,20);

// create minNote slider
JLabel lMinNote = new JLabel("Min Pitch :");
lMinNote.setBounds(25,130,180,20);
Globals.slMinNote = new JSlider(JSlider.HORIZONTAL, 0, 127, 25);
Globals.slMinNote.setBounds(180,130,155,25);
Globals.slMinNote.addChangeListener(new SliderListener());
Globals.slMinNote.setMajorTickSpacing(2);
Globals.slMinNote.setMinorTickSpacing(1);
Globals.slMinNote.setEnabled(false);
Globals.lValMinNote = new JLabel("25");
Globals.lValMinNote.setBounds(355,126,50,20);

// create maxNote slider
JLabel lMaxNote = new JLabel("Max Pitch :");
lMaxNote.setBounds(25,170,180,20);
Globals.slMaxNote = new JSlider(JSlider.HORIZONTAL, 0, 127, 100);
Globals.slMaxNote.setBounds(180,170,155,25);
Globals.slMaxNote.addChangeListener(new SliderListener());
Globals.slMaxNote.setMajorTickSpacing(2);
Globals.slMaxNote.setMinorTickSpacing(1);
Globals.slMaxNote.setEnabled(false);
Globals.lValMaxNote = new JLabel("100");
Globals.lValMaxNote.setBounds(355,166,50,20);
```

```
// create sonification options button
Globals.bOptions = new JButton("More sonification options...");
Globals.bOptions.setBounds(100,210,230,30);
Globals.bOptions.addActionListener(new Listener());
Globals.bOptions.setEnabled(false);

// add all these elements to sonset panel
pSonification.add(lProgram);
pSonification.add(Globals.cbProgram);
pSonification.add(lInterval);
pSonification.add(Globals.slTime);
pSonification.add(Globals.lTime);
pSonification.add(lMinNote);
pSonification.add(Globals.slMinNote);
pSonification.add(Globals.lValMinNote);
pSonification.add(lMaxNote);
pSonification.add(Globals.slMaxNote);
pSonification.add(Globals.lValMaxNote);
pSonification.add(Globals.bOptions);

// create sound data panel
JPanel pSoundData = new JPanel();
pSoundData.setLayout(null);
pSoundData.setBounds(475,10,425,260);
Border bEtched2 = BorderFactory.createEtchedBorder();
Border bTitle2 = BorderFactory.createTitledBorder(bEtched2, "Sound data");
pSoundData.setBorder(bTitle2);

// create chart combo box
JLabel lChart = new JLabel("Available charts :");
lChart.setBounds(25,40,180,20);
Globals.cbChart = new JComboBox();
Globals.cbChart.setEditable(false);
Globals.cbChart.setBounds(180,35,225,30);
Globals.cbChart.setEnabled(false);

// create mapping type radio buttons
JLabel lMapping = new JLabel("Mapping type :");
lMapping.setBounds(25,80,140,20);
Globals.rbLinearMapping = new JRadioButton("Linear mapping", true);
Globals.rbLinearMapping.setBounds(180,80,180,30);
Globals.rbLinearMapping.setEnabled(false);
Globals.rbChromaticMapping = new JRadioButton("Chromatic scale mapping", false);
Globals.rbChromaticMapping.setBounds(180,105,180,30);
Globals.rbChromaticMapping.setEnabled(false);
ButtonGroup bgMappingType = new ButtonGroup();
bgMappingType.add(Globals.rbLinearMapping);
bgMappingType.add(Globals.rbChromaticMapping);

// create sequence slider
Globals.slSequence = new JSlider(JSlider.HORIZONTAL, 0, 100, 0);
Globals.slSequence.setBounds(25,170,385,25);
Globals.slSequence.addChangeListener(new SliderListener());
Globals.slSequence.setMajorTickSpacing(10);
Globals.slSequence.setMinorTickSpacing(1);
Globals.slSequence.setEnabled(false);

/** create sonification sequence buttons **/
```

```
int space = 35 + 5;
int xtab[] = new int[7];
xtab[0] = (int)(217.5 - (7.0 * space) / 2.0);
for (int i = 1; i ≤ 6; i++)
    xtab[i] = xtab[i-1] + space;

ImageIcon iReset = new ImageIcon("data/reset1.gif");
ImageIcon iRew = new ImageIcon("data/rew1.gif");
ImageIcon iPlay = new ImageIcon("data/play1.gif");
ImageIcon iPause = new ImageIcon("data/pause1.gif");
ImageIcon iStop = new ImageIcon("data/stop1.gif");
ImageIcon iFfw = new ImageIcon("data/ffw1.gif");
ImageIcon iEnd = new ImageIcon("data/end1.gif");

Globals.bReset = new JButton(iReset);
Globals.bReset.setBounds(xtab[0],200,35,35);
Globals.bReset.addActionListener(new Listener());
Globals.bReset.setEnabled(false);

Globals.bRew = new JButton(iRew);
Globals.bRew.setBounds(xtab[1],200,35,35);
Globals.bRew.addActionListener(new Listener());
Globals.bRew.setEnabled(false);

Globals.bPlay = new JButton(iPlay);
Globals.bPlay.setBounds(xtab[2],200,35,35);
Globals.bPlay.addActionListener(new Listener());
Globals.bPlay.setEnabled(false);

Globals.bPause = new JButton(iPause);
Globals.bPause.setBounds(xtab[3],200,35,35);
Globals.bPause.addActionListener(new Listener());
Globals.bPause.setEnabled(false);

Globals.bStop = new JButton(iStop);
Globals.bStop.setBounds(xtab[4],200,35,35);
Globals.bStop.addActionListener(new Listener());
Globals.bStop.setEnabled(false);

Globals.bFfw = new JButton(iFfw);
Globals.bFfw.setBounds(xtab[5],200,35,35);
Globals.bFfw.addActionListener(new Listener());
Globals.bFfw.setEnabled(false);

Globals.bEnd = new JButton(iEnd);
Globals.bEnd.setBounds(xtab[6],200,35,35);
Globals.bEnd.addActionListener(new Listener());
Globals.bEnd.setEnabled(false);

// add all these elements to sound data panel
pSoundData.add(lChart);
pSoundData.add(Globals.cbChart);
pSoundData.add(lMapping);
pSoundData.add(Globals.rbLinearMapping);
pSoundData.add(Globals.rbChromaticMapping);
pSoundData.add(Globals.slSequence);
pSoundData.add(Globals.bReset);
pSoundData.add(Globals.bRew);
pSoundData.add(Globals.bPlay);
```

```
        pSoundData.add(Globals.bPause);
        pSoundData.add(Globals.bStop);
        pSoundData.add(Globals.bFfw);
        pSoundData.add(Globals.bEnd);

        // add sonset and sound data panel to main sonification panel
        panel.add(pSonification);
        panel.add(pSoundData);

        return panel;
    }


    protected static Component makePanelSoundOptions()
    {
        JPanel panel = new JPanel();
        panel.setLayout(null);

        /** create Extreme Values panel **/

        JPanel pExtreme = new JPanel();
        pExtreme.setLayout(null);
        pExtreme.setBounds(10,10,400,100);
        Border bEtched = BorderFactory.createEtchedBorder();
        Border bTitle = BorderFactory.createTitledBorder(bEtched, "Extreme values");
        pExtreme.setBorder(bTitle);

        Globals.chbExtremeValues = new JCheckBox("Detect extreme values", false);
        Globals.chbExtremeValues.setBounds(25,25,200,25);
        Globals.chbExtremeValues.setEnabled(true);
        Globals.chbExtremeValues.addActionListener(new Listener());
        pExtreme.add(Globals.chbExtremeValues);

        JLabel lExtremeProgram = new JLabel("Warning program:");
        lExtremeProgram.setBounds(25,60,150,20);
        Globals.cbExtremeProgram = new JComboBox(Globals.vProgramCB);
        Globals.cbExtremeProgram.setSelectedItem("72   Clarinet");
        Globals.cbExtremeProgram.setEditable(false);
        Globals.cbExtremeProgram.setBounds(180,55,200,30);
        Globals.cbExtremeProgram.setEnabled(false);
        pExtreme.add(lExtremeProgram);
        pExtreme.add(Globals.cbExtremeProgram);

        /** create Drum Beats panel **/

        JPanel pDrum = new JPanel();
        pDrum.setLayout(null);
        pDrum.setBounds(10,120,400,100);
        Border bEtched2 = BorderFactory.createEtchedBorder();
        Border bTitle2 = BorderFactory.createTitledBorder(bEtched2, "Drum beats");
        pDrum.setBorder(bTitle2);

        Globals.chbDrumBeats = new JCheckBox("Play drum beats", false);
        Globals.chbDrumBeats.setBounds(25,25,200,25);
        Globals.chbDrumBeats.setEnabled(true);
        Globals.chbDrumBeats.addActionListener(new Listener());
        pDrum.add(Globals.chbDrumBeats);

        JLabel lDrumProgram = new JLabel("Drum program:");
        lDrumProgram.setBounds(25,60,150,20);
        Globals.cbDrumProgram = new JComboBox(Globals.vProgramCB);
        Globals.cbDrumProgram.setSelectedItem("48    Timpani");
        Globals.cbDrumProgram.setEditable(false);
        Globals.cbDrumProgram.setBounds(180,55,200,30);
        Globals.cbDrumProgram.setEnabled(false);
        pDrum.add(lDrumProgram);
        pDrum.add(Globals.cbDrumProgram);

        /** create Stereo Panning checkbox **/

        Globals.chbStereo = new JCheckBox("Use left-to-right stereo panning", false);
        Globals.chbStereo.setBounds(15,230,200,25);
        Globals.chbStereo.setEnabled(true);

        // add all these elements to main sonification options panel
        panel.add(pExtreme);
        panel.add(pDrum);
        panel.add(Globals.chbStereo);

        return panel;
    }
```

STATUS PANEL

```
    protected static Component makePanelStatus()
    {
        JPanel panel = new JPanel();
        panel.setLayout(null);

        // create status panel
        JPanel pStatus = new JPanel();
        pStatus.setLayout(null);
        pStatus.setBounds(10, 10, 935, 260);
        Border bEtched = BorderFactory.createEtchedBorder();
        Border bTitle = BorderFactory.createTitledBorder(bEtched, "Status view");
        pStatus.setBorder(bTitle);

        // create the JTextArea
        Globals.taStatus = new JTextArea("--------------- Status -------------", 10, 100);
        Globals.taStatus.setEditable(false);
        Globals.taStatus.setLineWrap(true);

        // add a scrollpane to the area
        JScrollPane spStatus = new JScrollPane(Globals.taStatus);
        spStatus.setBounds(20, 30, 885, 205);
        pStatus.add(spStatus);

        // add the status panel to the main status panel
        panel.add(pStatus);

        return panel;
    }
}


class MyTable extends JTable
{
    public MyTable(Vector rowData, Vector names)
    {
        super(new MyTableModel(rowData, names));
        setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
    }

    public void valueChanged(ListSelectionEvent e)
    {
        super.valueChanged(e);
        if (Globals.bDelete ≠ null) Globals.bDelete.setEnabled(true);
    }

    public boolean isFocusTraversable()
    {
        return false;
    }

    public boolean isManagingFocus()
    {
        return false;
    }
}


class MyTableModel extends DefaultTableModel
{
    public MyTableModel(Vector rows, Vector names)
    {
        super(rows, names);
    }

    public boolean isCellEditable(int x, int y)
    {
        return false;
    }
}
```

Globals.java

The `Globals` class contains the static variables used by the SoundChart application. Working with a single big file containing all global variables is certainly not perfect from a programming point of vue, but it is the easiest way to handle such a large amount of variables since they can be accessed from any point in the program.

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.border.*;
import javax.sound.midi.*;
import java.util.Vector;
import java.awt.Graphics.*;
import java.awt.geom.*;


public class Globals
{
        MAIN CHART PANEL

    // frames representing the 2 windows of SoundChart
    static JFrame mainFrame;
    static JFrame optionsFrame;

    // objects to detect when the user changes the active pane
    static JTabbedPane tabbedpane;
    static ChartPanel chartPanel;

    // array and table containing the Main Chart values
    static Vector vCoord;
    static JTable tCoord;

    // information on Main Chart values
    static double dMinX, dMinY, dMaxX, dMaxY, dIntX, dIntY = 0.0;

    // textfields for entering a new value in the table
    static JTextField tfx;
    static JTextField tfy;

    // buttons on Main Chart panel
    static JButton bInsert;
    static JButton bDelete;
    static JButton bImport;
    static JButton bClearTable;
    static JButton bDrawMainChart;
    static JButton bHideMainChart;

    // current line read in a SoundChart file
    static String sCurrentLine;

    // objects containing the initial and moving average charts
```

```
static GeneralPath gp;
static GeneralPath gpMA;
static GeneralPath gpEMA;
static GeneralPath gpWMA;

// objects used for drawing the charts
static Line2D.Double axeX, axeY;
static Line2D.Float gridXY;
static Line2D.Float originX, originY;
static int cNumberValuesX;
static int cNumberValuesY;
static int cNdecimalsVal = 2;
static JTextField tfNumDecimals;
static JSlider slNumValuesX;
static JSlider slNumValuesY;

// panel on which the charts are drawn
static JPanel pChart;

// bounds of pChart panel
static int cWidthChart;
static int cHeightChart;
static int cLeftChart;
static int cTopChart;

// vectors containing the moving average values
static Vector vMACoord;
static Vector vEMACoord;
static Vector vWMACoord;

// indicates if a chart has already been drawn
static boolean bFirstTime = true;

|      ALGORITHMS PANEL      |

// combo box for selecting an algorithm
static JComboBox cbAlgorithm;

// parameters representing the smoothing window for each algorithm
static JSlider slOrder, slPercentage, slCoeff;

// objects used to display the smoothing window
static JLabel lValOrder, lOrder;
static JLabel lValPercentage, lPercentage;
static JLabel lValCoeff, lCoeff;

// buttons on Algorithms panel
static JButton bDrawSelAlg;
static JButton bHideAlgMA, bHideAlgEMA, bHideAlgWMA;

|      SONIFICATION PANEL      |

// array containing the MIDI instruments
static Vector vProgramCB;

// objects on Sonification panel
static JComboBox cbProgram, cbChart;
static JSlider slTime, slMinNote, slMaxNote, slSequence;
static JLabel lTime, lValMinNote, lValMaxNote;
```

```
    static JButton bPlay, bPause, bStop, bFfw, bRew, bReset, bEnd;
    static JButton bOptions;
    static JCheckBox chbExtremeValues, chbDrumBeats, chbStereo;
    static JComboBox cbDrumProgram, cbExtremeProgram;
    static JRadioButton rbLinearMapping, rbChromaticMapping;

    // used to handle vertical line on chart
    static boolean bClearOldTimeline = false;
    static float timelinePos = -1;

    // MIDI-specific objects
     static MidiDevice midiDevice = null;
    static Synthesizer midiSynthesizer = null;
    static Sequencer midiSequencer = null;
    static Sequence midiSequence = null;
    static Timer timerSequence;
    static float midiSequenceLength;
    static boolean midiSequenceCreated = false;

    | STATUS PANEL |

    // contains only a single element
    static JTextArea taStatus;

    | GENERAL INFORMATION |

    // the screen dimensions are initialized
    static int screenWidth, screenHeight;
    static
    {
        Toolkit tk = Toolkit.getDefaultToolkit();
        Dimension d = tk.getScreenSize();
        screenHeight = d.height * 8 / 9;
        screenWidth = d.width * 19 / 20;
    }
}
```

SliderListener.java

The main goal of the `SliderListener` class is to supervise the position of the sliders and to update the corresponding textfields with the appropriate value. The `stateChanged` method is called whenever the position of a slider changes. Based on the source of the change, the corresponding label is updated. A special case concerns the sonification slider, as the vertical timeline on the chart needs to be updated upon every change of the slider.

```
import java.awt.event.*;
import javax.swing.JSlider;
import java.awt.*;
import javax.swing.*;
import javax.swing.event.*;
import java.awt.Graphics.*;
import java.awt.geom.*;


class SliderListener implements ChangeListener
{
    public void updateTimeline(boolean drawNewOne)
    {
        Line2D.Float timeline;
        Graphics2D g2d = Globals.chartPanel.getGraphics2D();

        g2d.setXORMode(Color.lightGray);
        g2d.setColor(Color.cyan);

        // compute graph begin/end positions
        float dBegin = 0.0f, dEnd = 1.0f;
        String SItem = (String)(Globals.cbChart.getSelectedItem());
        if (SItem ≠ null)
        {
            if (SItem.equals("Linear MA Chart"))
            {
                float lengthB = ((float)(Globals.vMACoord.size()+1)
                    / (float)Globals.vCoord.size());
                float lengthE = ((float)(Globals.vMACoord.size()-1)
                    / (float)Globals.vCoord.size());
                dBegin = 0.5f - lengthB / 2.0f;
                dEnd = 0.5f + lengthE / 2.0f;
            }
            else if (SItem.equals("Weighted MA Chart"))
            {
                float length = ((float)Globals.vWMACoord.size()
                    / (float)Globals.vCoord.size());
                dBegin = 1.0f - length;
            }
        }

        float dLeft = (float)(Globals.cLeftChart);
        float dBottom = (float)(Globals.cTopChart + Globals.cHeightChart);
    float dTop = (float)(Globals.cTopChart);
    float dRight = (float)(Globals.cLeftChart + Globals.cWidthChart);

    // update timeline position with graph begin/end positions
    float dLength = dRight - dLeft;
    dRight = dLeft + dLength * dEnd;
    dLeft += (dLength * dBegin);

    // clear old timeline
    if (Globals.timelinePos ≠ -1 && Globals.bClearOldTimeline)
    {
        if (dBegin == 0.0f && Globals.timelinePos == dLeft)
        {
            g2d.setPaintMode();
            g2d.setColor(Color.black);

            // clear previous timeline
            timeline = new Line2D.Float(Globals.timelinePos, dBottom,
                Globals.timelinePos, dTop);
            g2d.draw(timeline);

            g2d.setXORMode(Color.lightGray);
            g2d.setColor(Color.cyan);
        }
        else
        {
            // clear previous timeline
            timeline = new Line2D.Float(Globals.timelinePos, dBottom,
                Globals.timelinePos, dTop);
            g2d.draw(timeline);
        }
    }

    if (drawNewOne)
    {
        // compute new timeline position
        Globals.timelinePos = dLeft + (((float)Globals.slSequence.getValue()
            / 100.0f) * (dRight -
        dLeft));

        if (dBegin == 0.0f && Globals.timelinePos == dLeft)
        {
            g2d.setPaintMode();
            g2d.setColor(Color.cyan);
        }

        // draw new timeline
        timeline = new Line2D.Float(Globals.timelinePos, dBottom,
            Globals.timelinePos, dTop);
        g2d.draw(timeline);
    }
    else
    {
        Globals.timelinePos = -1;
    }

    g2d.setPaintMode();
}
```

```
public void stateChanged(ChangeEvent e)
{
    JSlider source = (JSlider)e.getSource();

    if (!source.getValueIsAdjusting() && source == Globals.slNumValuesX)
    {
        Globals.cNumberValuesX = (int)source.getValue() - 2;
    }
    else if (!source.getValueIsAdjusting() && source == Globals.slNumValuesY)
    {
        Globals.cNumberValuesY = (int)source.getValue() - 2;
    }
    else if (source == Globals.slOrder)
    {
        int itmp = source.getValue();
        String Stmp = new String();
        Stmp = String.valueOf(itmp);
        Globals.lValOrder.setText(Stmp);
    }
    else if (source == Globals.slPercentage)
    {
        int itmp3 = source.getValue();
        String Stmp3 = new String();
        Stmp3 = String.valueOf(itmp3);
        Globals.lValPercentage.setText(Stmp3);
    }
    else if (source == Globals.slCoeff)
    {
        int itmp2 = source.getValue();
        String Stmp2 = new String();
        Stmp2 = String.valueOf(itmp2);
        Globals.lValCoeff.setText(Stmp2);
    }
    else if (source == Globals.slTime)
    {
        int itmp3 = source.getValue();
        String Stmp3 = new String();
        Stmp3 = String.valueOf(itmp3);
        Globals.lTime.setText(Stmp3 + " ms");
    }
    else if (source == Globals.slMinNote)
    {
        int itmp4 = source.getValue();
        String Stmp4 = new String();
        Stmp4 = String.valueOf(itmp4);
        Globals.lValMinNote.setText(Stmp4);
    }
    else if (source == Globals.slMaxNote)
    {
        int itmp5 = source.getValue();
        String Stmp5 = new String();
        Stmp5 = String.valueOf(itmp5);
        Globals.lValMaxNote.setText(Stmp5);
    }
    else if (source == Globals.slSequence)
    {
        if (Globals.slSequence.isEnabled())
        {
                if (Globals.slSequence.getValue() < 100)
                {
                    Globals.bReset.setEnabled(true);
                    Globals.bRew.setEnabled(true);
                    Globals.bPlay.setEnabled(true);
                    Globals.bFfw.setEnabled(true);
                    Globals.bEnd.setEnabled(true);
                    if (Globals.slSequence.getValue() ≤ 0) Sonification.reset();
                }
                else
                {
                    // stop sonification when at end of slider
                    Sonification.end();
                }
            }

            // update timeline position on graph
            updateTimeline(true);

            Globals.bClearOldTimeline = true;
        }
    }
}
```

Listener.java

The goal of the `Listener` class is simple: to detect when a button is pressed and to correctly handle the consequences. This happens in the `actionPerformed` method. The other methods are used to enable or disable some graphical elements from the user interface at certain periods. For example, the sonification-dependent elements are disabled as long as no chart has been created.

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.border.*;
import java.util.Vector;
import javax.swing.JFileChooser.*;
import java.io.*;


class Listener implements ActionListener
{
    // minimum number of points needed to enable chart drawing
    private final static int MinimumPointsNeeded = 3;

    private void enableChartDrawing()
    {
        Globals.bDrawMainChart.setEnabled(true);
        Globals.cbAlgorithm.setEnabled(true);
        Globals.slOrder.setEnabled(true);
        Globals.slPercentage.setEnabled(true);
        Globals.slCoeff.setEnabled(true);
        Globals.bDrawSelAlg.setEnabled(true);
    }


    private void disableChartDrawing()
    {
        Globals.bDrawMainChart.setEnabled(false);
        Globals.cbAlgorithm.setEnabled(false);
        Globals.slOrder.setEnabled(false);
        Globals.slPercentage.setEnabled(false);
        Globals.slCoeff.setEnabled(false);
        Globals.bDrawSelAlg.setEnabled(false);
    }


    private void enableSonification()
    {
        Sonification.openMidiSequencer();

        Globals.cbProgram.setEnabled(true);
        Globals.slMinNote.setEnabled(true);
        Globals.slMaxNote.setEnabled(true);
        Globals.slTime.setEnabled(true);
        Globals.bOptions.setEnabled(true);
        Globals.chbExtremeValues.setEnabled(true);
        Globals.chbDrumBeats.setEnabled(true);
        Globals.chbStereo.setEnabled(true);
        if (Globals.chbExtremeValues.isSelected()) Globals.cbExtremeProgram.setEnabled(true);
        if (Globals.chbDrumBeats.isSelected()) Globals.cbDrumProgram.setEnabled(true);
        Globals.cbChart.setEnabled(true);
        Globals.rbLinearMapping.setEnabled(true);
        Globals.rbChromaticMapping.setEnabled(true);
        Globals.slSequence.setEnabled(true);
        Globals.bReset.setEnabled(false);
        Globals.bRew.setEnabled(false);
        Globals.bPlay.setEnabled(true);
        Globals.bPause.setEnabled(false);
        Globals.bStop.setEnabled(false);
        Globals.bFfw.setEnabled(true);
        Globals.bEnd.setEnabled(true);
    }

    private void disableSonification()
    {
        Sonification.closeMidiSequencer();

        Globals.cbProgram.setEnabled(false);
        Globals.slMinNote.setEnabled(false);
        Globals.slMaxNote.setEnabled(false);
        Globals.slTime.setEnabled(false);
        Globals.bOptions.setEnabled(false);
        Globals.chbExtremeValues.setEnabled(false);
        Globals.chbDrumBeats.setEnabled(false);
        Globals.chbStereo.setEnabled(false);
        Globals.cbExtremeProgram.setEnabled(false);
        Globals.cbDrumProgram.setEnabled(false);
        Globals.cbChart.setEnabled(false);
        Globals.rbLinearMapping.setEnabled(false);
        Globals.rbChromaticMapping.setEnabled(false);
        Globals.slSequence.setValue(0);
        Globals.slSequence.setEnabled(false);
        Globals.bReset.setEnabled(false);
        Globals.bRew.setEnabled(false);
        Globals.bPlay.setEnabled(false);
        Globals.bPause.setEnabled(false);
        Globals.bStop.setEnabled(false);
        Globals.bFfw.setEnabled(false);
        Globals.bEnd.setEnabled(false);
    }

    private boolean exist(String s, JComboBox cb)
    {
        int num = cb.getItemCount();

        for (int i = 0; i < num; i++)
        {
            if (s.equals((String)(cb.getItemAt(i)))) return true;
        }

        return false;
```

```
}

public void actionPerformed(ActionEvent evt)
{
    // get event's source object
    Object source = evt.getSource();

    |                INSERT VALUE BUTTON                |

    if (source == Globals.bInsert)
    {
        int i;
        double el2;
        boolean inserted;
        int size = Globals.vCoord.size();
        Vector vData = new Vector();

        // get textfield values
        String sX = Globals.tfx.getText().trim();
        String sY = Globals.tfy.getText().trim();

        // check data length
        if (sX.length() == 0 && sY.length() == 0) return;

        // check data format
        try
        {
            el2 = Double.parseDouble(sX);
            double temp = Double.parseDouble(sY);
        }
        catch (NumberFormatException ex)
        {
            Globals.tfx.setText("");
            Globals.tfy.setText("");
            Globals.taStatus.append("\nTRYING TO INSERT INVALID VALUES IN MAIN CHART ->
                IGNORED\n");
            return;
        }

        // create vector with 2 values
        vData.add(sX);
        vData.add(sY);

        /** table vector must remain in ascending order for X **/

        i = 0;
        inserted = false;
        while ((i < size) && (!inserted))
        {
            Vector vPoint = (Vector)(Globals.vCoord.elementAt(i));

            double el1 = Double.parseDouble((String)(vPoint.elementAt(0)));

            if (el1 == el2)
            {
                // replace current (X,Y) value with new one
                Globals.vCoord.setElementAt(vData, i);
                inserted = true;
        }

        if (el1 > el2)
        {
            Globals.vCoord.add(i, vData);
            inserted = true;
        }

        i++;
    }

    // if not yet inserted, insert vector at end of vCoord
    if (!inserted) Globals.vCoord.add(vData);

    Globals.taStatus.append("\n Coordinate (" + Globals.tfx.getText()
        + "," + Globals.tfy.getText() + ") inserted");

    // update smoothing window settings (for MA and WMA)
    size = Globals.vCoord.size();
    if (size > 5) Globals.slOrder.setMaximum((size / 2) - 2);
    Globals.slCoeff.setMaximum(size / 2);

    // refresh main chart table
    Globals.tCoord.revalidate();
    Globals.tCoord.repaint();

    // enable chart drawing if table contains enough points
    if (size ≥ MinimumPointsNeeded) enableChartDrawing();
    Globals.bClearTable.setEnabled((size ≠ 0));
}
```

DELETE SELECTED LINE BUTTON

```
else if (source == Globals.bDelete)
{
    // get selected row
    int line = Globals.tCoord.getSelectedRow();

    // delete selected row from table and from corresponding vector
    if ((line ≥ 0) && (line < Globals.vCoord.size()))
    {
        Globals.tCoord.clearSelection();
        Globals.vCoord.removeElementAt(line);
        Globals.tCoord.revalidate();
        Globals.tCoord.repaint();
    }

    // disable chart drawing if table does not contain enough points
    Globals.bDelete.setEnabled(false);
    if (Globals.vCoord.size() < MinimumPointsNeeded) disableChartDrawing();
    if (Globals.vCoord.size() == 0) Globals.bClearTable.setEnabled(false);
}
```

CLEAR TABLE BUTTON

```
else if (source == Globals.bClearTable)
{
    // delete table and vector
    Globals.vCoord.clear();
```

```
        Globals.tCoord.revalidate();
        Globals.tCoord.repaint();
        Globals.tCoord.clearSelection();

        // disable chart drawing and sonification
        Globals.bDelete.setEnabled(false);
        Globals.bClearTable.setEnabled(false);
        disableChartDrawing();
        disableSonification();
}
```

IMPORT FROM FILE... BUTTON

```
else if (source == Globals.bImport)
{
        String sPath = new String();
        String sDirectory = new String();
        String sFileName = new String();

        // create a file chooser
        final JFileChooser fc = new JFileChooser();
        int returnVal = fc.showOpenDialog(null);

        if (returnVal == JFileChooser.APPROVE_OPTION)
        {
                // get complete file name
                sDirectory = (fc.getCurrentDirectory()).getAbsolutePath();
                sFileName = (fc.getSelectedFile()).getName();
                sPath = (sDirectory + "\\" + sFileName);
                Globals.taStatus.append("\n File opened : " + sPath);
        }
        else
        {
                Globals.taStatus.append("\n Open command cancelled by user.");
                return;
        }

        // create buffer reader with complete file name
        ChartFile.createBufferReader(sPath);
        int i = 0, j = 0;
        boolean inserted;

        // initialize table vector
        Globals.vCoord.clear();

        try
        {
                Globals.taStatus.append("\n Reading file...");

                while ((Globals.sCurrentLine = ChartFile.in.readLine()) ≠ null)
                {
                        // read single line in file (containing X—Y)
                        // and store results in ChartFile.readX and ChartFile.readY
                        ChartFile.readData(Globals.sCurrentLine);

                        Double dReadX = new Double(ChartFile.readX);
                        Double dReadY = new Double(ChartFile.readY);

                        Vector vData = new Vector();
                vData.add(dReadX.toString());
                vData.add(dReadY.toString());

                /** table vector must remain in ascending order for X **/

                j = 0;
                inserted = false;
                double el2 = Double.parseDouble((String)(vData.elementAt(0)));
                int size = Globals.vCoord.size();

                while ((j < size) && (!inserted))
                {
                    Vector vPoint = (Vector)(Globals.vCoord.elementAt(j));

                    double el1 = Double.parseDouble((String)(vPoint.elementAt(0)));

                    if (el1 == el2)
                    {
                        // replace current (X,Y) value with new one
                        Globals.vCoord.setElementAt(vData, j);
                        inserted = true;
                    }

                    if (el1 > el2)
                    {
                        Globals.vCoord.add(j, vData);
                        inserted = true;
                    }

                    j++;
                }

                if (!inserted) Globals.vCoord.add(vData);

                i++;
            }

            Globals.taStatus.append("done. < Total : " + i +" points >");
        }
        catch (IOException ex)
        {
            Globals.taStatus.append("\nIO EXCEPTION WHILE READING FILE...\n");
            Globals.vCoord.clear();
        }
        catch (NumberFormatException ex)
        {
            Globals.taStatus.append("\nNUMBER FORMAT EXCEPTION WHILE READING FILE...\n");
            Globals.vCoord.clear();
        }

        // update smoothing window settings (for MA and WMA)
        int newsize = Globals.vCoord.size();
        if (newsize > 5) Globals.slOrder.setMaximum((newsize / 2) - 2);
        Globals.slCoeff.setMaximum(newsize / 2);

        // refresh table
        Globals.tCoord.revalidate();
        Globals.tCoord.repaint();
        Globals.tCoord.clearSelection();
```

```
    // does table contain enough points for chart drawing?
    if (newsize > MinimumPointsNeeded) enableChartDrawing();
    else disableChartDrawing();
    Globals.bClearTable.setEnabled((newsize ≠ 0));
    Globals.bDelete.setEnabled(false);
}
```

DRAW MAIN CHART BUTTON

```
else if (source == Globals.bDrawMainChart)
{
    // chart will be drawn, so set bFirstTime to false
    Globals.bFirstTime = false;

    // draw main chart
    DrawInitChart.drawChart();

    // add corresponding string to chart combobox in sonification panel
    if (!(exist("Main Chart", Globals.cbChart)))
    {
        Globals.cbChart.addItem("Main Chart");
        enableSonification();
    }

    Globals.bHideMainChart.setEnabled(true);
    Globals.bClearOldTimeline = false;
}
```

HIDE CHART BUTTON

```
else if (source == Globals.bHideMainChart)
{
    // remove corresponding string from chart combobox in sonification panel
    Globals.cbChart.removeItem("Main Chart");
    if (Globals.cbChart.getItemCount() == 0) disableSonification();

    // clear main chart
    Globals.gp.reset();
    Globals.pChart.repaint();

    Globals.bHideMainChart.setEnabled(false);
    Globals.bClearOldTimeline = false;
}
```

DRAW LINEAR MOVING AVERAGE BUTTON

```
else if (source == Globals.bDrawSelAlg && (Globals.cbAlgorithm.getSelectedIndex()
    == 0))
{
    // chart will be drawn, so set bFirstTime to false
    Globals.bFirstTime = false;

    // get smoothing window
    int iMAOrder = (new Integer(Globals.lValOrder.getText())).intValue();

    // compute linear moving average
    Globals.vMACoord = Algorithm.movingAverage(Globals.vCoord, iMAOrder);
    Globals.taStatus.append("\n ---------- Linear Moving Average serie : information
        ----------");
    Globals.taStatus.append("\n Size  : "+ Globals.vMACoord.size() +" points.");
    Globals.taStatus.append("\n Order : "+ Globals.lValOrder.getText());

    // draw linear moving average chart
    DrawInitChart.drawMovingAverage();

    // add corresponding string to chart combobox in sonification panel
    if (!(exist("Linear MA Chart", Globals.cbChart)))
    {
        Globals.cbChart.addItem("Linear MA Chart");
        enableSonification();
    }

    Globals.bHideAlgMA.setEnabled(true);
    Globals.bClearOldTimeline = false;
}
```

DRAW EXPONENTIAL MOVING AVERAGE BUTTON

```
else if (source == Globals.bDrawSelAlg && (Globals.cbAlgorithm.getSelectedIndex()
    == 1))
{
    // chart will be drawn, so set bFirstTime to false
    Globals.bFirstTime = false;

    // get smoothing window
    int iEMAPercentage = (new Integer(Globals.lValPercentage.getText())).intValue();

    // compute exponential moving average
    Globals.vEMACoord = Algorithm.expMovingAverage(Globals.vCoord, iEMAPercentage);

    Globals.taStatus.append(
    "\n ---------- Exponential Moving Average serie : information ----------");
    Globals.taStatus.append("\n Size  : "+ Globals.vEMACoord.size() +" points.");
    Globals.taStatus.append("\n Percentage : "+ Globals.lValPercentage.getText());

    // draw exponential moving average chart
    DrawInitChart.drawExpMovingAverage();

    // add corresponding string to chart combobox in sonification panel
    if (!(exist("Exponential MA Chart", Globals.cbChart)))
    {
        Globals.cbChart.addItem("Exponential MA Chart");
        enableSonification();
    }

    Globals.bHideAlgEMA.setEnabled(true);
    Globals.bClearOldTimeline = false;
}
```

DRAW WEIGTHED MOVING AVERAGE BUTTON

```
else if (source == Globals.bDrawSelAlg && (Globals.cbAlgorithm.getSelectedIndex()
    == 2))
{
    // chart will be drawn, so set bFirstTime to false
    Globals.bFirstTime = false;
```

```
        // get smoothing window
        int iWMACoeff = (new Integer(Globals.lValCoeff.getText())).intValue();

        // compute weighted moving average
        Globals.vWMACoord = Algorithm.weightedMovingAverage(Globals.vCoord, iWMACoeff);

        Globals.taStatus.append(
        "\n ---------- Weighted Moving Average serie : information ----------");
        Globals.taStatus.append("\n Size   : "+ Globals.vWMACoord.size() +" points.");
        Globals.taStatus.append("\n Order  : "+ Globals.lValCoeff.getText());

        // draw weighted moving average chart
        DrawInitChart.drawWeightedMovingAverage();

        // add corresponding string to chart combobox in sonification panel
        if (!(exist("Weighted MA Chart", Globals.cbChart)))
        {
            Globals.cbChart.addItem("Weighted MA Chart");
            enableSonification();
        }

        Globals.bHideAlgWMA.setEnabled(true);
        Globals.bClearOldTimeline = false;
    }
```

Hide Linear Moving Average BUTTON

```
    else if (source == Globals.bHideAlgMA)
    {
        // remove corresponding string from chart combobox in sonification panel
        Globals.cbChart.removeItem("Linear MA Chart");
        if (Globals.cbChart.getItemCount() == 0) disableSonification();

        // clear linear moving average chart
        Globals.gpMA.reset();
        Globals.pChart.repaint();

        Globals.bHideAlgMA.setEnabled(false);
        Globals.bClearOldTimeline = false;
    }
```

Hide Exponential Moving Average BUTTON

```
    else if (source == Globals.bHideAlgEMA)
    {
        // remove corresponding string from chart combobox in sonification panel
        Globals.cbChart.removeItem("Exponential MA Chart");
        if (Globals.cbChart.getItemCount() == 0) disableSonification();

        // clear exponential moving average chart
        Globals.gpEMA.reset();
        Globals.pChart.repaint();

        Globals.bHideAlgEMA.setEnabled(false);
        Globals.bClearOldTimeline = false;
    }
```

HIDE WEIGHTED MOVING AVERAGE BUTTON

```
else if (source == Globals.bHideAlgWMA)
{
    // remove corresponding string from chart combobox in sonification panel
    Globals.cbChart.removeItem("Weighted MA Chart");
    if (Globals.cbChart.getItemCount() == 0) disableSonification();

    // clear weighted moving average chart
    Globals.gpWMA.reset();
    Globals.pChart.repaint();

    Globals.bHideAlgWMA.setEnabled(false);
    Globals.bClearOldTimeline = false;
}
```

ALGORITHM CHECKBOX

```
else if (source == Globals.cbAlgorithm)
{
    // linear moving average
    if (Globals.cbAlgorithm.getSelectedIndex() == 0)
    {
        Globals.slCoeff.setVisible(false);
        Globals.lCoeff.setVisible(false);
        Globals.lValCoeff.setVisible(false);

        Globals.slPercentage.setVisible(false);
        Globals.lPercentage.setVisible(false);
        Globals.lValPercentage.setVisible(false);

        Globals.slOrder.setVisible(true);
        Globals.lOrder.setVisible(true);
        Globals.lValOrder.setVisible(true);
    }

    // exponential moving average
    if (Globals.cbAlgorithm.getSelectedIndex() == 1)
    {
        Globals.slOrder.setVisible(false);
        Globals.lOrder.setVisible(false);
        Globals.lValOrder.setVisible(false);

        Globals.slPercentage.setVisible(true);
        Globals.lPercentage.setVisible(true);
        Globals.lValPercentage.setVisible(true);

        Globals.slCoeff.setVisible(false);
        Globals.lCoeff.setVisible(false);
        Globals.lValCoeff.setVisible(false);
    }

    // weighted moving average
    if (Globals.cbAlgorithm.getSelectedIndex() == 2)
    {
        Globals.slOrder.setVisible(false);
        Globals.lOrder.setVisible(false);
        Globals.lValOrder.setVisible(false);
```

```
            Globals.slPercentage.setVisible(false);
            Globals.lPercentage.setVisible(false);
            Globals.lValPercentage.setVisible(false);

            Globals.slCoeff.setVisible(true);
            Globals.lCoeff.setVisible(true);
            Globals.lValCoeff.setVisible(true);
        }
}
```

SONIFICATION BUTTONS

```
else if (source == Globals.bOptions)
{
    // show sonification options window
    Globals.bOptions.setEnabled(false);
    Globals.optionsFrame.show();
}

else if (source == Globals.chbExtremeValues)
{
    Globals.cbExtremeProgram.setEnabled(Globals.chbExtremeValues.isSelected());
}

else if (source == Globals.chbDrumBeats)
{
    Globals.cbDrumProgram.setEnabled(Globals.chbDrumBeats.isSelected());
}

else if (source == Globals.bReset)
{
    Sonification.reset();
}

else if (source == Globals.bRew)
{
    Sonification.rew();
}

else if (source == Globals.bPlay)
{
    Sonification.play();
}

else if (source == Globals.bPause)
{
    Sonification.pause();
}

else if (source == Globals.bStop)
{
    Sonification.stop();
}

else if (source == Globals.bFfw)
{
    Sonification.ffw();
}
```

```
        else if (source == Globals.bEnd)
        {
            Sonification.end();
        }
    }
}
```

ChartPanel.java

The ChartPanel class firstly defines some objects (defined in the Globals class) used for drawing charts. The paintComponent method is called every time a chart must be redrawn. It draws not only the specified chart (the main chart or a moving average chart), but also the chart axes and values. The formatValue method is called by paintComponent to format chart values to the desired format, using the specified number of decimals.

```
import javax.swing.*;
import javax.swing.border.*;
import javax.swing.JFileChooser.*;
import java.awt.event.*;
import java.awt.*;
import java.awt.Graphics.*;
import java.awt.geom.*;
import java.util.Vector;
import java.lang.Double;
import java.io.*;


public class ChartPanel extends JPanel
{
    public Graphics2D getGraphics2D()
    {
        // return parent's graphics object
        return (Graphics2D)getGraphics();
    }


    public void paintComponent(Graphics g)
    {
        // firstly clear the existing chart
        clear(g);

        // initialize paint environment and set color to black
        Graphics2D g2d = (Graphics2D)g;
        g2d.setColor(Color.black);

        // get the desired number of values on each axis
        // (min and max are always displayed)
        Globals.cNumberValuesX = Globals.slNumValuesX.getValue() - 2;
        Globals.cNumberValuesY = Globals.slNumValuesY.getValue() - 2;

        // bFirstTime is only true until a Draw button has been pushed
        if (!Globals.bFirstTime)
        {
            // draw min and max values along the X axis
            g2d.drawString(formatValue((new Double(Globals.dMinX)).toString()),
                (float)(Globals.axeX.getX1() - 5), (float)(Globals.axeX.getY1() + 20));
            g2d.drawString(formatValue((new Double(Globals.dMaxX)).toString()),
                (float)(Globals.axeX.getX2() - 5), (float)(Globals.axeX.getY2() + 20));
```

```
// draw min and max values along the Y axis
g2d.drawString(formatValue((new Double(Globals.dMinY)).toString()),
    (float)(Globals.axeY.getX1() - 50), (float)(Globals.axeY.getY1() + 5));
g2d.drawString(formatValue((new Double(Globals.dMaxY)).toString()),
    (float)(Globals.axeY.getX2() - 50), (float)(Globals.axeY.getY2() + 5));

/** draw the other values **/

int i;
String val;
float fX, fY;
float fX1, fY1, fX2, fY2;

// distance between 2 values on each axis
double dValIntAxeX = (Globals.dMaxX - Globals.dMinX)
    / (Globals.cNumberValuesX + 1);
double dValIntAxeY = (Globals.dMaxY - Globals.dMinY)
    / (Globals.cNumberValuesY + 1);

// X axis
for (i = 1; i ≤ Globals.cNumberValuesX; i++)
{
    // draw the value in black
    g2d.setColor(Color.black);
    val = formatValue((new Double(Globals.dMinX + (i * dValIntAxeX))).toString());
    fX = (float)(Globals.axeX.getX1() + (dValIntAxeX * Globals.dIntX * i));
    fY = (float)(Globals.axeX.getY1() + 20);
    g2d.drawString(val, fX - 5, fY);

    // draw the grid in gray
    g2d.setColor(new Color(166, 166, 166));
    Globals.gridXY.setLine(fX, fY - 20, fX, (float)(Globals.cTopChart));
    g2d.draw(Globals.gridXY);
}

// draw last gridline on X axis
fX = (float)(Globals.axeX.getX1() + (dValIntAxeX * Globals.dIntX * i));
fY = (float)(Globals.axeX.getY1() + 20);
Globals.gridXY.setLine(fX, fY - 20, fX, (float)(Globals.cTopChart));
g2d.draw(Globals.gridXY);

// Y axis
for (i = 1; i ≤ Globals.cNumberValuesY; i++)
{
    // draw the value in black
    g2d.setColor(Color.black);
    val = formatValue((new Double(Globals.dMinY + (i * dValIntAxeY))).toString());
    fX = (float)(Globals.axeY.getX1() - 50);
    fY = (float)(Globals.axeY.getY1() - (dValIntAxeY * Globals.dIntY * i));
    g2d.drawString(val, fX, fY + 5);

    // draw the grid in gray
    g2d.setColor(new Color(166, 166, 166));
    Globals.gridXY.setLine((float)(Globals.cLeftChart), fY,
        (float)(Globals.cLeftChart + Globals.cWidthChart), fY);
    g2d.draw(Globals.gridXY);
}
```

```
            // draw last gridline on Y axis
            fX = (float)(Globals.axeY.getX1() - 50);
            fY = (float)(Globals.axeY.getY1() - (dValIntAxeY * Globals.dIntY * i));
            Globals.gridXY.setLine((float)(Globals.cLeftChart), fY,
                (float)(Globals.cLeftChart + Globals.cWidthChart), fY);
            g2d.draw(Globals.gridXY);

            // draw origin X lines
            if (Globals.dMinX ≤ 0)
            {
                fX1 = (float)(Globals.axeX.getX1() - (Globals.dMinX * Globals.dIntX));
                fY1 = (float)(Globals.axeX.getY1());
                fX2 = fX1;
                fY2 = (float)(Globals.cTopChart);

                // draw lines in dark green
                g2d.setColor(new Color(0, 110, 0));
                Globals.originX.setLine(fX1, fY1, fX2, fY2);
                g2d.draw(Globals.originX);
            }

            // draw origin Y lines
            if (Globals.dMinY ≤ 0)
            {
                fX1 = (float)(Globals.cLeftChart);
                fY1 = (float)(Globals.axeY.getY1() + (Globals.dMinY * Globals.dIntY));
                fX2 = (float)(Globals.cLeftChart + Globals.cWidthChart);
                fY2 = fY1;

                // draw lines in dark green
                g2d.setColor(new Color(0,110,0));
                Globals.originY.setLine(fX1, fY1, fX2, fY2);
                g2d.draw(Globals.originY);
            }
        }

        // draw the axes in black
        g2d.setColor(Color.black);
        g2d.draw(Globals.axeX);
        g2d.draw(Globals.axeY);

        /** finally draw the charts **/

        // main chart in red
        g2d.setColor(Color.red);
        g2d.draw(Globals.gp);

        // linear moving average in blue
        g2d.setColor(Color.blue);
        g2d.draw(Globals.gpMA);

        // exponential moving average in yellow
        g2d.setColor(new Color(248,248,0));
        g2d.draw(Globals.gpEMA);

        // weighted moving average in green
        g2d.setColor(new Color(0,128,128));
        g2d.draw(Globals.gpWMA);
    }
```

```
protected void clear(Graphics g)
{
    // call clear method of parent (JPanel)
    super.paintComponent(g);
}


protected String formatValue(String str)
{
    int index = str.indexOf(".");
    try
    {
        Integer INumDec = new Integer(Globals.tfNumDecimals.getText());
        Globals.cNdecimalsVal = INumDec.intValue();
    }
    catch (NumberFormatException ex)
    {
        // reset value to 2
        Globals.tfNumDecimals.setText("2");
        Globals.cNdecimalsVal = 2;
        index = -1;
        Globals.taStatus.append("\nWARNING: Number format exception while reading number
            of decimals\n");
        Globals.taStatus.append("Resetted number of decimals to 2\n");
    }

    // if no '.' is present, it's an integer so return the same string
    if (index == -1) return str;
    else
    {
        // get number of digits after the decimal point
        int iCharAfterPoint = str.length() - index - 1;

        // format the string
        if (iCharAfterPoint ≤ Globals.cNdecimalsVal) return str;
        else
        {
            String val = str.toString();
            val = val.substring(0, index + 1) + val.substring(index +
                1, index + Globals.cNdecimalsVal + 1);
            return val;
        }
    }
}


public ChartPanel()
{
    // save current panel
    Globals.chartPanel = this;

    // initialize pChart and add it to the panel
    Globals.pChart = new JPanel();
    Globals.pChart.setOpaque(false);
    Globals.pChart.setLayout(null);
    Globals.pChart.setBounds(10,10,Globals.screenWidth - 30,
        Globals.screenHeight / 2 - 25);
        Border bEtched = BorderFactory.createEtchedBorder();
        Border bTitle = BorderFactory.createTitledBorder(bEtched, "");
        Globals.pChart.setBorder(bTitle);
        add(Globals.pChart);

        // create chart objects
        Globals.gp = new GeneralPath();
        Globals.gpMA = new GeneralPath();
        Globals.gpEMA = new GeneralPath();
        Globals.gpWMA = new GeneralPath();
        Globals.axeX = new Line2D.Double();
        Globals.axeY = new Line2D.Double();
        Globals.gridXY = new Line2D.Float();
        Globals.originX = new Line2D.Float();
        Globals.originY = new Line2D.Float();
    }
}
```

ChartFile.java

The two methods in the `ChartFile` class are called from the `Listener` class when the user opens a SoundChart file. The first method reads one line from the file, and stores the X and Y values in the corresponding variables. The second method is only called once, since it initializes the file pointer which will be used to access the file denoted by `sPath`.

```
import javax.swing.*;
import java.awt.*;
import java.util.StringTokenizer;
import java.util.*;
import java.io.*;
import java.lang.Double;


public class ChartFile
{
    public static void readData(String s)
    {
        StringTokenizer t = new StringTokenizer(s, "|");

        readX = Double.parseDouble(t.nextToken());
        readY = Double.parseDouble(t.nextToken());
    }


    public static void createBufferReader(String sPath)
    {
        try
        {
            in = new BufferedReader(new FileReader(sPath));
        }
        catch (IOException e)
        {
            System.out.println("ERROR in createBufferReader(): " + e);
            System.exit(0);
        }
    }

    public static double readX, readY;
    public static BufferedReader in;
}
```

DrawInitChart.java

The `DrawInitChart` class contains a method to define the scale of the chart and five drawing methods. In fact these methods don't draw anything, but fill the objects that are later used by `ChartPanel` for the actual drawing. These objects are `Globals.axeX` and `Globals.axeY` for the grid, `Globals.gp` for the main chart, `Globals.gpMA` for the linear moving average chart, `Globals.gpEMA` for the exponential moving average chart, and `Globals.gpWMA` for the weighted moving average chart. The grid-drawing method is private, since it is called by the other drawing methods.

```
import javax.swing.*;
import javax.swing.border.*;
import javax.swing.JFileChooser.*;
import java.awt.event.*;
import java.awt.*;
import java.awt.Graphics.*;
import java.awt.geom.*;
import java.util.Vector;
import java.lang.Double;
import java.io.*;


public class DrawInitChart
{
    private static void findScale()
    {
        int i;
        double dTemp;
        Vector vTemp;

        Globals.taStatus.append("\n ----- Scale information -----");
        Globals.taStatus.append("\n Table size : " + Globals.vCoord.size());

        // find min X and max X values in main chart table
        vTemp = (Vector)(Globals.vCoord.elementAt(0));
        Globals.dMinX = Double.parseDouble( (String)(vTemp.elementAt(0)) );
        vTemp = (Vector)(Globals.vCoord.elementAt(Globals.vCoord.size() - 1));
        Globals.dMaxX = Double.parseDouble( (String)(vTemp.elementAt(0)) );

        Globals.taStatus.append("\n Min X : " + Globals.dMinX);
        Globals.taStatus.append("\n Max X : " + Globals.dMaxX);

        // find min Y and max Y values in main chart table
        Globals.dMinY = Double.MAX_VALUE;
        Globals.dMaxY = Double.MIN_VALUE;
        vTemp = (Vector)(Globals.vCoord.elementAt(0));
        dTemp = Double.parseDouble( (String)(vTemp.elementAt(1)) );
        for (i = 0; i < Globals.vCoord.size(); i++)
        {
            vTemp = (Vector)(Globals.vCoord.elementAt(i));
            dTemp = Double.parseDouble( (String)(vTemp.elementAt(1)) );
            if (dTemp < Globals.dMinY) Globals.dMinY = dTemp;
            if (dTemp > Globals.dMaxY) Globals.dMaxY = dTemp;
        }

        Globals.taStatus.append("\n Min Y : " + Globals.dMinY);
        Globals.taStatus.append("\n Max Y : " + Globals.dMaxY);

        // get dimension of the chart (pChart panel)
        Globals.cLeftChart = Globals.pChart.getX() + 70;
        Globals.cTopChart = Globals.pChart.getY() + 20;
        Globals.cWidthChart = Globals.pChart.getWidth() - 110;
        Globals.cHeightChart = Globals.pChart.getHeight() - 50;

        // find X and Y interval
        Globals.dIntX = (Globals.cWidthChart) / (Globals.dMaxX - Globals.dMinX);
        Globals.dIntY = (Globals.cHeightChart) / (Globals.dMaxY - Globals.dMinY);

        Globals.taStatus.append("\n Interval X : " + Globals.dIntX);
        Globals.taStatus.append("\n Interval Y : " + Globals.dIntY);
        Globals.taStatus.append("\n -----------------------------");
    }


    private static void drawGrid(float fBottomChart, float fLeftChart)
    {
        // get bounds for X and Y axis
        double dLeft = (double)(Globals.cLeftChart);
        double dBottom = (double)(Globals.cTopChart + Globals.cHeightChart);
        double dTop = (double)(Globals.cTopChart);
        double dRight = (double)(Globals.cLeftChart + Globals.cWidthChart);

        // create axes
        Globals.axeX.setLine(dLeft, dBottom, dRight, dBottom);
        Globals.axeY.setLine(dLeft, dBottom, dLeft, dTop);
    }


    public static void drawChart()
    {
        int i;
        float fTempX, fTempY, fDistFromLeft, fDistFromBottom;
        float fLeftChart = 0, fBottomChart = 0;

        // reset main chart graphic object
        Globals.gp.reset();
        Globals.pChart.repaint();
        (Globals.pChart.getParent()).repaint();

        // (re)draw grid
        findScale();
        fBottomChart = (float)(Globals.cTopChart + Globals.cHeightChart);
        fLeftChart = (float)(Globals.cLeftChart);
        drawGrid(fLeftChart, fBottomChart);

        /** draw the chart **/

        // move to the first point
        Vector vTemp = (Vector)(Globals.vCoord.elementAt(0));
        fTempX =(float)(Double.parseDouble( (String)(vTemp.elementAt(0)) ) );
        fTempY =(float)(Double.parseDouble( (String)(vTemp.elementAt(1)) ) );
        fDistFromBottom = (float)((fTempY - Globals.dMinY) * Globals.dIntY);
        fDistFromLeft = (float)((fTempX - Globals.dMinX) * Globals.dIntX);
        Globals.gp.moveTo(fLeftChart, fBottomChart - fDistFromBottom);

        // now draw lines from each point to the next one
        for (i = 1; i < Globals.vCoord.size(); i++)
        {
            vTemp = (Vector)(Globals.vCoord.elementAt(i));
            fTempX =(float)(Double.parseDouble( (String)(vTemp.elementAt(0)) ) );
            fTempY =(float)(Double.parseDouble( (String)(vTemp.elementAt(1)) ) );
            fDistFromBottom = (float)((fTempY - Globals.dMinY) * Globals.dIntY);
            fDistFromLeft = (float)((fTempX - Globals.dMinX) * Globals.dIntX);
            Globals.gp.lineTo(fLeftChart + fDistFromLeft, fBottomChart - fDistFromBottom);
        }

        // finally draw the chart
        Globals.pChart.repaint();
    }


    public static void drawMovingAverage()
    {
        int i;
        float fLeftChart = 0, fBottomChart = 0;
        float fTempX, fTempY, fDistFromLeft, fDistFromBottom;

        // reset linear moving average chart graphic object
        Globals.gpMA.reset();
        Globals.pChart.repaint();
        (Globals.pChart.getParent()).repaint();

        // (re)draw grid
        findScale();
        fBottomChart = (float)(Globals.cTopChart + Globals.cHeightChart);
        fLeftChart = (float)(Globals.cLeftChart);
        drawGrid(fLeftChart, fBottomChart);

        /** draw the linear moving average chart **/

        // move to the first point
        Vector vTemp = (Vector)(Globals.vMACoord.elementAt(0));
        fTempX =(float)(Double.parseDouble( (String)(vTemp.elementAt(0)) ) );
        fTempY =(float)(Double.parseDouble( (String)(vTemp.elementAt(1)) ) );
        fDistFromBottom = (float)((fTempY - Globals.dMinY) * Globals.dIntY);
        fDistFromLeft = (float)((fTempX - Globals.dMinX) * Globals.dIntX);
        Globals.gpMA.moveTo(fLeftChart + fDistFromLeft, fBottomChart - fDistFromBottom);

        // now draw lines from each point to the next one
        for (i = 1; i < Globals.vMACoord.size(); i++)
        {
            vTemp = (Vector)(Globals.vMACoord.elementAt(i));
            fTempX =(float)(Double.parseDouble( (String)(vTemp.elementAt(0)) ) );
            fTempY =(float)(Double.parseDouble( (String)(vTemp.elementAt(1)) ) );
            fDistFromBottom = (float)((fTempY - Globals.dMinY) * Globals.dIntY);
            fDistFromLeft = (float)((fTempX - Globals.dMinX) * Globals.dIntX);
            Globals.gpMA.lineTo(fLeftChart + fDistFromLeft, fBottomChart - fDistFromBottom);
        }
        // finally draw the chart
        Globals.pChart.repaint();
    }


    public static void drawExpMovingAverage()
    {
        int i;
        float fLeftChart = 0, fBottomChart = 0;
        float fTempX, fTempY, fDistFromLeft, fDistFromBottom;

        // reset exponential moving average chart graphic object
        Globals.gpEMA.reset();
        Globals.pChart.repaint();
        (Globals.pChart.getParent()).repaint();

        // (re)draw grid
        findScale();
        fBottomChart = (float)(Globals.cTopChart + Globals.cHeightChart);
        fLeftChart = (float)(Globals.cLeftChart);
        drawGrid(fLeftChart, fBottomChart);

        /** draw the exponential moving average chart **/

        // move to the first point
        Vector vTemp = (Vector)(Globals.vEMACoord.elementAt(0));
        fTempX =(float)(Double.parseDouble( (String)(vTemp.elementAt(0)) ) );
        fTempY =(float)(Double.parseDouble( (String)(vTemp.elementAt(1)) ) );
        fDistFromBottom = (float)((fTempY - Globals.dMinY) * Globals.dIntY);
        fDistFromLeft = (float)((fTempX - Globals.dMinX) * Globals.dIntX);
        Globals.gpEMA.moveTo(fLeftChart + fDistFromLeft, fBottomChart - fDistFromBottom);

        // now draw lines from each point to the next one
        for (i = 1; i < Globals.vEMACoord.size(); i++)
        {
            vTemp = (Vector)(Globals.vEMACoord.elementAt(i));
            fTempX =(float)(Double.parseDouble( (String)(vTemp.elementAt(0)) ) );
            fTempY =(float)(Double.parseDouble( (String)(vTemp.elementAt(1)) ) );
            fDistFromBottom = (float)((fTempY - Globals.dMinY) * Globals.dIntY);
            fDistFromLeft = (float)((fTempX - Globals.dMinX) * Globals.dIntX);
            Globals.gpEMA.lineTo(fLeftChart + fDistFromLeft, fBottomChart - fDistFromBottom);
        }

        // finally draw the chart
        Globals.pChart.repaint();
    }


    public static void drawWeightedMovingAverage()
    {
        int i;
        float fLeftChart = 0, fBottomChart = 0;
        float fTempX, fTempY, fDistFromLeft, fDistFromBottom;

        // reset exponential moving average chart graphic object
        Globals.gpWMA.reset();
        Globals.pChart.repaint();
        (Globals.pChart.getParent()).repaint();
        // (re)draw grid
        findScale();
        fBottomChart = (float)(Globals.cTopChart + Globals.cHeightChart);
        fLeftChart = (float)(Globals.cLeftChart);
        drawGrid(fLeftChart, fBottomChart);

        /** draw the weighted moving average **/

        // move to the first point
        Vector vTemp = (Vector)(Globals.vWMACoord.elementAt(0));
        try
        {
            fTempX =(float)(Double.parseDouble( (String)(vTemp.elementAt(0)) ) );
            fTempY =(float)(Double.parseDouble( (String)(vTemp.elementAt(1)) ) );
        }
        catch (NumberFormatException ex)
        {
            Globals.taStatus.append("\nNUMBER FORMAT EXCEPTION WHILE DRAWING PMA CHART ->
                IGNORING\n");
            Globals.pChart.repaint();
            return;
        }
        fDistFromBottom = (float)((fTempY - Globals.dMinY) * Globals.dIntY);
        fDistFromLeft = (float)((fTempX - Globals.dMinX) * Globals.dIntX);
        Globals.gpWMA.moveTo(fLeftChart + fDistFromLeft, fBottomChart - fDistFromBottom);

        // now draw lines from each point to the next one
        for (i = 1; i < Globals.vWMACoord.size(); i++)
        {
            vTemp = (Vector)(Globals.vWMACoord.elementAt(i));
            try
            {
                fTempX =(float)(Double.parseDouble( (String)(vTemp.elementAt(0)) ) );
                fTempY =(float)(Double.parseDouble( (String)(vTemp.elementAt(1)) ) );
            }
            catch (NumberFormatException ex)
            {
                Globals.taStatus.append("\nNUMBER FORMAT EXCEPTION WHILE DRAWING PMA CHART ->
                    IGNORING\n");
                break;
            }
            fDistFromBottom = (float)((fTempY - Globals.dMinY) * Globals.dIntY);
            fDistFromLeft = (float)((fTempX - Globals.dMinX) * Globals.dIntX);
            Globals.gpWMA.lineTo(fLeftChart + fDistFromLeft, fBottomChart - fDistFromBottom);
        }

        // finally draw the chart
        Globals.pChart.repaint();
    }
}
```

Algorithm.java

The `Algorithm` class contains three methods for computing the moving averages of the initial chart. Each method takes the initial chart values and outputs the corresponding moving average values, using the given parameter *k* as smoothing window.

```
import javax.swing.*;
import java.awt.*;
import java.util.StringTokenizer;
import java.util.*;
import java.io.*;
import java.lang.Double;


public class Algorithm
{
    LINEAR MOVING AVERAGE

    public static Vector movingAverage(Vector vInitChart, int k)
    {
        Vector vTemp = new Vector();
        Vector vTemp2 = new Vector();
        Vector vMA = new Vector();;
        float fTempX, fTempY;
        double dMh;

        int n = vInitChart.size() - 1;

        for (int h = k + 1; h ≤ n - k; h++)
        {
            // compute the sum
            float sum = 0.0f;
            for (int l = h - k; l ≤ h + k; l++)
            {
                vTemp = (Vector)(vInitChart.elementAt(l - 1));
                fTempY = (float)(Double.parseDouble( (String)(vTemp.elementAt(1)) ) );
                sum += fTempY;
            }
            dMh = (double)((1.0f / ( (2.0f * (float)k) + 1.0f)) * sum);

            Double DMh = new Double(dMh);

            vTemp2 = (Vector)(vInitChart.elementAt(h - 1));
            fTempX = (float)(Double.parseDouble ( (String)(vTemp2.elementAt(0)) ) );

            Float FTempX = new Float(fTempX);
            Vector vData = new Vector();

            vData.add(FTempX.toString());
            vData.add(DMh.toString());
            vMA.add(vData);
        }

        return vMA;
    }
```

EXPONENTIAL MOVING AVERAGE

```
public static Vector expMovingAverage(Vector vInitChart, int k)
{
    Vector vTemp = new Vector();
    Vector vMA = new Vector();
    double dEMAi, dOldEMAi, E;
    float fTempX, fTempY;

    int n = vInitChart.size() - 1;

    // first point of the EMA serie
    vTemp = (Vector)(vInitChart.elementAt(0));
    fTempY = (float)(Double.parseDouble( (String)(vTemp.elementAt(1)) ) );
    dOldEMAi = (double)fTempY;
    vMA.add(vTemp);

    // now the rest
    for (int i = 1; i ≤ n; i++)
    {
        vTemp = (Vector)(vInitChart.elementAt(i));
        fTempX = (float)(Double.parseDouble( (String)(vTemp.elementAt(0)) ) );
        fTempY = (float)(Double.parseDouble( (String)(vTemp.elementAt(1)) ) );

        E = (2.0f / (k + 1.0f));

        dEMAi = (1.0f - E) * dOldEMAi + E * fTempY;

        Double DEMAi = new Double(dEMAi);
        Float FTempX = new Float(fTempX);
        Vector vData = new Vector();

        vData.add(FTempX.toString());
        vData.add(DEMAi.toString());

        dOldEMAi = dEMAi;

        vMA.add(vData);
    }

    return vMA;
}
```

WEIGHTED MOVING AVERAGE

```
public static Vector weightedMovingAverage(Vector vInitChart, int k)
{
    Vector vTemp = new Vector();
    Vector vTemp2 = new Vector();
    Vector vWMA = new Vector();;
    float fTempX, fTempY;

    int n = vInitChart.size() - 1;
    Globals.taStatus.append("\n coeff : " + k);

    double dDivisor = (k * (k+1)) / 2.0f;
    double dTopDivision = 0.0f;
    double dRes = 0.0f;
    int start = 0;
    int v = k;
    int c = 0;

    for (int i = k; i ≤ n; i++)
    {
        // compute the ponderated sum
        c = 0;
        dTopDivision = 0.0f;

        for (int j = start; j ≤ start + k - 1; j++)
        {
            vTemp = (Vector)(vInitChart.elementAt(j));
            fTempX = (float)(Double.parseDouble( (String)(vTemp.elementAt(0)) ) );
            fTempY = (float)(Double.parseDouble( (String)(vTemp.elementAt(1)) ) );

            dTopDivision += (c+1) * fTempY;
            c++;
        }

        start++;
        dRes = dTopDivision / dDivisor;

        Vector vData = new Vector();

        vTemp = (Vector)(vInitChart.elementAt(v));
        fTempX = (float)(Double.parseDouble( (String)(vTemp.elementAt(0)) ) );

        Float FTempX = new Float(fTempX);
        Double DRes = new Double(dRes);

        vData.add(FTempX.toString());
        vData.add(DRes.toString());

        vWMA.add(vData);
        v++;
    }

    Globals.taStatus.append("\n size : " + vWMA.size());

    return vWMA;
}
```

Sonification.java

The `Sonification` class is obviously the most important and complex class in SoundChart. Its main method is used to create a MIDI sequence, which consists of at most 4 steps: creating the base pitch mapping, computing extreme values, adding beat drums, and finally handling stereo panning. Other methods are used to open and close a MIDI sequence, and to handle the sonification buttons that control a sequence.

```
import javax.swing.*;
import javax.swing.border.*;
import javax.swing.JFileChooser.*;
import javax.sound.midi.*;
import java.awt.event.*;
import java.awt.*;
import java.awt.Graphics.*;
import java.awt.geom.*;
import java.util.Vector;
import java.lang.Double;
import java.io.*;
import java.util.*;


class SequenceSliderListener implements ActionListener
{
    public void actionPerformed(ActionEvent e)
    {
        // get sequence position
        long lPos = Globals.midiSequencer.getMicrosecondPosition();

        // set slider position
        Globals.slSequence.setValue((int)((float)lPos / Globals.midiSequenceLength * 100.0f));

        // are we finished playing the current sequence?
        if (!Globals.midiSequencer.isRunning())
        {
            // stop sonification
            Sonification.stop();
        }
    }
}


public class Sonification
{
    // converts milliseconds to ticks
    private static long Msec2Tick(long msec)
    {
        return (long)(msec * 0.2);
    }
```

```
private static MidiEvent createMidiEvent(int channel, int command,
    int data1, int data2, long msec)
{
    MidiEvent event = null;

    try
    {
        // create message
        ShortMessage msg = new ShortMessage();
        msg.setMessage(command, channel, data1, data2);

        // create event
        event = new MidiEvent(msg, Msec2Tick(msec));
    }
    catch (InvalidMidiDataException ex)
    {
        System.out.println("\nERROR in createMidiEvent(): " + ex + "\n");
        System.exit(0);
    }

    return event;
}


private static void createMidiSequence()
{
    int iProgram = 51, iWarningProgram = 71, iDrumProgram = 47;
    int iInterval, iMinNote, iMaxNote, iNote, iTime, iLastDrum = -5000;
    Vector vChartSonify = null, extremeValues = null;
    double dMinY = Double.MAX_VALUE, dMaxY = Double.MIN_VALUE;

    Globals.taStatus.append("\nCreating MIDI sequence:\n");
    Globals.taStatus.append("\ttempo (BPM): " + Globals.midiSequencer.getTempoInBPM()
        + "\n");

    // get the program number
    iProgram = Globals.cbProgram.getSelectedIndex();
    if (iProgram == 0 || (iProgram > 8 && iProgram  9 == 0)) iProgram++;
    if (iProgram < 9) iProgram–;
    else iProgram = iProgram - (iProgram / 9) - 1;
    Globals.taStatus.append("\tprogram number: " + iProgram + "\n");

    if (Globals.chbDrumBeats.isSelected())
    {
        // get the drum program number
        iDrumProgram = Globals.cbDrumProgram.getSelectedIndex();
        if (iDrumProgram == 0 || (iDrumProgram > 8 && iDrumProgram  9 ==
            0)) iDrumProgram++;
        if (iDrumProgram < 9) iDrumProgram–;
        else iDrumProgram = iDrumProgram - (iDrumProgram / 9) - 1;
        Globals.taStatus.append("\tdrum program number: " + iDrumProgram + "\n");
    }

    if (Globals.chbExtremeValues.isSelected())
    {
        // get the warnings program number
        iWarningProgram = Globals.cbExtremeProgram.getSelectedIndex();
        if (iWarningProgram == 0 || (iWarningProgram > 8 && iWarningProgram  9 == 0))
        iWarningProgram++;
        if (iWarningProgram < 9) iWarningProgram--;
        else iWarningProgram = iWarningProgram - (iWarningProgram / 9) - 1;
        Globals.taStatus.append("\twarning program number: " + iWarningProgram + "\n");
    }

    // get chart to sonify
    String SItem = (String)(Globals.cbChart.getSelectedItem());
    if (SItem.equals("Main Chart"))
    {
        vChartSonify = Globals.vCoord;
        Globals.taStatus.append("\tselected chart: Main Chart\n");
    }
    else if (SItem.equals("Linear MA Chart"))
    {
        vChartSonify = Globals.vMACoord;
        Globals.taStatus.append("\tselected chart: Linear MA Chart\n");
    }
    else if (SItem.equals("Exponential MA Chart"))
    {
        vChartSonify = Globals.vEMACoord;
        Globals.taStatus.append("\tselected chart: Exponential MA Chart\n");
    }
    else if (SItem.equals("Weighted MA Chart"))
    {
        vChartSonify = Globals.vWMACoord;
        Globals.taStatus.append("\tselected chart: Weighted MA Chart\n");
    }
    else
    {
        System.out.println("\nERROR in createMidiSequence(): Sonification Chart not
            present\n");
        System.exit(0);
    }

    if (Globals.chbExtremeValues.isSelected())
    {
        // compute extreme values indices (from initial chart !!)
        extremeValues = ExtremeValues.getExtremeValuesIndices(Globals.vCoord);

        // recompute max and min Y
        for (int i = 0; i < Globals.vCoord.size(); i++)
        {
            // skip exteme values
            if (extremeValues.contains(new Integer(i))) continue;

            Vector vTemp = (Vector)(Globals.vCoord.elementAt(i));
            double dY = Double.parseDouble((String)(vTemp.elementAt(1)));
            if (dY < dMinY) dMinY = dY;
            if (dY > dMaxY) dMaxY = dY;
        }
    }
    else
    {
        dMinY = Globals.dMinY;
        dMaxY = Globals.dMaxY;
    }

    // get sonification settings
    iInterval = Globals.slTime.getValue();
```

```
iMinNote = Globals.slMinNote.getValue();
iMaxNote = Globals.slMaxNote.getValue();
Globals.taStatus.append("\tinterval: " + iInterval + "\n");
Globals.taStatus.append("\tminnote: " + iMinNote + "\n");
Globals.taStatus.append("\tmaxnote: " + iMaxNote + "\n");

// create MIDI sequence
try
{
    Globals.midiSequence = new Sequence(Sequence.PPQ, 100);
    Track track = Globals.midiSequence.createTrack();

    // initialize program numbers
    track.add(createMidiEvent(0, ShortMessage.PROGRAM_CHANGE, iProgram, 0, 0));
    track.add(createMidiEvent(1, ShortMessage.PROGRAM_CHANGE, iWarningProgram, 0, 0));
    track.add(createMidiEvent(2, ShortMessage.PROGRAM_CHANGE, iDrumProgram, 0, 0));

    // compute dYdelta (used to compute the slope)
    double dYdelta = (dMaxY == dMinY) ? 1 : (dMaxY - dMinY);

    // add MIDI events
    for (int i = 0; i < vChartSonify.size(); i++)
    {
        Vector vTemp = (Vector)(vChartSonify.elementAt(i));
        double dY = Double.parseDouble((String)(vTemp.elementAt(1)));

        if (Globals.rbLinearMapping.isSelected())
        {
            // LINEAR MAPPING
            iNote = (int)(((dY - dMinY) * (iMaxNote - iMinNote)) / (dMaxY - dMinY));
            iNote += iMinNote;
        }
        else
        {
            // CHROMATIC SCALE MAPPING
            double exp = (dY - dMinY) / (dMaxY - dMinY);
            iNote = (int)(iMinNote * Math.pow(iMaxNote / iMinNote, exp));
        }

        iTime = i * iInterval;

        // add warning for extreme values (on channel 1)
        boolean bAdded = false;
        if (Globals.chbExtremeValues.isSelected())
        {
            int index = i;
            if (SItem.equals("Linear MA Chart"))
            {
                int iMAOrder = (new Integer(Globals.lValOrder.getText())).intValue();
                index = i + iMAOrder;
            }
            else if (SItem.equals("Weighted MA Chart"))
            {
                int iWMACoeff = (new Integer(Globals.lValCoeff.getText())).intValue();
                index = i + iWMACoeff;
            }

            if (extremeValues.contains(new Integer(index)))
            {
                // play warning
                track.add(createMidiEvent(1, ShortMessage.NOTE_ON, 50, 127, iTime));
                track.add(createMidiEvent(1, ShortMessage.NOTE_OFF,
                    50, 127, iTime + 300));

                bAdded = true;
            }
        }

        if (!bAdded)
        {
            if (Globals.chbStereo.isSelected())
            {
                int iPan = (int)(((float)i / (float)vChartSonify.size()) * 127.0);

                // add PAN control change
                track.add(createMidiEvent(0, ShortMessage.CONTROL_CHANGE,
                    10, iPan, iTime));
                track.add(createMidiEvent(2, ShortMessage.CONTROL_CHANGE,
                    10, iPan, iTime));
            }

            // add NOTE ON event
            track.add(createMidiEvent(0, ShortMessage.NOTE_ON, iNote, 127, iTime));

            // add NOTE OFF event
            track.add(createMidiEvent(0, ShortMessage.NOTE_OFF,
                iNote, 127, iTime + iInterval));
        }
    }

    // add drum beats (on channel 2)
    if (Globals.chbDrumBeats.isSelected())
    {
        for (int i = 1; i < vChartSonify.size(); i++)
        {
            Vector vTemp = (Vector)(vChartSonify.elementAt(i));
            double dY = Double.parseDouble((String)(vTemp.elementAt(1)));
            Vector vTemp0 = (Vector)(vChartSonify.elementAt(i-1));
            double dY0 = Double.parseDouble((String)(vTemp0.elementAt(1)));

            double slope = (dY - dY0) / dYdelta;
            int drumInterval = (slope == 0.0) ? 1000 : (int)Math.abs(10.0 / slope);

            int iY0 = (i-1) * iInterval;
            int iY = i * iInterval;

            int iStart = iLastDrum + drumInterval;
            if (iStart < iY0) iStart = iY0;

            for (int j = iStart; j < iY; j+=drumInterval)
            {
                // add drum beat
                track.add(createMidiEvent(2, ShortMessage.NOTE_ON, 60, 127, j));
                track.add(createMidiEvent(2, ShortMessage.NOTE_OFF, 60, 127, j + 250));
                iLastDrum = j;
            }
        }
    }
```

```
            Globals.midiSequenceLength = Globals.midiSequence.getMicrosecondLength();
            Globals.taStatus.append("Sequence length in seconds: " +
                Globals.midiSequenceLength / (float)1e6 + "\n");
            Globals.taStatus.append("                       ticks:    " +
                Globals.midiSequence.getTickLength() + "\n\n");

            // add sequence to sequencer
            Globals.midiSequencer.setSequence(Globals.midiSequence);
        }
        catch (NumberFormatException ex)
        {
            System.out.println("\nERROR in createMidiSequence(): " + ex + "\n");
            System.exit(0);
        }
        catch (InvalidMidiDataException ex)
        {
            System.out.println("\nERROR in createMidiSequence(): " + ex + "\n");
            System.exit(0);
        }

        Globals.midiSequenceCreated = true;
    }


    public static void openMidiSequencer()
    {
        try
        {
            Globals.taStatus.append("\nOpening MIDI sequencer...");
            Globals.midiSequencer.open();
            Globals.taStatus.append(" ok\n");
        }
        catch (MidiUnavailableException ex)
        {
            System.out.println("ERROR in enableSonification(): " + ex + "\n");
            Globals.taStatus.append("\nERROR in enableSonification(): " + ex + "\n\n");
            return;
        }

        // create timer for slSequence updates
        Globals.timerSequence = new javax.swing.Timer(100, new SequenceSliderListener());
    }


    public static void closeMidiSequencer()
    {
        Globals.taStatus.append("\nClosing MIDI sequencer...");
        Globals.midiSequencer.close();
        Globals.taStatus.append(" ok\n");
    }


    |            SONIFICATION BUTTONS            |


    public static void reset()
    {
        Globals.slSequence.setEnabled(true);
        Globals.bReset.setEnabled(false);
        Globals.bRew.setEnabled(false);
        Globals.bPlay.setEnabled(true);
        Globals.bPause.setEnabled(false);
        Globals.bStop.setEnabled(false);
        Globals.bFfw.setEnabled(true);
        Globals.bEnd.setEnabled(true);

        Globals.timerSequence.stop();
        if (Globals.midiSequencer.isOpen()) Globals.midiSequencer.stop();
        Globals.slSequence.setValue(0);
    }

    public static void rew()
    {
        Globals.slSequence.setEnabled(true);

        Globals.bReset.setEnabled(true);
        Globals.bRew.setEnabled(true);
        Globals.bPlay.setEnabled(true);
        Globals.bPause.setEnabled(false);
        Globals.bStop.setEnabled(false);
        Globals.bFfw.setEnabled(true);
        Globals.bEnd.setEnabled(true);

        Globals.timerSequence.stop();
        if (Globals.midiSequencer.isOpen()) Globals.midiSequencer.stop();
        Globals.slSequence.setValue(Globals.slSequence.getValue() - 10);
    }

    public static void play()
    {
        Globals.cbProgram.setEnabled(false);
        Globals.slMinNote.setEnabled(false);
        Globals.slMaxNote.setEnabled(false);
        Globals.slTime.setEnabled(false);
        Globals.chbExtremeValues.setEnabled(false);
        Globals.chbDrumBeats.setEnabled(false);
        Globals.chbStereo.setEnabled(false);
        Globals.cbChart.setEnabled(false);
        Globals.cbExtremeProgram.setEnabled(false);
        Globals.cbDrumProgram.setEnabled(false);
        Globals.rbLinearMapping.setEnabled(false);
        Globals.rbChromaticMapping.setEnabled(false);
        Globals.slSequence.setEnabled(false);

        Globals.bReset.setEnabled(true);
        Globals.bRew.setEnabled(true);
        Globals.bPlay.setEnabled(false);
        Globals.bPause.setEnabled(true);
        Globals.bStop.setEnabled(true);
        Globals.bFfw.setEnabled(true);
        Globals.bEnd.setEnabled(true);

        if (!Globals.midiSequenceCreated) createMidiSequence();

        Globals.timerSequence.start();
        Globals.midiSequencer.start();
        // match sequence position with slSequence position
        float pos = (float)Globals.slSequence.getValue() * Globals.midiSequenceLength;
        Globals.midiSequencer.setMicrosecondPosition((long)(pos / 100.0f));

        Globals.tabbedpane.setEnabledAt(0, false);
        Globals.tabbedpane.setEnabledAt(1, false);
    }

    public static void pause()
    {
        Globals.slSequence.setEnabled(true);

        Globals.bReset.setEnabled(true);
        Globals.bRew.setEnabled(true);
        Globals.bPlay.setEnabled(true);
        Globals.bPause.setEnabled(false);
        Globals.bStop.setEnabled(true);
        Globals.bFfw.setEnabled(true);
        Globals.bEnd.setEnabled(true);

        Globals.timerSequence.stop();
        if (Globals.midiSequencer.isOpen()) Globals.midiSequencer.stop();
    }

    public static void stop()
    {
        Globals.cbProgram.setEnabled(true);
        Globals.slMinNote.setEnabled(true);
        Globals.slMaxNote.setEnabled(true);
        Globals.slTime.setEnabled(true);
        Globals.chbExtremeValues.setEnabled(true);
        Globals.chbDrumBeats.setEnabled(true);
        Globals.chbStereo.setEnabled(true);
        Globals.cbChart.setEnabled(true);
        if (Globals.chbExtremeValues.isSelected()) Globals.cbExtremeProgram.setEnabled(true);
        if (Globals.chbDrumBeats.isSelected()) Globals.cbDrumProgram.setEnabled(true);
        Globals.rbLinearMapping.setEnabled(true);
        Globals.rbChromaticMapping.setEnabled(true);
        Globals.slSequence.setEnabled(true);

        Globals.bReset.setEnabled(false);
        Globals.bRew.setEnabled(false);
        Globals.bPlay.setEnabled(true);
        Globals.bPause.setEnabled(false);
        Globals.bStop.setEnabled(false);
        Globals.bFfw.setEnabled(true);
        Globals.bEnd.setEnabled(true);

        Globals.timerSequence.stop();
        if (Globals.midiSequencer.isOpen()) Globals.midiSequencer.stop();
        Globals.slSequence.setValue(0);
        Globals.midiSequenceCreated = false;

        Globals.tabbedpane.setEnabledAt(0, true);
        Globals.tabbedpane.setEnabledAt(1, true);
    }

    public static void ffw()
    {
        Globals.slSequence.setEnabled(true);

        Globals.bReset.setEnabled(true);
        Globals.bRew.setEnabled(true);
        Globals.bPlay.setEnabled(true);
        Globals.bPause.setEnabled(false);
        Globals.bStop.setEnabled(false);
        Globals.bFfw.setEnabled(true);
        Globals.bEnd.setEnabled(true);

        Globals.timerSequence.stop();
        if (Globals.midiSequencer.isOpen()) Globals.midiSequencer.stop();
        Globals.slSequence.setValue(Globals.slSequence.getValue() + 10);
    }

    public static void end()
    {
        Globals.slSequence.setEnabled(true);

        Globals.bReset.setEnabled(true);
        Globals.bRew.setEnabled(true);
        Globals.bPlay.setEnabled(false);
        Globals.bPause.setEnabled(false);
        Globals.bStop.setEnabled(false);
        Globals.bFfw.setEnabled(false);
        Globals.bEnd.setEnabled(false);

        Globals.timerSequence.stop();
        if (Globals.midiSequencer.isOpen()) Globals.midiSequencer.stop();
        Globals.slSequence.setValue(100);
    }
}
```

ExtremeValues.java

The `ExtremeValues` class can only be accessed by calling the public method `getExtremeValuesIndices`, which creates a `Vector` object with the indices of the detected outliers. The actual z-score test is done by the private method `getZScoreOutliers`, which uses the mean and standard deviation of the initial time series.

```java
import javax.swing.*;
import javax.sound.midi.*;
import java.awt.*;
import java.awt.geom.*;
import java.util.Vector;
import java.lang.Double;
import java.io.*;
import java.util.*;


public class ExtremeValues
{
    private static Vector getZScoreOutliers(Vector vChart, int n, double mean, double s)
    {
        Vector evi = new Vector();

        for (int i = 0; i < n; i++)
        {
            Vector vTemp = (Vector)(vChart.elementAt(i));
            double dY = Double.parseDouble((String)(vTemp.elementAt(1)));

            double zscore = (dY - mean) / s;

            if (Math.abs(zscore) > 3.0)
            {
                evi.add(new Integer(i));

                Globals.taStatus.append("Extreme value detected at Y = " + dY);
                Globals.taStatus.append(" (z-score = " + zscore + ")\n");
            }
        }

        return evi;
    }


    public static Vector getExtremeValuesIndices(Vector vChart)
    {
        double s, mean;
        int i, n;

        // get number of elements
        n = vChart.size();
    // compute the mean
    double sumY = 0.0;
    for (i = 0; i < n; i++)
    {
        Vector vTemp = (Vector)(vChart.elementAt(i));
        double dY = Double.parseDouble((String)(vTemp.elementAt(1)));

        sumY += dY;
    }
    mean = sumY / n;

    // compute s
    double sumS = 0.0;
    for (i = 0; i < n; i++)
    {
        Vector vTemp = (Vector)(vChart.elementAt(i));
        double dY = Double.parseDouble((String)(vTemp.elementAt(1)));

        sumS += Math.abs(mean - dY);
    }
    s = sumS / n;

    // compute extreme values
    return getZScoreOutliers(vChart, n, mean, s);
}
```

Midi.java

The `Midi` class simply contains a method for initializing the `Globals.vProgramCB` vector with the 128 standard MIDI instruments. The created vector contains 144 elements, because 16 group names are added to classify the instruments.

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.border.*;
import java.util.Vector;


public class Midi
{
    public static void fillProgramComboBox()
    {
        Globals.vProgramCB = new Vector(144);

        Globals.vProgramCB.add("--- PIANO ---");
        Globals.vProgramCB.add("1     Acoustic Grand");
        Globals.vProgramCB.add("2     Bright Acoustic");
        Globals.vProgramCB.add("3     Electric Grand");
        Globals.vProgramCB.add("4     Honky-Tonk");
        Globals.vProgramCB.add("5     Electric Piano 1");
        Globals.vProgramCB.add("6     Electric Piano 2");
        Globals.vProgramCB.add("7     Harpsichord");
        Globals.vProgramCB.add("8     Clavinet");
        Globals.vProgramCB.add("--- CHROMATIC PERCUSSION ---");
        Globals.vProgramCB.add("9     Celesta");
        Globals.vProgramCB.add("10    Glockenspiel");
        Globals.vProgramCB.add("11    Music Box");
        Globals.vProgramCB.add("12    Vibraphone");
        Globals.vProgramCB.add("13    Marimba");
        Globals.vProgramCB.add("14    Xylophone");
        Globals.vProgramCB.add("15    Tubular Bells");
        Globals.vProgramCB.add("16    Dulcimer");
        Globals.vProgramCB.add("--- ORGAN ---");
        Globals.vProgramCB.add("17    Drawbar Organ");
        Globals.vProgramCB.add("18    Percussive Organ");
        Globals.vProgramCB.add("19    Rock Organ");
        Globals.vProgramCB.add("20    Church Organ");
        Globals.vProgramCB.add("21    Reed Organ");
        Globals.vProgramCB.add("22    Accoridan");
        Globals.vProgramCB.add("23    Harmonica");
        Globals.vProgramCB.add("24    Tango Accordian");
        Globals.vProgramCB.add("--- GUITAR ---");
        Globals.vProgramCB.add("25    Nylon String Guitar");
        Globals.vProgramCB.add("26    Steel String Guitar");
        Globals.vProgramCB.add("27    Electric Jazz Guitar");
        Globals.vProgramCB.add("28    Electric Clean Guitar");
        Globals.vProgramCB.add("29    Electric Muted Guitar");
Globals.vProgramCB.add("30   Overdriven Guitar");
Globals.vProgramCB.add("31   Distortion Guitar");
Globals.vProgramCB.add("32   Guitar Harmonics");
Globals.vProgramCB.add("--- BASS ---");
Globals.vProgramCB.add("33   Acoustic Bass");
Globals.vProgramCB.add("34   Electric Bass(finger)");
Globals.vProgramCB.add("35   Electric Bass(pick)");
Globals.vProgramCB.add("36   Fretless Bass");
Globals.vProgramCB.add("37   Slap Bass 1");
Globals.vProgramCB.add("38   Slap Bass 2");
Globals.vProgramCB.add("39   Synth Bass 1");
Globals.vProgramCB.add("40   Synth Bass 2");
Globals.vProgramCB.add("--- SOLO STRINGS ---");
Globals.vProgramCB.add("41   Violin");
Globals.vProgramCB.add("42   Viola");
Globals.vProgramCB.add("43   Cello");
Globals.vProgramCB.add("44   Contrabass");
Globals.vProgramCB.add("45   Tremolo Strings");
Globals.vProgramCB.add("46   Pizzicato Strings");
Globals.vProgramCB.add("47   Orchestral Strings");
Globals.vProgramCB.add("48   Timpani");
Globals.vProgramCB.add("--- ENSEMBLE ---");
Globals.vProgramCB.add("49   String Ensemble 1");
Globals.vProgramCB.add("50   String Ensemble 2");
Globals.vProgramCB.add("51   SynthStrings 1");
Globals.vProgramCB.add("52   SynthStrings 2");
Globals.vProgramCB.add("53   Choir Aahs");
Globals.vProgramCB.add("54   Voice Oohs");
Globals.vProgramCB.add("55   Synth Voice");
Globals.vProgramCB.add("56   Orchestra Hit");
Globals.vProgramCB.add("--- BRASS ---");
Globals.vProgramCB.add("57   Trumpet");
Globals.vProgramCB.add("58   Trombone");
Globals.vProgramCB.add("59   Tuba");
Globals.vProgramCB.add("60   Muted Trumpet");
Globals.vProgramCB.add("61   French Horn");
Globals.vProgramCB.add("62   Brass Section");
Globals.vProgramCB.add("63   SynthBrass 1");
Globals.vProgramCB.add("64   SynthBrass 2");
Globals.vProgramCB.add("--- REED ---");
Globals.vProgramCB.add("65   Soprano Sax");
Globals.vProgramCB.add("66   Alto Sax");
Globals.vProgramCB.add("67   Tenor Sax");
Globals.vProgramCB.add("68   Baritone Sax");
Globals.vProgramCB.add("69   Oboe");
Globals.vProgramCB.add("70   English Horn");
Globals.vProgramCB.add("71   Bassoon");
Globals.vProgramCB.add("72   Clarinet");
Globals.vProgramCB.add("--- PIPE ---");
Globals.vProgramCB.add("73   Piccolo");
Globals.vProgramCB.add("74   Flute");
Globals.vProgramCB.add("75   Recorder");
Globals.vProgramCB.add("76   Pan Flute");
Globals.vProgramCB.add("77   Blown Bottle");
Globals.vProgramCB.add("78   Skakuhachi");
Globals.vProgramCB.add("79   Whistle");
Globals.vProgramCB.add("80   Ocarina");
Globals.vProgramCB.add("--- SYNTH LEAD ---");
Globals.vProgramCB.add("81   Lead 1 (square)");
```

```
        Globals.vProgramCB.add("82  Lead 2 (sawtooth)");
        Globals.vProgramCB.add("83  Lead 3 (calliope)");
        Globals.vProgramCB.add("84  Lead 4 (chiff)");
        Globals.vProgramCB.add("85  Lead 5 (charang)");
        Globals.vProgramCB.add("86  Lead 6 (voice)");
        Globals.vProgramCB.add("87  Lead 7 (fifths)");
        Globals.vProgramCB.add("88  Lead 8 (bass+lead)");
        Globals.vProgramCB.add("--- SYNTH PAD ---");
        Globals.vProgramCB.add("89  Pad 1 (new age)");
        Globals.vProgramCB.add("90  Pad 2 (warm)");
        Globals.vProgramCB.add("91  Pad 3 (polysynth)");
        Globals.vProgramCB.add("92  Pad 4 (choir)");
        Globals.vProgramCB.add("93  Pad 5 (bowed)");
        Globals.vProgramCB.add("94  Pad 6 (metallic)");
        Globals.vProgramCB.add("95  Pad 7 (halo)");
        Globals.vProgramCB.add("96  Pad 8 (sweep)");
        Globals.vProgramCB.add("--- SYNTH EFFECTS ---");
        Globals.vProgramCB.add("97  FX 1 (rain)");
        Globals.vProgramCB.add("98  FX 2 (soundtrack)");
        Globals.vProgramCB.add("99  FX 3 (crystal)");
        Globals.vProgramCB.add("100 FX 4 (atmosphere)");
        Globals.vProgramCB.add("101 FX 5 (brightness)");
        Globals.vProgramCB.add("102 FX 6 (goblins)");
        Globals.vProgramCB.add("103 FX 7 (echoes)");
        Globals.vProgramCB.add("104 FX 8 (sci-fi)");
        Globals.vProgramCB.add("--- ETHNIC ---");
        Globals.vProgramCB.add("105 Sitar");
        Globals.vProgramCB.add("106 Banjo");
        Globals.vProgramCB.add("107 Shamisen");
        Globals.vProgramCB.add("108 Koto");
        Globals.vProgramCB.add("109 Kalimba");
        Globals.vProgramCB.add("110 Bagpipe");
        Globals.vProgramCB.add("111 Fiddle");
        Globals.vProgramCB.add("112 Shanai");
        Globals.vProgramCB.add("--- PERCUSSIVE ---");
        Globals.vProgramCB.add("113 Tinkle Bell");
        Globals.vProgramCB.add("114 Agogo");
        Globals.vProgramCB.add("115 Steel Drums");
        Globals.vProgramCB.add("116 Woodblock");
        Globals.vProgramCB.add("117 Taiko Drum");
        Globals.vProgramCB.add("118 Melodic Tom");
        Globals.vProgramCB.add("119 Synth Drum");
        Globals.vProgramCB.add("120 Reverse Cymbal");
        Globals.vProgramCB.add("--- SOUND EFFECTS ---");
        Globals.vProgramCB.add("121 Guitar Fret Noise");
        Globals.vProgramCB.add("122 Breath Noise");
        Globals.vProgramCB.add("123 Seashore");
        Globals.vProgramCB.add("124 Bird Tweet");
        Globals.vProgramCB.add("125 Telephone Ring");
        Globals.vProgramCB.add("126 Helicopter");
        Globals.vProgramCB.add("127 Applause");
        Globals.vProgramCB.add("128 Gunshot");
}
```

SoundChart3D.java

The SoundChart3D class is the main class of the application. The first action is to initialize the two windows used in the application. SC3DFrame represents the main window, where the user can manipulate 3D charts and alter the sonification options. SOFrame represents the window containing additional sonification options. The other main operation of the SoundChart3D class is to initialize the MIDI objects used throughout the application. The same objects are closed when the program terminates (i.e. when the main window is closed).

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.sound.midi.*;
import java.io.*;
import java.io.IOException;
import java.text.DecimalFormat;
import java.text.ParseException;
import java.applet.*;


class MainSplitPane
{
    // the main window is split in two parts
    private JSplitPane splitPane;
    private JPanel pSouth;
    private JPanel pScene;
    private JPanel pEditScene;
    private JTabbedPane tpNorth;

    public MainSplitPane()
    {
        tpNorth = new JTabbedPane();
        pSouth = new JPanel();
        pScene = GuiSouth.makeScenePanel();
        pEditScene = GuiSouth.makeEditScenePanel();

        pSouth.setLayout(null);
        pEditScene.setBounds(10,10, 450, 310);
        pScene.setBounds(480,10, 460, 310);

        pSouth.add(pEditScene);
        pSouth.add(pScene);

        Component panel2 = GuiNorth.makePanelSonification();
        tpNorth.addTab("    Sonification    ", null, panel2, "Sonification view");

        Component panel1 = GuiNorth.makePanelStatus();
        tpNorth.addTab("    Status    ", null, panel1, "Status view");

        splitPane = new JSplitPane(JSplitPane.VERTICAL_SPLIT, tpNorth, pSouth);
        splitPane.setOneTouchExpandable(false);
        splitPane.setDividerLocation(Globals.screenHeight / 2 - 30);
        splitPane.setContinuousLayout(false);

        // provide minimum sizes for the two components in the split pane
        Dimension minimumSize = new Dimension(100, 50);
        tpNorth.setMinimumSize(minimumSize);
        pSouth.setMinimumSize(minimumSize);
        pScene.setMinimumSize(minimumSize);
        pEditScene.setMinimumSize(minimumSize);

        // provide a preferred size for the split pane
        splitPane.setPreferredSize(new Dimension(400, 200));
    }

    public JSplitPane getSplitPane()
    {
        return splitPane;
    }
}


class SC3DFrame extends JFrame
{
    public SC3DFrame()
    {
        setTitle("SoundChart 3D");

        addWindowListener(new WindowAdapter()
        {
            public void windowClosing(WindowEvent e)
            {
                SoundChart3D.closeMidiObjects();
                System.exit(0);
            }
        } );

        setSize(Globals.screenWidth, Globals.screenHeight);

        Container contentPane = getContentPane();

        MainSplitPane sp = new MainSplitPane();

        contentPane.add(sp.getSplitPane());
    }
}


class SOFrame extends JFrame
{
    public SOFrame()
    {
        setTitle("Sonification options");
        setBounds(300,220,430,290);

        addWindowListener(new WindowAdapter()
        {
            public void windowClosing(WindowEvent e)
            {
                Globals.bOptions.setEnabled(true);
            }
        } );

        getContentPane().add(GuiNorth.makePanelSoundOptions());
    }
}


public class SoundChart3D
{
    // retrieves the number pressed by the user
    private static int getKbdInt()
    {
        int val = 0;

        try
        {
            InputStreamReader isr = new InputStreamReader(System.in);
            BufferedReader br = new BufferedReader(isr);

            String s = br.readLine();
            DecimalFormat df = new DecimalFormat();

            val = (df.parse(s)).intValue();
        }
        catch (IOException ex)
        {
            System.out.println("\nERROR in getKbdInt(): " + ex + "\n");
            System.exit(1);
        }
        catch (ParseException ex)
        {
            System.out.println("\nERROR in getKbdInt(): " + ex + "\n");
            System.exit(1);
        }

        return val;
    }


    static void initMidiObjects()
    {
        MidiDevice.Info[] midiDevices = MidiSystem.getMidiDeviceInfo();

        if (midiDevices.length == 0)
        {
            System.out.println("No MIDI devices found on this system!\n");
            System.exit(0);
        }

        // print available MIDI devices
        System.out.println("Available MIDI devices on this system:\n");
        for (int i = 0; i < midiDevices.length; i++)
            System.out.println(i + ": " + midiDevices[i]);

        // the user must choose a MIDI device
        System.out.print("\nUse device number: ");
        int iDevice = getKbdInt();
        try
        {
            System.out.print("\nGetting MIDI device...");
            Globals.midiDevice = MidiSystem.getMidiDevice(midiDevices[iDevice]);
            System.out.println(" ok");

            System.out.print("Getting synthesizer...");
            Globals.midiSynthesizer = MidiSystem.getSynthesizer();
            System.out.println(" ok");

            System.out.print("Getting sequencer...");
            Globals.midiSequencer = MidiSystem.getSequencer();
            System.out.println(" ok\n");
        }
        catch (MidiUnavailableException ex)
        {
            System.out.println("\nERROR in initMidiDevice(): " + ex + "\n");
            System.exit(1);
        }
        catch (SecurityException ex)
        {
            System.out.println("\nERROR in initMidiDevice(): " + ex + "\n");
            System.exit(1);
        }
    }


    static void closeMidiObjects()
    {
        System.out.print("Closing MIDI objects...");
        Globals.midiSynthesizer.close();
        Globals.midiSequencer.close();
        Globals.midiDevice.close();
        System.out.println(" ok\n");
    }


    public static void main(String[] args)
    {
        // create windows
        JFrame frame = new SC3DFrame();
        Globals.optionsFrame = new SOFrame();

        System.out.println(" _________________________");
        System.out.println("|                         |");
        System.out.println("|     SOUNDCHART 3D       |");
        System.out.println("|_________________________|\n\n");

        // initialize MIDI objects
        initMidiObjects();

        // show main window
        frame.show();
    }
}
```

GuiNorth.java

The `GuiNorth` class creates the northern part of SoundChart3D's main window, as well as the Sonification Options panel. Unlike SoundChart's window, it contains only 2 panels, since the 3D scene is created and manipulated by the southern part of the window. The two panels are the Sonification panel and the Status panel.

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.border.*;
import java.util.Vector;
import javax.swing.JSlider;
import javax.swing.table.*;
import javax.swing.event.ListSelectionEvent;


public class GuiNorth
{
    SONIFICATION PANEL

    protected static Component makePanelSonification()
    {
        JPanel panel = new JPanel(false);
        panel.setLayout(null);

        /** MIDI settings panel **/

        JPanel pSonification = new JPanel();
        pSonification.setLayout(null);
        pSonification.setBounds(25,10,425,260);
        Border bEtched = BorderFactory.createEtchedBorder();
        Border bTitle = BorderFactory.createTitledBorder(bEtched, "MIDI settings");
        pSonification.setBorder(bTitle);

        // program combobox
        JLabel lProgram = new JLabel("Select program :");
        lProgram.setBounds(25,40,180,20);
        // fill vProgramCB vector with MIDI programs
        Midi.fillProgramComboBox();
        Globals.cbProgram = new JComboBox(Globals.vProgramCB);
        Globals.cbProgram.setSelectedItem("52    SynthStrings 2");
        Globals.cbProgram.setEditable(false);
        Globals.cbProgram.setBounds(180,35,225,30);
        Globals.cbProgram.addActionListener(new AListener());
        Globals.cbProgram.setEnabled(false);

        // interval slider
        JLabel lInterval = new JLabel("Interval :");
        lInterval.setBounds(25,90,180,20);
        Globals.slTime = new JSlider(JSlider.HORIZONTAL, 0, 1000, 200);
        Globals.slTime.setBounds(180,90,155,25);
Globals.slTime.addChangeListener(new SliderListener());
Globals.slTime.setMajorTickSpacing(5);
Globals.slTime.setMinorTickSpacing(1);
Globals.slTime.setEnabled(false);
Globals.lTime = new JLabel("200 ms" );
Globals.lTime.setBounds(355,86,50,20);

// minNote slider
JLabel lMinNote = new JLabel("Min Pitch :");
lMinNote.setBounds(25,130,180,20);
Globals.slMinNote = new JSlider(JSlider.HORIZONTAL, 0, 127, 25);
Globals.slMinNote.setBounds(180,130,155,25);
Globals.slMinNote.addChangeListener(new SliderListener());
Globals.slMinNote.setMajorTickSpacing(2);
Globals.slMinNote.setMinorTickSpacing(1);
Globals.slMinNote.setEnabled(false);
Globals.lValMinNote = new JLabel("25");
Globals.lValMinNote.setBounds(355,126,50,20);

// maxNote slider
JLabel lMaxNote = new JLabel("Max Pitch :");
lMaxNote.setBounds(25,170,180,20);
Globals.slMaxNote = new JSlider(JSlider.HORIZONTAL, 0, 127, 100);
Globals.slMaxNote.setBounds(180,170,155,25);
Globals.slMaxNote.addChangeListener(new SliderListener());
Globals.slMaxNote.setMajorTickSpacing(2);
Globals.slMaxNote.setMinorTickSpacing(1);
Globals.slMaxNote.setEnabled(false);
Globals.lValMaxNote = new JLabel("100");
Globals.lValMaxNote.setBounds(355,166,50,20);

// sonification options button
Globals.bOptions = new JButton("More sonification options...");
Globals.bOptions.setBounds(100,210,230,30);
Globals.bOptions.addActionListener(new AListener());
Globals.bOptions.setEnabled(false);

// add all these elements to MIDI settings panel
pSonification.add(lProgram);
pSonification.add(Globals.cbProgram);
pSonification.add(lInterval);
pSonification.add(Globals.slTime);
pSonification.add(Globals.lTime);
pSonification.add(lMinNote);
pSonification.add(Globals.slMinNote);
pSonification.add(Globals.lValMinNote);
pSonification.add(lMaxNote);
pSonification.add(Globals.slMaxNote);
pSonification.add(Globals.lValMaxNote);
pSonification.add(Globals.bOptions);

/** Sonification type panel **/

JPanel pSonificationType = new JPanel();
pSonificationType.setLayout(null);
pSonificationType.setBounds(475,10,425,260);
Border bEtched2 = BorderFactory.createEtchedBorder();
Border bTitle2 = BorderFactory.createTitledBorder(bEtched2, "Sonification type");
pSonificationType.setBorder(bTitle2);
```

```
// sonification type combobox
JLabel lSonificationType = new JLabel("Sonification type :");
lSonificationType.setBounds(25,40,180,20);
String[] sonificationTypes = { "Horizontal Travelling (along X-axis)",
    "Vertical Travelling (along Z-axis)", "Diagonal Travelling"};
Globals.cbSonificationType = new JComboBox(sonificationTypes);
Globals.cbSonificationType.setEditable(false);
Globals.cbSonificationType.setBounds(180,35,225,30);
Globals.cbSonificationType.setEnabled(false);

// mapping type radio buttons
JLabel lMapping = new JLabel("Mapping type :");
lMapping.setBounds(25,80,140,20);
Globals.rbLinearMapping = new JRadioButton("Linear mapping", true);
Globals.rbLinearMapping.setBounds(180,80,180,30);
Globals.rbLinearMapping.setEnabled(false);
Globals.rbChromaticMapping = new JRadioButton("Chromatic scale mapping", false);
Globals.rbChromaticMapping.setBounds(180,105,180,30);
Globals.rbChromaticMapping.setEnabled(false);
ButtonGroup bgMappingType = new ButtonGroup();
bgMappingType.add(Globals.rbLinearMapping);
bgMappingType.add(Globals.rbChromaticMapping);

// sequence buttons
ImageIcon iPlay = new ImageIcon("data/play1.gif");
ImageIcon iPause = new ImageIcon("data/pause1.gif");
ImageIcon iStop = new ImageIcon("data/stop1.gif");

Globals.bPlay = new JButton(iPlay);
Globals.bPlay.setBounds(200,200,35,35);
Globals.bPlay.addActionListener(new AListener());
Globals.bPlay.setEnabled(false);

Globals.bPause = new JButton(iPause);
Globals.bPause.setBounds(240,200,35,35);
Globals.bPause.addActionListener(new AListener());
Globals.bPause.setEnabled(false);

Globals.bStop = new JButton(iStop);
Globals.bStop.setBounds(280,200,35,35);
Globals.bStop.addActionListener(new AListener());
Globals.bStop.setEnabled(false);

// add all these elements to sonification type panel
pSonificationType.add(lSonificationType);
pSonificationType.add(Globals.cbSonificationType);
pSonificationType.add(lMapping);
pSonificationType.add(Globals.rbLinearMapping);
pSonificationType.add(Globals.rbChromaticMapping);
pSonificationType.add(Globals.bPlay);
pSonificationType.add(Globals.bPause);
pSonificationType.add(Globals.bStop);

// add 2 panels to main sonification panel
panel.add(pSonification);
panel.add(pSonificationType);

return panel;
```

```
}
```

SOUND OPTIONS PANEL

```
protected static Component makePanelSoundOptions()
{
    JPanel panel = new JPanel();
    panel.setLayout(null);

    /** End-Of-Line elements **/

    JPanel pEol = new JPanel();
    pEol.setLayout(null);
    pEol.setBounds(10,10,400,100);
    Border bEtched = BorderFactory.createEtchedBorder();
    Border bTitle = BorderFactory.createTitledBorder(bEtched, "End of line");
    pEol.setBorder(bTitle);

    Globals.chbEndLine = new JCheckBox("Notify end of line", false);
    Globals.chbEndLine.setBounds(25,25,200,25);
    Globals.chbEndLine.setEnabled(true);
    Globals.chbEndLine.addActionListener(new AListener());
    pEol.add(Globals.chbEndLine);

    JLabel lEolProgram = new JLabel("End of line program:");
    lEolProgram.setBounds(25,60,150,20);
    Globals.cbEolProgram = new JComboBox(Globals.vProgramCB);
    Globals.cbEolProgram.setSelectedItem("113  Tinkle Bell");
    Globals.cbEolProgram.setEditable(false);
    Globals.cbEolProgram.setBounds(180,55,200,30);
    Globals.cbEolProgram.setEnabled(false);
    pEol.add(lEolProgram);
    pEol.add(Globals.cbEolProgram);

    /** Drum Beats elements **/

    JPanel pDrum = new JPanel();
    pDrum.setLayout(null);
    pDrum.setBounds(10,120,400,100);
    Border bEtched2 = BorderFactory.createEtchedBorder();
    Border bTitle2 = BorderFactory.createTitledBorder(bEtched2, "Drum beats");
    pDrum.setBorder(bTitle2);

    Globals.chbDrumBeats = new JCheckBox("Play drum beats", false);
    Globals.chbDrumBeats.setBounds(25,25,200,25);
    Globals.chbDrumBeats.setEnabled(true);
    Globals.chbDrumBeats.addActionListener(new AListener());
    pDrum.add(Globals.chbDrumBeats);

    JLabel lDrumProgram = new JLabel("Drum program:");
    lDrumProgram.setBounds(25,60,150,20);
    Globals.cbDrumProgram = new JComboBox(Globals.vProgramCB);
    Globals.cbDrumProgram.setSelectedItem("48    Timpani");
    Globals.cbDrumProgram.setEditable(false);
    Globals.cbDrumProgram.setBounds(180,55,200,30);
    Globals.cbDrumProgram.setEnabled(false);
    pDrum.add(lDrumProgram);
    pDrum.add(Globals.cbDrumProgram);
```

```
        panel.add(pEol);
        panel.add(pDrum);

        return panel;
    }
```

STATUS PANEL

```
    protected static Component makePanelStatus()
    {
        JPanel panel = new JPanel(false);
        panel.setLayout(null);

        JPanel pStatus = new JPanel();
        pStatus.setLayout(null);
        pStatus.setBounds(10, 10, 935, 260);
        Border bEtched = BorderFactory.createEtchedBorder();
        Border bTitle = BorderFactory.createTitledBorder(bEtched, "Status view");
        pStatus.setBorder(bTitle);

        // create the JTextArea
        Globals.taStatus = new JTextArea("--------------- Status -------------\n", 10, 100);
        Globals.taStatus.setEditable(false);
        Globals.taStatus.setLineWrap(true);

        JScrollPane spStatus = new JScrollPane(Globals.taStatus);
        spStatus.setBounds(20, 30, 885, 205);

        pStatus.add(spStatus);

        panel.add(pStatus);

        return panel;
    }
}
```

GuiSouth.java

The southern part of SoundChart3D's main window contains three elements, which are all created and initialized by the `GuiSouth` class. The Scene panel is used to create, load, save or delete a 3D scene. The Edition panel gives some basic information on the selected PolySound and can be used to alter the current scene. The third element is of course the 3D scene itself.

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.border.*;
import java.util.Vector;
import javax.swing.JSlider;
import javax.swing.table.*;
import javax.swing.event.ListSelectionEvent;
import java.awt.Component;
import javax.media.j3d.Canvas3D;
import java.awt.GraphicsConfiguration;
import java.awt.GraphicsEnvironment;
import java.awt.GraphicsDevice;
import com.sun.j3d.utils.universe.*;


public class GuiSouth
{
    private static Component makeCanvas3D(GraphicsConfiguration config)
    {
        GraphicsEnvironment ge = GraphicsEnvironment.getLocalGraphicsEnvironment();
        GraphicsDevice gs = ge.getDefaultScreenDevice();
        GraphicsConfiguration gc = gs.getDefaultConfiguration();

        // create canvas3D
        Canvas3D canvas3D = new Canvas3D(config);

        // add key listener
        canvas3D.addKeyListener(new KbListener());

        return canvas3D;
    }


    protected static JPanel makeScenePanel()
    {
        JPanel panel = new JPanel(true);
        panel.setLayout(new BorderLayout());

        // create Canvas3D and initialize virtual universe
        GraphicsConfiguration config = SimpleUniverse.getPreferredConfiguration();
        Globals.canvas3D = GuiSouth.makeCanvas3D(config);
        Scene3D.initScene3D();
    // add Canvas3D to center of scene panel
    panel.add("Center", Globals.canvas3D);

    return panel;
}


protected static JPanel makeEditScenePanel()
{
    JPanel panel = new JPanel(true);
    panel.setLayout(null);

    /** scene sub-panel **/

    JPanel pScene = new JPanel();
    pScene.setLayout(null);
    pScene.setBounds(10, 10, 417, 185);

    Border bEtched1 = BorderFactory.createEtchedBorder();
    Border bTitle1 = BorderFactory.createTitledBorder(bEtched1, "Scene");
    pScene.setBorder(bTitle1);

    JLabel ltfRow = new JLabel("Number of rows");
    JLabel ltfCol = new JLabel("Number of columns");
    JLabel ltfInitHeight = new JLabel("Initial height");
    ltfRow.setBounds(16, 32, 150, 25);
    ltfCol.setBounds(16, 64, 150, 25);
    ltfInitHeight.setBounds(16, 96, 150, 25);

    Globals.tfRow = new JTextField("10");
    Globals.tfCol = new JTextField("10");
    Globals.tfInitHeight = new JTextField("0.0");
    Globals.tfRow.setBounds(170, 32, 41, 20);
    Globals.tfCol.setBounds(170, 64, 41, 20);
    Globals.tfInitHeight.setBounds(170, 96, 41, 20);

    Globals.bCreate = new JButton("Create new scene");
    Globals.bCreate.setBounds(16, 140, 198, 30);
    Globals.bCreate.addActionListener(new AListener());

    Globals.bLoad = new JButton("Load scene from file...");
    Globals.bLoad.setBounds(235, 32, 165, 30);
    Globals.bLoad.addActionListener(new AListener());

    Globals.bSave = new JButton("Save scene to file...");
    Globals.bSave.setBounds(235, 72, 165, 30);
    Globals.bSave.addActionListener(new AListener());
    Globals.bSave.setEnabled(false);

    Globals.bDelete = new JButton("Delete whole scene");
    Globals.bDelete.setBounds(235, 140, 165, 30);
    Globals.bDelete.addActionListener(new AListener());
    Globals.bDelete.setEnabled(false);

    pScene.add(ltfRow);
    pScene.add(ltfCol);
    pScene.add(ltfInitHeight);

    pScene.add(Globals.tfRow);
pScene.add(Globals.tfCol);
pScene.add(Globals.tfInitHeight);

pScene.add(Globals.bCreate);
pScene.add(Globals.bLoad);
pScene.add(Globals.bSave);
pScene.add(Globals.bDelete);

panel.add(pScene);

/** edition sub-panel **/

JPanel pEdition = new JPanel();
pEdition.setLayout(null);
pEdition.setBounds(10, 210, 417, 100);

Border bEtched2 = BorderFactory.createEtchedBorder();
Border bTitle2 = BorderFactory.createTitledBorder(bEtched1, "Edition");
pEdition.setBorder(bTitle2);

Globals.ltfSelection = new JLabel("Ctrl + Left click to select");
Globals.ltfSelection.setBounds(16, 20, 150, 25);

JLabel ltfHeight = new JLabel("Coordinates :");
ltfHeight.setBounds(16, 50, 150, 25);

Globals.ltfX = new JLabel("X");
Globals.ltfX.setBounds(115, 50, 40, 25);

Globals.ltfY = new JLabel("Y");
Globals.ltfY.setBounds(185, 50, 40, 25);

Globals.ltfZ = new JLabel("Z");
Globals.ltfZ.setBounds(255, 50, 40, 25);

Globals.tfX = new JTextField();
Globals.tfX.setBounds(130, 50, 41, 20);
Globals.tfX.setEditable(false);

Globals.tfY = new JTextField();
Globals.tfY.setBounds(200, 50, 41, 20);
Globals.tfY.setEnabled(false);
Globals.tfY.addActionListener(new AListener());

Globals.tfZ = new JTextField();
Globals.tfZ.setBounds(270, 50, 41, 20);
Globals.tfZ.setEditable(false);

Globals.bSet = new JButton("set");
Globals.bSet.setBounds(340, 50, 52, 22);
Globals.bSet.setEnabled(false);
Globals.bSet.addActionListener(new AListener());

pEdition.add(ltfHeight);
pEdition.add(Globals.ltfSelection);
pEdition.add(Globals.ltfX);
pEdition.add(Globals.ltfY);
pEdition.add(Globals.ltfZ);
```

```
        pEdition.add(Globals.tfX);
        pEdition.add(Globals.tfY);
        pEdition.add(Globals.tfZ);

        pEdition.add(Globals.bSet);

        panel.add(pEdition);

        return panel;
    }
}
```

Globals.java

The SoundChart3D `Globals` class has the same utility as in the SoundChart application. It provides an easy access to the important variables used throughout the application.

```
import com.sun.j3d.utils.applet.MainFrame;
import com.sun.j3d.utils.universe.*;
import com.sun.j3d.utils.geometry.ColorCube;
import javax.media.j3d.*;
import javax.vecmath.*;
import java.awt.Component;
import java.awt.Graphics.*;
import java.awt.*;
import javax.sound.midi.*;
import javax.swing.*;
import java.util.Vector;


public class Globals
{
    // color constants
    public static final Color3f red = new Color3f(1.0f,0.0f,0.0f);
    public static final Color3f green = new Color3f(0.0f,1.0f,0.0f);
    public static final Color3f blue = new Color3f(0.0f,0.0f,1.0f);
    public static final Color3f yellow = new Color3f(1.0f,1.0f,0.0f);
    public static final Color3f cyan = new Color3f(0.0f,1.0f,1.0f);
    public static final Color3f magenta = new Color3f(1.0f,0.0f,1.0f);
    public static final Color3f white = new Color3f(1.0f,1.0f,1.0f);
    public static final Color3f black = new Color3f(0.0f,0.0f,0.0f);

    // color level constants
    public static final Color3f level0 = new Color3f(0.0f, 0.5f, 0.0f);
    public static final Color3f level1 = new Color3f(0.1f, 0.5f, 0.0f);
    public static final Color3f level2 = new Color3f(0.2f, 0.5f, 0.0f);
    public static final Color3f level3 = new Color3f(0.3f, 0.5f, 0.0f);
    public static final Color3f level4 = new Color3f(0.4f, 0.5f, 0.0f);
    public static final Color3f level5 = new Color3f(0.5f, 0.5f, 0.0f);
    public static final Color3f level6 = new Color3f(0.6f, 0.5f, 0.0f);
    public static final Color3f level7 = new Color3f(0.7f, 0.5f, 0.0f);
    public static final Color3f level8 = new Color3f(0.8f, 0.5f, 0.0f);
    public static final Color3f level9 = new Color3f(0.9f, 0.5f, 0.0f);
    public static final Color3f level10 = new Color3f(1.0f, 0.5f, 0.0f);

    // main scene
    public static BranchGroup scene;

    // grid and dimensions
    public static Point3f[] grid;
    public static int row;
    public static int column;
    public static float maxHeight;
    // selection variables
    public static int idSelectedPoly = -1;
    public static int idSelectedCorner = -1;

    // 3D appearance
    public static String rendering;
    public static String shading;

    // canvas used for drawing
    public static Component canvas3D;

    // define large bounding sphere
    public static BoundingSphere bounds = new BoundingSphere(new Point3d(0.0,0.0,0.0), 100.0);

    // main window dimensions
    static int screenWidth = 973;
    static int screenHeight = 683;

    // sonification options variables
    static JFrame optionsFrame;
    static JButton bOptions;

    // main panel objects
    public static JTextField tfRow, tfCol, tfInitHeight, tfX, tfY, tfZ;
    public static JButton bCreate, bLoad, bSave, bDelete, bSet;
    public static JLabel ltfSelection, ltfX, ltfY, ltfZ;

    // status panel object
    public static JTextArea taStatus;

    // used to read a file
    public static String sCurrentLine;

    // MIDI objects
    static MidiDevice midiDevice = null;
    static Synthesizer midiSynthesizer = null;
    static Sequencer midiSequencer = null;
    static Sequence midiSequence = null;
    static boolean midiSequenceCreated = false;
    static float midiSequenceLength;
    static Timer timerSequence;

    // sonification panel objects
    static Vector vProgramCB;
    static JComboBox cbProgram, cbDrumProgram, cbEolProgram;
    static JSlider slTime;
    static JLabel lTime;
    static JSlider slMinNote;
    static JLabel lValMinNote;
    static JSlider slMaxNote;
    static JLabel lValMaxNote;
    static JCheckBox chbEndLine, chbDrumBeats;
    static JRadioButton rbLinearMapping, rbChromaticMapping;
    static JComboBox cbSonificationType;
    static JButton bPlay, bPause, bStop;
}
```

PolySound.java

The PolySound class is a very important one. Indeed, PolySound objects form the basic elements of the 3D chart, and thus of the sonification in general. A PolySound has four points, which are defined by three coordinates (x, y and z) and other 3D characteristics.

```
import javax.media.j3d.*;
import javax.vecmath.*;
import javax.media.j3d.PolygonAttributes;
import javax.media.j3d.ColoringAttributes;
import javax.media.j3d.LineAttributes;


public class PolySound extends Shape3D
{
    // polysound attributes (3D data, color, etc.)
    public Point3f[] points;
    public int[] indices;
    public Color3f[] colors;
    public Point3f gcenter;
    public int id;


    // get coordinates of a specified point
    public float[] getPointCoord(int point)
    {
        float[] coord = new float[3];

        coord[0] = this.points[point].x;
        coord[1] = this.points[point].y;
        coord[2] = this.points[point].z;

        return coord;
    }


    // get a specified coordinate of a specified point
    public float getPointCoord(int point, char var)
    {
        float[] coord = getPointCoord(point);
        float value = 0;

        if (var == 'x' || var == 'X') value = coord[0];
        else if (var == 'y' || var == 'Y') value = coord[1];
        else if (var == 'z' || var == 'Z') value = coord[2];

        return value;
    }


    // set coordinates of a specified point
```

```
public void setPointCoord(int point, float[] coord)
{
    IndexedQuadArray geom = (IndexedQuadArray)(this.getGeometry());

    this.points[point].x = coord[0];
    this.points[point].y = coord[1];
    this.points[point].z = coord[2];

    geom.setCoordinates(point, coord);

    int index = geom.getCoordinateIndex(point);
    Globals.grid[index].x = coord[0];
    Globals.grid[index].y = coord[1];
    Globals.grid[index].z = coord[2];

    this.setGeometry(geom);
}


// set a specified coordinate of a specified point
public void setPointCoord(int point, char var, float value)
{
    float[] coord = this.getPointCoord(point);

    if (var == 'x' || var == 'X') coord[0] = value;
    else if (var == 'y' || var == 'Y') coord[1] = value;
    else if (var == 'z' || var == 'Z') coord[2] = value;

    this.setPointCoord(point, coord);
}


// set the color of a specified point
public void setColor(int point, Color3f color)
{
    IndexedQuadArray geom = (IndexedQuadArray)(this.getGeometry());

    geom.setColor(point, color);
    this.colors[point] = color;
    this.setGeometry(geom);
}


// set the gloabal color of the PolySound
public void setColor(Color3f color)
{
    IndexedQuadArray geom = (IndexedQuadArray)(this.getGeometry());

    for (int i = 0; i < 4; i++)
    {
        geom.setColor(i, color);
        this.colors[i] = color;
    }
    this.setGeometry(geom);
}


// compute the gravity center of the PolySound
public void updateGcenter()
```

```
{
    this.gcenter.x = (this.points[0].x + this.points[1].x +
        this.points[2].x + this.points[3].x) / 4.0f;
    this.gcenter.y = (this.points[0].y + this.points[1].y +
        this.points[2].y + this.points[3].y) / 4.0f;
    this.gcenter.z = (this.points[0].z + this.points[1].z +
        this.points[2].z + this.points[3].z) / 4.0f;
}


// constructor
public PolySound(int vertexCount, int indexCount, double scale, Color3f color)
{
    // initialize attributes
    points = new Point3f[4];
    indices = new int[4];
    colors = new Color3f[4];
    gcenter = new Point3f();

    /** GEOMETRY **/

    IndexedQuadArray geom = new IndexedQuadArray(vertexCount,
        IndexedQuadArray.COORDINATES |
        IndexedQuadArray.NORMALS | IndexedQuadArray.COLOR_3, indexCount);

    // set Java3D geometry capabilities
    geom.setCapability(IndexedQuadArray.ALLOW_COORDINATE_READ);
    geom.setCapability(IndexedQuadArray.ALLOW_COORDINATE_WRITE);
    geom.setCapability(IndexedQuadArray.ALLOW_COUNT_READ);
    geom.setCapability(IndexedQuadArray.ALLOW_COORDINATE_INDEX_READ);
    geom.setCapability(IndexedQuadArray.ALLOW_COORDINATE_INDEX_WRITE);
    geom.setCapability(IndexedQuadArray.ALLOW_NORMAL_READ);
    geom.setCapability(IndexedQuadArray.ALLOW_NORMAL_WRITE);
    geom.setCapability(IndexedQuadArray.ALLOW_NORMAL_INDEX_READ);
    geom.setCapability(IndexedQuadArray.ALLOW_NORMAL_INDEX_WRITE);
    geom.setCapability(IndexedQuadArray.ALLOW_COLOR_READ);
    geom.setCapability(IndexedQuadArray.ALLOW_COLOR_WRITE);
    geom.setCapability(IndexedQuadArray.ALLOW_FORMAT_READ);
    this.setCapability(ALLOW_GEOMETRY_READ);
    this.setCapability(ALLOW_GEOMETRY_WRITE);

    // initialize colors
    for (int i = 0; i < 4; i++)
    {
        geom.setColor(i, color);
        colors[i] = color;
    }

    // set geometry
    this.setGeometry(geom);

    /** APPEARANCE **/

    PolygonAttributes polyAttrib = new PolygonAttributes();

    // set Java3D appearance capabilities
    this.setCapability(ALLOW_APPEARANCE_READ);
    this.setCapability(ALLOW_APPEARANCE_WRITE);
    polyAttrib.setCapability(PolygonAttributes.ALLOW_MODE_READ);
```

```
        polyAttrib.setCapability(PolygonAttributes.ALLOW_MODE_WRITE);

        polyAttrib.setCullFace(PolygonAttributes.CULL_NONE);
        polyAttrib.setPolygonMode(PolygonAttributes.POLYGON_LINE);

        ColoringAttributes colorAttrib = new ColoringAttributes();
        colorAttrib.setShadeModel(ColoringAttributes.NICEST);

        LineAttributes lineAttrib = new LineAttributes(2.0f,
            LineAttributes.PATTERN_SOLID, true);

        Material material = new Material();
        material.setDiffuseColor(Globals.white);
        material.setSpecularColor(Globals.white);
        material.setShininess(0.0f);

        // set Java3D appearance capabilities
        Appearance Appear = new Appearance();
        Appear.setCapability(Appearance.ALLOW_POLYGON_ATTRIBUTES_READ);
        Appear.setCapability(Appearance.ALLOW_POLYGON_ATTRIBUTES_WRITE);
        Appear.setCapability(Appearance.ALLOW_COLORING_ATTRIBUTES_READ);
        Appear.setCapability(Appearance.ALLOW_COLORING_ATTRIBUTES_WRITE);
        Appear.setCapability(Appearance.ALLOW_LINE_ATTRIBUTES_READ);
        Appear.setCapability(Appearance.ALLOW_LINE_ATTRIBUTES_WRITE);
        Appear.setCapability(Appearance.ALLOW_MATERIAL_READ);
        Appear.setCapability(Appearance.ALLOW_MATERIAL_WRITE);
        Appear.setPolygonAttributes(polyAttrib);
        Appear.setColoringAttributes(colorAttrib);
        Appear.setLineAttributes(lineAttrib);
        Appear.setMaterial(material);

        this.setAppearance(Appear);
    }
}
```

AListener.java

The `AListener` class is automatically called by Java when certain elements of the interface are activated by the user (e.g. pressing a button). Java calls the `actionPerformed` method, which identifies the source of the action and performs the requested operations.

```
import javax.swing.*;
import java.awt.event.*;
import java.awt.*;
import javax.swing.border.*;
import java.io.DataInputStream;
import java.io.IOException;
import java.util.Random;
import java.io.File;
import java.io.FileInputStream;
import java.util.Vector;
import javax.vecmath.*;
import javax.media.j3d.*;


class AListener implements ActionListener
{
    public void actionPerformed(ActionEvent evt)
    {
        Object source = evt.getSource();

        CREATE BUTTON

        if (source == Globals.bCreate)
        {
            Scene3D.createScene3D();
        }

        DELETE BUTTON

        else if (source == Globals.bDelete)
        {
            Scene3D.deleteScene3D();
        }

        SET BUTTON

        else if (source == Globals.bSet)
        {
            String sY = Globals.tfY.getText().trim();
            float fY;

            // check data length
            if (sY.length() == 0) return;

            // check data format
```

```
try
{
    fY = Float.parseFloat(sY);
}
catch (NumberFormatException ex)
{
    Globals.tfY.setText("");
    Globals.taStatus.append("\nTRYING TO SET AN INVALID Y VALUE -> IGNORED\n");

    return;
}

// get selected PolySound
PolySound poly = Method3D.getPolySound(Globals.idSelectedPoly);

// get new Y value
float valY = (new Float(Globals.tfY.getText())).floatValue();

// update height
if (Globals.idSelectedCorner == -1)
{
    // a PolySound is selected
    float oldvalY = poly.gcenter.y;
    float modifY = valY - oldvalY;

    // set new height of selected PolySound
    for (int i = 0; i < 4; i++)
    {
        float coordY = poly.getPointCoord(i, 'y');
        coordY += modifY;

        poly.setPointCoord(i, 'y', coordY);
    }
}
else
{
    // a corner is selected
    poly.setPointCoord(Globals.idSelectedCorner, 'y', valY);
}

// refresh the scene
Method3D.refresh();

// update gravity centers
Method3D.updateAllGCenters();

// find the height of the highest point after modification
float maxY = 0;
float currY;

for (int i = 0; i < Globals.grid.length; i++)
{
    currY = Globals.grid[i].y;
    if (currY > maxY) maxY = currY;
}
Globals.maxHeight = maxY;

// update axis, texts and lights
Method3D.updatePicking(Globals.maxHeight);
```

```
        // update PolySound colors
        Method3D.updateColors();

        // update corners
        Method3D.delete("CORNERS");
        Method3D.addCorners(poly, Globals.magenta);

        if (Globals.idSelectedCorner ≠ -1)
        {
            TransformGroup TgChild = (TransformGroup)(Globals.scene.getChild(0));
            BranchGroup BgChild = (BranchGroup)(TgChild.getChild(9));

            // update selected corners
            for (int i = 0; i < BgChild.numChildren(); i++)
            {
                Shape3D shape = (Shape3D)BgChild.getChild(i);

                Color3f color = new Color3f();

                if (i ≠ Globals.idSelectedCorner)
                    color = new Color3f(0.46f, 0.0f, 0.34f);
                else
                    color = Globals.magenta;

                QuadArray qa = (QuadArray)shape.getGeometry();
                qa.setColor(0, color);
                qa.setColor(1, color);
                qa.setColor(2, color);
                qa.setColor(3, color);

                shape.setGeometry(qa);
            }
        }
    }
```

SAVE BUTTON

```
    else if (source == Globals.bSave)
    {
        Scene3D.saveScene3D();
    }
```

LOAD BUTTON

```
    else if (source == Globals.bLoad)
    {
        Scene3D.loadScene3D();
    }
```

PLAY BUTTON

```
    else if (source == Globals.bPlay)
    {
        // start sonification
        Sonification.play();

        // update GUI elements
        Globals.bPlay.setEnabled(false);
        Globals.bPause.setEnabled(true);
        Globals.bStop.setEnabled(true);
        Globals.tfY.setEnabled(false);
        Globals.bSet.setEnabled(false);
        Globals.bSave.setEnabled(false);
        Globals.bDelete.setEnabled(false);
        Sonification.enableSonificationComponents(false);
}
```

PAUSE BUTTON

```
else if (source == Globals.bPause)
{
        // pause sonification
        Sonification.pause();

        // update GUI elements
        Globals.bPause.setEnabled(false);
        Globals.bPlay.setEnabled(true);
        Globals.bStop.setEnabled(true);
        Sonification.enableSonificationComponents(false);
}
```

STOP BUTTON

```
else if (source == Globals.bStop)
{
        // stop sonification
        Sonification.stop();

        // update GUI elements
        Globals.bStop.setEnabled(false);
        Globals.bPause.setEnabled(false);
        Globals.bPlay.setEnabled(true);
        if (Globals.idSelectedPoly ≠ -1)
        {
                Globals.tfY.setEnabled(true);
                Globals.bSet.setEnabled(true);
        }
        Globals.bSave.setEnabled(true);
        Globals.bDelete.setEnabled(true);
        Sonification.enableSonificationComponents(true);
}
```

SONIFICATION OPTIONS ELEMENTS

```
else if (source == Globals.bOptions)
{
        Globals.bOptions.setEnabled(false);
        // open sonification options window
        Globals.optionsFrame.show();
}

else if (source == Globals.chbEndLine)
{
        Globals.cbEolProgram.setEnabled(Globals.chbEndLine.isSelected());
}

else if (source == Globals.chbDrumBeats)
```

```
        {
            Globals.cbDrumProgram.setEnabled(Globals.chbDrumBeats.isSelected());
        }
    }
}
```

Scene3D.java

The `Scene3D` class contains five methods. The `initScene3D` method is only called once, since it is used to initialize the scene (from a Java3D point of view). The other methods are used to create a new scene, delete the current scene, save the current scene to a file, and load a scene from a file. They have a corresponding button in the graphical interface.

```
import java.applet.Applet;
import java.awt.BorderLayout;
import java.awt.Frame;
import java.awt.event.*;
import java.awt.Component;
import java.awt.GraphicsConfiguration;
import com.sun.j3d.utils.applet.MainFrame;
import com.sun.j3d.utils.universe.*;
import com.sun.j3d.utils.geometry.ColorCube;
import com.sun.j3d.utils.behaviors.mouse.*;
import javax.media.j3d.PickPoint;
import javax.media.j3d.*;
import javax.vecmath.*;
import javax.media.j3d.GraphicsContext3D;
import java.io.File;
import java.io.FileInputStream;
import javax.swing.*;
import java.io.IOException;


public class Scene3D
{
    INITIALIZE THE 3D SCENE

    public static void initScene3D()
    {
        // create an empty root Branch Group
        Globals.scene = new BranchGroup();
        Globals.scene.setCapability(BranchGroup.ALLOW_CHILDREN_EXTEND);
        Globals.scene.setCapability(BranchGroup.ALLOW_CHILDREN_READ);
        Globals.scene.setCapability(BranchGroup.ALLOW_CHILDREN_WRITE);
        Globals.scene.setCapability(BranchGroup.ENABLE_PICK_REPORTING);

        // create the Main Transform Group
        TransformGroup sceneTG = new TransformGroup();
        sceneTG.setCapability(TransformGroup.ALLOW_CHILDREN_EXTEND);
        sceneTG.setCapability(TransformGroup.ALLOW_CHILDREN_READ);
        sceneTG.setCapability(TransformGroup.ALLOW_CHILDREN_WRITE);
        sceneTG.setCapability(TransformGroup.ALLOW_TRANSFORM_WRITE);
        sceneTG.setCapability(TransformGroup.ALLOW_TRANSFORM_READ);
        sceneTG.setCapability(TransformGroup.ENABLE_PICK_REPORTING);

        // add Child (0)
        Globals.scene.addChild(sceneTG);
    // create the Mouse Rotate Behavior
    MouseRotate myMouseRotate = new MouseRotate();
    myMouseRotate.setTransformGroup(sceneTG);
    myMouseRotate.setSchedulingBounds(new BoundingSphere());

    // add Child (1)
    Globals.scene.addChild(myMouseRotate);

    // create the Mouse Translate Behavior
    MouseTranslate myMouseTranslate = new MouseTranslate();
    myMouseTranslate.setTransformGroup(sceneTG);
    myMouseTranslate.setSchedulingBounds(new BoundingSphere());

    // add Child (2)
    Globals.scene.addChild(myMouseTranslate);

    // create the Mouse Zoom Behavior
    MouseZoom myMouseZoom = new MouseZoom();
    myMouseZoom.setTransformGroup(sceneTG);
    myMouseZoom.setSchedulingBounds(new BoundingSphere());

    // add Child (3)
    Globals.scene.addChild(myMouseZoom);

    // create the Picking Behavior
    MyPickBehavior myPick = new MyPickBehavior(Globals.scene,
        (Canvas3D)Globals.canvas3D, new BoundingSphere());

    // add Child (4)
    Globals.scene.addChild(myPick);

    // set initial rendering mode
    Globals.rendering = "LINE";
    Globals.shading = "NICEST";

    // compile the scene
    Globals.scene.compile();

    // SimpleUniverse is a Convenience Utility class
    SimpleUniverse simpleU = new SimpleUniverse((Canvas3D)(Globals.canvas3D));

    // this moves the ViewPlatform back a bit so the
    // objects in the scene can be viewed
    simpleU.getViewingPlatform().setNominalViewingTransform();

    // set Back Clip distance to 20 meters
    simpleU.getViewer().getView().setBackClipDistance(20.0);

    // add scene to universe
    simpleU.addBranchGraph(Globals.scene);
}
```

CREATE THE 3D SCENE

```
public static void createScene3D()
{
    String sRow = Globals.tfRow.getText().trim();
String sCol = Globals.tfCol.getText().trim();
String sHeight = Globals.tfInitHeight.getText().trim();
int iRow, iCol;
float fHeight;

// check data length
if (sRow.length() == 0 && sCol.length() == 0 && sHeight.length() == 0) return;

// check data format
try
{
     iRow = Integer.parseInt(sRow);
     iCol = Integer.parseInt(sCol);
     fHeight = Float.parseFloat(sHeight);

}
catch (NumberFormatException ex)
{
     Globals.tfRow.setText("");
     Globals.tfCol.setText("");
     Globals.tfInitHeight.setText("");

     Globals.taStatus.append("\nTRYING TO CREATE SCENE WITH INVALID PARAMETERS ->
          IGNORED\n");
     return;
}

// create Grid PolySound
int row = new Integer(Globals.tfRow.getText()).intValue();
int col = new Integer(Globals.tfCol.getText()).intValue();
float height = new Float(Globals.tfInitHeight.getText()).floatValue();

Method3D.createGridPoint(row + 1, col + 1, height);
Method3D.createGridPolySound(row + 1, col + 1, height);

// create axis
Method3D.createAxis(Globals.column, height + 1.0f, Globals.row);

// create text along axis
float valx = (float)(Globals.column) - 1.0f;
float valz = (float)(Globals.row) - 1.0f;


// add max values on each axis (with specified size)
int size = 90;
String s = String.valueOf(valx);
Method3D.createText3d(s, 'X', valx, Globals.white, size);
Method3D.createText3d((new Float(height + 1.0f)).toString(), 'Y',
     height + 1.0f, Globals.white, size);
s = String.valueOf(valz);
Method3D.createText3d(s, 'Z', valz, Globals.white, size);

// add axis names
Method3D.createText3d("X", 'X', valx + 2, Globals.red, 120);
Method3D.createText3d("Y", 'Y', height + 0.5f, Globals.red, 120);
Method3D.createText3d("Z", 'Z', valz + 2, Globals.red, 120);

// add lights
Color3f color = new Color3f(0.8f, 0.8f, 0.7f);
```

```
    Point3f attenuation = new Point3f(0.9f, 0.1f, 0.1f);
    int interval = 8;
    Method3D.addPointLights(color, attenuation, interval, height + 3.0f);

    // refresh the scene
    Method3D.refresh();

    // update buttons
    Globals.bCreate.setEnabled(false);
    Globals.bDelete.setEnabled(true);
    Globals.bLoad.setEnabled(false);
    Globals.bSave.setEnabled(true);
    Globals.tfRow.setEnabled(false);
    Globals.tfCol.setEnabled(false);
    Globals.tfInitHeight.setEnabled(false);

    /** enable sonification **/

    // open MIDI sequencer
    Sonification.openMidiSequencer();

    // enable sonification elements
    Sonification.enableSonificationComponents(true);
    Globals.bPlay.setEnabled(true);
    Globals.bPause.setEnabled(false);
    Globals.bStop.setEnabled(false);
}
```

DELETE THE 3D SCENE

```
public static void deleteScene3D()
{
    Method3D.delete("ALL");

    // close sequencer
    Sonification.closeMidiSequencer();

    // update buttons
    Globals.bCreate.setEnabled(true);
    Globals.bDelete.setEnabled(false);
    Globals.bLoad.setEnabled(true);
    Globals.bSave.setEnabled(false);
    Globals.tfY.setText("");
    Globals.tfY.setEnabled(false);
    Globals.bSet.setEnabled(false);
    Globals.tfRow.setEnabled(true);
    Globals.tfCol.setEnabled(true);
    Globals.tfInitHeight.setEnabled(true);

    // disable sonification elements
    Sonification.enableSonificationComponents(false);
    Globals.bPlay.setEnabled(false);
    Globals.bPause.setEnabled(false);
    Globals.bStop.setEnabled(false);
}
```

SAVE THE 3D SCENE

```
public static void saveScene3D()
{
    String sPath = new String();
    String sDirectory = new String();
    String sFileName = new String();

    // create a file chooser
    JFileChooser fc = new JFileChooser();

    int returnVal = fc.showSaveDialog(null);

    if (returnVal == JFileChooser.APPROVE_OPTION)
    {
        sDirectory = (fc.getCurrentDirectory()).getAbsolutePath();
        sFileName = (fc.getSelectedFile()).getName();
        sPath = (sDirectory + "\\" + sFileName);
        Globals.taStatus.append("\n Saving file : " + sPath + " ...");
    }
    else
    {
        Globals.taStatus.append("\n Save command cancelled by user.");
            return; .
    }

    // begin saving procedure
    try
    {
        // create file with specified path
        ChartFile.createBufferWriter(sPath);

        /** write file header **/

        String text = new String("SOUNDCHART 3D file");
        ChartFile.out.write(text, 0, text.length());
        ChartFile.out.newLine();
        ChartFile.out.write("BEGIN", 0, 5);
        ChartFile.out.newLine();

        /** write 3D data **/

        ChartFile.out.write(Globals.tfRow.getText(), 0,
            (Globals.tfRow.getText()).length());
        ChartFile.out.newLine();
        ChartFile.out.write(Globals.tfCol.getText(), 0,
            (Globals.tfCol.getText()).length());
        ChartFile.out.newLine();
        ChartFile.out.write(Globals.tfInitHeight.getText(), 0,
            (Globals.tfInitHeight.getText()).length());
        ChartFile.out.newLine();

        // select PolySound's BranchGroup
        TransformGroup TgChild = (TransformGroup)(Globals.scene.getChild(0));
        BranchGroup BgChild = (BranchGroup)(TgChild.getChild(0));

        for (int i = 0; i < BgChild.numChildren(); i++)
        {
            PolySound poly = (PolySound)(BgChild.getChild(i));
```

```
                for (int j = 0; j < 4; j++)
                {
                    text = (new Float(poly.points[j].y)).toString();
                    ChartFile.out.write(text, 0, text.length());
                    ChartFile.out.write("|", 0, 1);
                }
                ChartFile.out.newLine();
            }

            ChartFile.out.write("END", 0, 3);

            // close file
            ChartFile.out.close();
            Globals.taStatus.append("done\n");
        }
        catch (IOException ex)
        {
            Globals.taStatus.append("\nIO EXCEPTION WHILE SAVING FILE...\n");
        }
    }
```

LOAD A 3D SCENE

```
public static void loadScene3D()
{
    String sPath = new String();
    String sDirectory = new String();
    String sFileName = new String();

    // create a file chooser
    JFileChooser fc = new JFileChooser();

    int returnVal = fc.showOpenDialog(null);

    if (returnVal == JFileChooser.APPROVE_OPTION)
    {
        sDirectory = (fc.getCurrentDirectory()).getAbsolutePath();
        sFileName = (fc.getSelectedFile()).getName();
        sPath = (sDirectory + "\\" + sFileName);
        Globals.taStatus.append("\n Opening file : " + sPath + " ...");
    }
    else
    {
        Globals.taStatus.append("\n Open command cancelled by user.");
            return;
    }

    // begin loading procedure
    try
    {
        ChartFile.createBufferReader(sPath);

        Globals.sCurrentLine = ChartFile.in.readLine();

        ChartFile.readLine(Globals.sCurrentLine);
        if (!ChartFile.line.equals("SOUNDCHART 3D file"))
        {
            Globals.taStatus.append("\nError : WRONG FILE FORMAT !");
    return;
}

Globals.sCurrentLine = ChartFile.in.readLine();

ChartFile.readLine(Globals.sCurrentLine);
if (!ChartFile.line.equals("BEGIN"))
{
    Globals.taStatus.append("\nError : WRONG FILE FORMAT !");
    return;
}

// get row, col and initHeight
Globals.sCurrentLine = ChartFile.in.readLine();
ChartFile.readLine(Globals.sCurrentLine);
Globals.tfRow.setText(ChartFile.line);

Globals.sCurrentLine = ChartFile.in.readLine();
ChartFile.readLine(Globals.sCurrentLine);
Globals.tfCol.setText(ChartFile.line);

Globals.sCurrentLine = ChartFile.in.readLine();
ChartFile.readLine(Globals.sCurrentLine);
Globals.tfInitHeight.setText(ChartFile.line);

// create the beginning of the scene with these values
Scene3D.createScene3D();

// select PolySound's BranchGroup
TransformGroup TgChild = (TransformGroup)(Globals.scene.getChild(0));
BranchGroup BgChild = (BranchGroup)(TgChild.getChild(0));

while (((Globals.sCurrentLine = ChartFile.in.readLine()) ≠ null) &&
        !Globals.sCurrentLine.equals("END"))
{
    ChartFile.readData(Globals.sCurrentLine);

    PolySound poly = (PolySound)(BgChild.getChild(i));

    poly.setPointCoord(0, 'Y', ChartFile.read0);
    poly.setPointCoord(1, 'Y', ChartFile.read1);
    poly.setPointCoord(2, 'Y', ChartFile.read2);
    poly.setPointCoord(3, 'Y', ChartFile.read3);
}

if (!Globals.sCurrentLine.equals("END"))
{
    Globals.taStatus.append("\nError : WRONG FILE FORMAT !");
    return;
}

// update scene3D
Method3D.refresh();
Method3D.updateAllGCenters();

// find the height of the highest point of the whole scene after modification
float maxY = 0, currY;
for (int j = 0; j < Globals.grid.length; j++)
{
```

```
                currY = Globals.grid[j].y;
                if (currY > maxY) maxY = currY;
            }
            Globals.maxHeight = maxY;

            // update axis, texts and lights
            Method3D.updatePicking(Globals.maxHeight);

            // update colors
            Method3D.updateColors();

            // refresh scene
            Method3D.refresh();

            Globals.taStatus.append("done\n");

            /** enable sonification **/

            // open sequencer
            Sonification.openMidiSequencer();

            // update sonification elements
            Sonification.enableSonificationComponents(true);
            Globals.bPlay.setEnabled(true);
            Globals.bPause.setEnabled(false);
            Globals.bStop.setEnabled(false);
        }
        catch (IOException ex)
        {
            Globals.taStatus.append("\nIO EXCEPTION WHILE READING FILE...\n");
            Globals.taStatus.append("LINE : " + Globals.sCurrentLine + "\n");
            Scene3D.deleteScene3D();
            return;
        }
        catch (NumberFormatException ex)
        {
            Globals.taStatus.append("\nNUMBER FORMAT EXCEPTION WHILE READING FILE...\n");
            Globals.taStatus.append("LINE : " + Globals.sCurrentLine + "\n");
            Scene3D.deleteScene3D();
            return;
        }
    }
}
```

Method3D.java

The **Method3D** class contains several methods for handling the 3D scene, which are called from about every place in the program. While they are obviously all useful, the essential one is the **refresh** method, which is called very frequently by other SoundChart3D methods.

```
import com.sun.j3d.utils.applet.MainFrame;
import com.sun.j3d.utils.universe.*;
import com.sun.j3d.utils.geometry.ColorCube;
import com.sun.j3d.utils.geometry.Text2D;
import javax.media.j3d.*;
import javax.vecmath.*;
import java.awt.Font;
import java.util.Random;


public class Method3D
{
    // get a specified PolySound Child of the BG
    public static PolySound getPolySound(int index)
    {
        TransformGroup TgChild = (TransformGroup)(Globals.scene.getChild(0));

        BranchGroup BgChild = (BranchGroup)(TgChild.getChild(0));
        PolySound poly = (PolySound)(BgChild.getChild(index));

        return poly;
    }


    public static void createGridPoint(int row, int column, float height)
    {
        float z = 0.0f;
        int index = 0;

        Globals.grid = new Point3f[row * column];
        Globals.taStatus.append("\ncreating points grid...");

        for (int i = 0; i < row; i++)
        {
            float x = 0.0f;
            for (int j = 0; j < column; j++)
            {
                Globals.grid[index] = new Point3f(x, height, z);
                x += 1.0f;
                index ++;
            }
            z += 1.0f;
        }

        Globals.taStatus.append("done (" + Globals.grid.length + " points inserted)\n");
```

```
}

public static void refresh()
{
    // refresh PolySound BranchGroup
    TransformGroup TgChild = (TransformGroup)(Globals.scene.getChild(0));
    BranchGroup BgChild = (BranchGroup)(TgChild.getChild(0));

    for (int i = 0; i < BgChild.numChildren(); i++)
    {
        PolySound poly = (PolySound)(BgChild.getChild(i));
        IndexedQuadArray geom = (IndexedQuadArray)(poly.getGeometry());

        geom.setCoordinates(0, Globals.grid);

        geom.setCoordinateIndex(0, poly.indices[0]);
        geom.setCoordinateIndex(1, poly.indices[1]);
        geom.setCoordinateIndex(2, poly.indices[2]);
        geom.setCoordinateIndex(3, poly.indices[3]);

        geom.setColors(0, poly.colors);

        poly.setGeometry(geom);

        // get appearance
        Appearance Appear = poly.getAppearance();

        Material material = new Material();

        Color3f polycolor = new Color3f(poly.colors[0].x - 0.2f,
                                        poly.colors[0].y - 0.2f,
                                        poly.colors[0].z - 0.2f);

        material.setDiffuseColor(poly.colors[0]);
        material.setSpecularColor(poly.colors[0]);
        material.setShininess(10.0f);

        Appear.setMaterial(material);

        poly.setAppearance(Appear);
    }
}

public static void createGridPolySound(int row, int column, float height)
{
    Color3f color, lastcolor, nextcolor = new Color3f();

    Globals.taStatus.append("creating polysound grid...");

    Color3f green1 = new Color3f(0.0f, 1.0f, 0.0f);
    Color3f green2 = new Color3f(0.0f, 0.5f, 0.0f);

    Globals.row = row;
    Globals.column = column;

    // create BranchGroup
    BranchGroup bg = new BranchGroup();
```

```
bg.setCapability(BranchGroup.ALLOW_CHILDREN_READ);
bg.setCapability(BranchGroup.ALLOW_CHILDREN_WRITE);
bg.setCapability(BranchGroup.ALLOW_DETACH);

color = green1;
lastcolor = green1;
nextcolor = green1;

Globals.maxHeight = height;

// create grid PolySounds
for (int i = 0; i < (row - 1) * (column - 1); i++)
{
    color = nextcolor;

    bg.addChild(new PolySound( (row * column), 4, 0.1, Globals.level0));

    lastcolor = color;

    if (color.equals(green1)) nextcolor = green2;
    else nextcolor = green1;

    // if column is odd then the number of poly on a column is pair
    if ((column  2) == 1)
    {
        if (i ≠0 && (((i+1)  (column - 1)) == 0))
        {
            nextcolor = lastcolor;
        }
    }

    PolySound poly = (PolySound)(bg.getChild(i));
    IndexedQuadArray geom = (IndexedQuadArray)(poly.getGeometry());

    geom.setCoordinates(0, Globals.grid);

    // compute normals
    Vector3f n = new Vector3f(0.0f, height + 1.0f, 0.0f);
    n.normalize();

    geom.setNormal(0, n);
    geom.setNormal(1, n);
    geom.setNormal(2, n);
    geom.setNormal(3, n);

    // compute Coordinate Indices according to the grid
    int i0 = (column) * (i / (column - 1)) + (i  (column - 1)) + 1;
    int i1 = i0 + column;
    int i2 = i1 - 1;
    int i3 = i2 - column;

    geom.setCoordinateIndex( 0, i0);
    geom.setCoordinateIndex( 1, i1);
    geom.setCoordinateIndex( 2, i2);
    geom.setCoordinateIndex( 3, i3);

    poly.setGeometry(geom);

    poly.points[0] = Globals.grid[i0];
```

```
            poly.points[1] = Globals.grid[i1];
            poly.points[2] = Globals.grid[i2];
            poly.points[3] = Globals.grid[i3];

            poly.updateGcenter();

            poly.indices[0] = i0;
            poly.indices[1] = i1;
            poly.indices[2] = i2;
            poly.indices[3] = i3;

            poly.id = i;
        }

        TransformGroup TgChild = (TransformGroup)(Globals.scene.getChild(0));
        TgChild.insertChild(bg, 0);

        Globals.taStatus.append("done (" + bg.numChildren() + " PolySounds inserted)\n");
    }


    public static void addPointLights(Color3f color, Point3f attenuation,
        int interval, float height)
    {
        PointLight ptlight;

        Globals.taStatus.append("creating Point lights...");

        // create ambient light
        Globals.taStatus.append("creating Ambient light...");
        AmbientLight lightA = new AmbientLight(Globals.white);
        lightA.setInfluencingBounds(Globals.bounds);
        Globals.taStatus.append("done\n");

        // compute PointLight intervals
        int intX = Globals.column / interval;
        int intZ = Globals.row / interval;

        // add ambient light to TransformGroup
        TransformGroup TgChild = (TransformGroup)(Globals.scene.getChild(0));
        BranchGroup bg = new BranchGroup();
        bg.setCapability(BranchGroup.ALLOW_CHILDREN_READ);
        bg.setCapability(BranchGroup.ALLOW_CHILDREN_WRITE);
        bg.setCapability(BranchGroup.ALLOW_DETACH);
        bg.addChild(lightA);

        // add point lights
        for (int i = 0; i < intZ; i++)
        {
            for (int j = 0; j < intX; j++)
            {
                float x = 0.0f + (j+1) * interval;
                float y = height;
                float z = 0.0f + (i+1) * interval;

                Point3f position = new Point3f(x, y, z);
                ptlight = new PointLight(color, position, attenuation);
                ptlight.setCapability(PointLight.ALLOW_POSITION_READ);
                ptlight.setCapability(PointLight.ALLOW_POSITION_WRITE);
                ptlight.setInfluencingBounds(Globals.bounds);

                bg.addChild(ptlight);
            }
        }

        Globals.taStatus.append("done\n");
        Globals.taStatus.append("creating Spot light...");

        // create SpotLight
        SpotLight splight = new SpotLight(Globals.yellow,
            new Point3f((Globals.column / 2.0f), height + 7.0f, (Globals.row / 2.0f)),
            new Point3f(0.9f, 0.02f, 0.01f),
            new Vector3f((Globals.column / 2.0f), height - 3.0f, (Globals.column / 2.0f)),
            3.14f, 0.0f);

        // add SpotLight
        splight.setCapability(PointLight.ALLOW_POSITION_READ);
        splight.setCapability(PointLight.ALLOW_POSITION_WRITE);
        splight.setInfluencingBounds(Globals.bounds);
        bg.addChild(splight);

        Globals.taStatus.append("done\n");

        // add lights to scene
        TgChild.insertChild(bg, 8);

        Globals.taStatus.append("--- "+ bg.numChildren() + " lights inserted ---\n");
    }


    public static void addCorners(PolySound poly, Color3f color)
    {
        TransformGroup TgChild = (TransformGroup)(Globals.scene.getChild(0));

        // create BranchGroup for corners
        BranchGroup bg = new BranchGroup();
        bg.setCapability(BranchGroup.ALLOW_CHILDREN_READ);
        bg.setCapability(BranchGroup.ALLOW_CHILDREN_WRITE);
        bg.setCapability(BranchGroup.ALLOW_DETACH);

        // create and add corners
        for (int i = 0; i < 4; i++)
        {
            QuadArray pa = new QuadArray(4,
                QuadArray.COORDINATES | PointArray.NORMALS | PointArray.COLOR_3);
            pa.setCapability(QuadArray.ALLOW_COORDINATE_READ);
            pa.setCapability(QuadArray.ALLOW_COORDINATE_WRITE);
            pa.setCapability(QuadArray.ALLOW_COLOR_READ);
            pa.setCapability(QuadArray.ALLOW_COLOR_WRITE);
            pa.setCapability(QuadArray.ALLOW_COUNT_READ);
            pa.setCapability(QuadArray.ALLOW_FORMAT_READ);

            Point3f coord = poly.points[i];
            Point3f corner0 = new Point3f(coord.x + 0.1f, coord.y + 0.01f, coord.z - 0.1f);
            Point3f corner1 = new Point3f(coord.x + 0.1f, coord.y + 0.01f, coord.z + 0.1f);
            Point3f corner2 = new Point3f(coord.x - 0.1f, coord.y + 0.01f, coord.z + 0.1f);
            Point3f corner3 = new Point3f(coord.x - 0.1f, coord.y + 0.01f, coord.z - 0.1f);
            pa.setCoordinate(0, corner0);
            pa.setCoordinate(1, corner1);
            pa.setCoordinate(2, corner2);
            pa.setCoordinate(3, corner3);

            pa.setColor(0, color);
            pa.setColor(1, color);
            pa.setColor(2, color);
            pa.setColor(3, color);

            Appearance Appear = new Appearance();
            Appear.setCapability(Appearance.ALLOW_POLYGON_ATTRIBUTES_READ);
            Appear.setCapability(Appearance.ALLOW_POLYGON_ATTRIBUTES_WRITE);
            PolygonAttributes polyAttrib = new PolygonAttributes();
            polyAttrib.setCullFace(PolygonAttributes.CULL_NONE);
            Appear.setPolygonAttributes(polyAttrib);

            Shape3D Corner = new Shape3D(pa, Appear);
            Corner.setCapability(Shape3D.ALLOW_GEOMETRY_READ);
            Corner.setCapability(Shape3D.ALLOW_GEOMETRY_WRITE);

            bg.addChild(Corner);
        }

        // add corner BranchGroup to scene
        TgChild.insertChild(bg, 9);
    }


    public static void setNextRenderMode()
    {
        PolygonAttributes polyAttrib = new PolygonAttributes();
        polyAttrib.setCullFace(PolygonAttributes.CULL_NONE);

        if (Globals.rendering.equals("POINT"))
        {
            polyAttrib.setPolygonMode(PolygonAttributes.POLYGON_LINE);
            Globals.rendering = "LINE";
        }
        else if (Globals.rendering.equals("LINE"))
        {
            polyAttrib.setPolygonMode(PolygonAttributes.POLYGON_FILL);
            Globals.rendering = "FILL";
        }
        else if (Globals.rendering.equals("FILL"))
        {
            polyAttrib.setPolygonMode(PolygonAttributes.POLYGON_POINT);
            Globals.rendering = "POINT";
        }

        TransformGroup TgChild = (TransformGroup)(Globals.scene.getChild(0));
        BranchGroup BgChild = (BranchGroup)(TgChild.getChild(0));

        for (int i = 0; i < BgChild.numChildren(); i++)
        {
            PolySound poly = (PolySound)(BgChild.getChild(i));
            Appearance Appear = poly.getAppearance();
            Appear.setPolygonAttributes(polyAttrib);
            poly.setAppearance(Appear);
        }
    }


    public static void setNextShadeMode()
    {
        ColoringAttributes colorAttrib = new ColoringAttributes();

        if (Globals.shading.equals("FLAT"))
        {
            colorAttrib.setShadeModel(ColoringAttributes.NICEST);
            Globals.shading = "NICEST";
        }
        else if (Globals.shading.equals("NICEST"))
        {
            colorAttrib.setShadeModel(ColoringAttributes.SHADE_FLAT);
            Globals.shading = "FLAT";
        }

        TransformGroup TgChild = (TransformGroup)(Globals.scene.getChild(0));
        BranchGroup BgChild = (BranchGroup)(TgChild.getChild(0));

        for (int i = 0; i < BgChild.numChildren(); i++)
        {
            PolySound poly = (PolySound)(BgChild.getChild(i));
            Appearance Appear = poly.getAppearance();
            Appear.setColoringAttributes(colorAttrib);
            poly.setAppearance(Appear);
        }
    }


    public static void createAxis(float maxX, float maxY, float maxZ)
    {
        Globals.taStatus.append("creating axis...");

        TransformGroup TgChild = (TransformGroup)(Globals.scene.getChild(0));

        BranchGroup bg = new BranchGroup();
        bg.setCapability(BranchGroup.ALLOW_CHILDREN_READ);
        bg.setCapability(BranchGroup.ALLOW_CHILDREN_WRITE);
        bg.setCapability(BranchGroup.ALLOW_DETACH);

        Axis axis = new Axis(maxX, maxY, maxZ);

        bg.addChild(axis);

        // add axis to scene
        TgChild.insertChild(bg, 1);

        Globals.taStatus.append("done\n");
    }


    public static void createText3d(String s, char axis, float value, Color3f color, int size)
    {
        Globals.taStatus.append("creating text...");

        TransformGroup TgChild = (TransformGroup)(Globals.scene.getChild(0));
```

```
BranchGroup bg = new BranchGroup();
bg.setCapability(BranchGroup.ALLOW_CHILDREN_READ);
bg.setCapability(BranchGroup.ALLOW_CHILDREN_WRITE);
bg.setCapability(BranchGroup.ALLOW_DETACH);

String fontName = "Arial";//"Helvetica";
float sl = s.length();

// create Text2D object
Text2D text2D = new Text2D(s, color, fontName, size, Font.ITALIC);
text2D.setCapability(Shape3D.ALLOW_GEOMETRY_READ);
text2D.setCapability(Shape3D.ALLOW_GEOMETRY_WRITE);
text2D.setCapability(Shape3D.ALLOW_APPEARANCE_READ);
text2D.setCapability(Shape3D.ALLOW_APPEARANCE_WRITE);
text2D.setPickable(false);

QuadArray qa = (QuadArray)text2D.getGeometry();

qa.setCapability(QuadArray.ALLOW_COUNT_READ);
qa.setCapability(QuadArray.ALLOW_FORMAT_READ);
qa.setCapability(QuadArray.ALLOW_COORDINATE_READ);

text2D.setGeometry(qa);

Appearance textAppear = text2D.getAppearance();

// make the Text2D object 2-sided.
PolygonAttributes polyAttrib = new PolygonAttributes();
polyAttrib.setCullFace(PolygonAttributes.CULL_NONE);
polyAttrib.setBackFaceNormalFlip(true);
textAppear.setPolygonAttributes(polyAttrib);

text2D.setAppearance(textAppear);

TransformGroup bbTransY = new TransformGroup();
bbTransY.setCapability(TransformGroup.ALLOW_TRANSFORM_WRITE);
bbTransY.setCapability(TransformGroup.ALLOW_TRANSFORM_READ);
bbTransY.setCapability(TransformGroup.ALLOW_CHILDREN_READ);

Billboard bboardY = new Billboard(bbTransY);

bg.addChild(bboardY);

bg.addChild(bbTransY);

// define the translation
    Transform3D textTranslation = new Transform3D();
TransformGroup textTranslationGroup;

if (axis == 'x' || axis == 'X')
{
    Billboard bb = (Billboard)bg.getChild(0);

    bb.setSchedulingBounds(new BoundingSphere());
    bb.setAlignmentMode(Billboard.ROTATE_ABOUT_POINT);
    bb.setRotationPoint(value, 0.0f, -1.0f);

    textTranslation.setTranslation(new Vector3f(value, 0.0f, 0.0f));
```

```
            textTranslationGroup = new TransformGroup(textTranslation);
            textTranslationGroup.setCapability(TransformGroup.ALLOW_TRANSFORM_WRITE);
            textTranslationGroup.setCapability(TransformGroup.ALLOW_CHILDREN_READ);
            textTranslationGroup.setCapability(TransformGroup.ALLOW_CHILDREN_WRITE);

            textTranslationGroup.addChild(text2D);
            bbTransY.addChild(textTranslationGroup);
        }
        else if (axis == 'y' || axis == 'Y')
        {
            Billboard bb = (Billboard)bg.getChild(0);

            bb.setSchedulingBounds(new BoundingSphere());
            bb.setAlignmentMode(Billboard.ROTATE_ABOUT_POINT);
            bb.setRotationPoint(0.0f, value, 0.0f);

            textTranslation.setTranslation(new Vector3f(0.0f, value, 0.0f));
            textTranslationGroup = new TransformGroup(textTranslation);
            textTranslationGroup.setCapability(TransformGroup.ALLOW_TRANSFORM_WRITE);
            textTranslationGroup.setCapability(TransformGroup.ALLOW_CHILDREN_READ);
            textTranslationGroup.setCapability(TransformGroup.ALLOW_CHILDREN_WRITE);
            textTranslationGroup.addChild(text2D);
            bbTransY.addChild(textTranslationGroup);
        }
        else if (axis == 'z' || axis == 'Z')
        {
            Billboard bb = (Billboard)bg.getChild(0);

            bb.setSchedulingBounds(new BoundingSphere());
            bb.setAlignmentMode(Billboard.ROTATE_ABOUT_POINT);
            bb.setRotationPoint(-0.5f, 0.0f, value);

            textTranslation.setTranslation(new Vector3f(0.0f, 0.0f, value));
            textTranslationGroup = new TransformGroup(textTranslation);
            textTranslationGroup.setCapability(TransformGroup.ALLOW_TRANSFORM_WRITE);
            textTranslationGroup.setCapability(TransformGroup.ALLOW_CHILDREN_READ);
            textTranslationGroup.setCapability(TransformGroup.ALLOW_CHILDREN_WRITE);
            textTranslationGroup.addChild(text2D);
            bbTransY.addChild(textTranslationGroup);
        }

        TgChild.insertChild(bg, 2);

        Globals.taStatus.append("done\n");
    }


    public static void delete(String s)
    {
        TransformGroup TgChild = (TransformGroup)(Globals.scene.getChild(0));

        if (s.equals("GRID")) TgChild.removeChild(0);
        else if (s.equals("AXIS")) TgChild.removeChild(1);
        else if (s.equals("TEXT"))
        {
            TgChild.removeChild(7);
            TgChild.removeChild(6);
            TgChild.removeChild(5);
            TgChild.removeChild(4);
        TgChild.removeChild(3);
        TgChild.removeChild(2);
    }
    else if (s.equals("LIGHT")) TgChild.removeChild(8);
    else if (s.equals("CORNERS")) TgChild.removeChild(9);
    else if (s.equals("ALL"))
    {
        Globals.taStatus.append("deleting whole scene...");
        for (int i = TgChild.numChildren() - 1; i ≥ 0; i–)
            TgChild.removeChild(i);

        Globals.taStatus.append("done\n");
        Globals.taStatus.append("--- "+ TgChild.numChildren() +
            " children remaining ---\n");
    }
}


public static void updatePicking(float coordY)
{
    // update PointLights if maxY ¿ 3 meters
    if (coordY > 3.0f)
    {
        TransformGroup TgChild = (TransformGroup)(Globals.scene.getChild(0));
        BranchGroup bg = (BranchGroup)(TgChild.getChild(8));

        Point3f coord = new Point3f();
        // start at 1 because light 0 is an AmbientLight
        for (int i = 1; i < bg.numChildren(); i++)
        {
            PointLight pl = (PointLight)(bg.getChild(i));
            pl.getPosition(coord);
            coord.y = coordY + 1.0f;
            pl.setPosition(coord);
        }

        SpotLight sp = (SpotLight)(bg.getChild(bg.numChildren() - 1));
        sp.getPosition(coord);
        coord.y = coordY + 1.0f;
        sp.setPosition(coord);
    }

    // update Axis
    Method3D.delete("AXIS");
    Method3D.createAxis(Globals.column, coordY, Globals.row);

    // update Text along Axis
    TransformGroup TgChild = (TransformGroup)(Globals.scene.getChild(0));

    // "Y"
    BranchGroup bg = (BranchGroup)(TgChild.getChild(3));
    Billboard bb = (Billboard)(bg.getChild(0));
    bb.setRotationPoint(0.0f, coordY + 0.5f, 0.0f);
    TransformGroup Tg2 = (TransformGroup)(bg.getChild(1));
    TransformGroup Tg3 = (TransformGroup)(Tg2.getChild(0));
    Transform3D textTranslation = new Transform3D();
    textTranslation.setTranslation(new Vector3f(0.0f, coordY + 0.5f, 0.0f));
    Tg3.setTransform(textTranslation);
```

```
        // Y val
        bg = (BranchGroup)(TgChild.getChild(6));
        bb = (Billboard)(bg.getChild(0));
        bb.setRotationPoint(0.0f, coordY, 0.0f);
        Tg2 = (TransformGroup)(bg.getChild(1));
        Tg3 = (TransformGroup)(Tg2.getChild(0));
        textTranslation = new Transform3D();
        textTranslation.setTranslation(new Vector3f(0.0f, coordY, 0.0f));
        Tg3.setTransform(textTranslation);

        Text2D t2d = (Text2D)(Tg3.getChild(0));
        t2d.setString(new Float(coordY).toString());
    }


    public static void updateColors()
    {
        TransformGroup TgChild = (TransformGroup)(Globals.scene.getChild(0));
        BranchGroup BgChild = (BranchGroup)(TgChild.getChild(0));

        for (int i = 0; i < BgChild.numChildren(); i++)
        {
            PolySound poly = (PolySound)(BgChild.getChild(i));

            float gY = poly.points[0].y;
            for (int j = 1; j < 4; j++)
                if (poly.points[j].y > gY) gY = poly.points[j].y;

            if (gY > 5.5) poly.setColor(Globals.level10);
            else if (gY > 4.5) poly.setColor(Globals.level8);
            else if (gY > 3) poly.setColor(Globals.level6);
            else if (gY > 1.5) poly.setColor(Globals.level4);
            else if (gY > 0) poly.setColor(Globals.level2);
            else if (gY == 0) poly.setColor(Globals.level0);
        }
    }


    public static void updateAllGCenters()
    {
        TransformGroup TgChild = (TransformGroup)(Globals.scene.getChild(0));
        BranchGroup BgChild = (BranchGroup)(TgChild.getChild(0));

        for (int i = 0; i < BgChild.numChildren(); i++)
        {
            PolySound poly = (PolySound)(BgChild.getChild(i));

            // re-compute gravity centers
            poly.gcenter.x = (poly.points[0].x + poly.points[1].x +
                    poly.points[2].x + poly.points[3].x) / 4.0f;
            poly.gcenter.y = (poly.points[0].y + poly.points[1].y +
                    poly.points[2].y + poly.points[3].y) / 4.0f;
            poly.gcenter.z = (poly.points[0].z + poly.points[1].z +
                    poly.points[2].z + poly.points[3].z) / 4.0f;
        }
    }
}
```

Axis.java

The `Axis` class creates the 3D axes. The constructor is called by the `createAxis` method in the `Method3D` class. After the geometry of the axes is created, the line attributes and appearance are defined.

```
import javax.media.j3d.*;
import javax.vecmath.*;
import javax.media.j3d.LineAttributes;


public class Axis extends Shape3D
{
    public Axis(float maxX, float maxY, float maxZ)
    {
        // create geometry
        this.setGeometry(createGeometry(maxX, maxY, maxZ));
        this.setCapability(ALLOW_GEOMETRY_READ);

        // set line attributes
        LineAttributes lineAttrib = new LineAttributes(0.5f,
            LineAttributes.PATTERN_SOLID, true);
        ColoringAttributes colorAttrib = new ColoringAttributes();
        colorAttrib.setShadeModel(ColoringAttributes.NICEST);

        // set appearance
        Appearance appear = new Appearance();
        appear.setLineAttributes(lineAttrib);
        appear.setColoringAttributes(colorAttrib);
        this.setAppearance(appear);
    }


    private Geometry createGeometry(float maxX, float maxY, float maxZ)
    {
        // create lines geometry
        IndexedLineArray axisLines = new IndexedLineArray(18, GeometryArray.COORDINATES, 30);

        // set capabilities
        axisLines.setCapability(IndexedLineArray.ALLOW_COUNT_READ);
        axisLines.setCapability(IndexedLineArray.ALLOW_FORMAT_READ);
        axisLines.setCapability(IndexedLineArray.ALLOW_COORDINATE_READ);
        axisLines.setCapability(IndexedLineArray.ALLOW_COORDINATE_INDEX_READ);

        // set X-axis coordinates
        axisLines.setCoordinate( 0, new Point3f(-1.0f, 0.0f, 0.0f));
        axisLines.setCoordinate( 1, new Point3f( maxX, 0.0f, 0.0f));
        axisLines.setCoordinate( 2, new Point3f( maxX - 0.1f, 0.1f, 0.1f));
        axisLines.setCoordinate( 3, new Point3f( maxX - 0.1f, -0.1f, 0.1f));
        axisLines.setCoordinate( 4, new Point3f( maxX - 0.1f, 0.1f,-0.1f));
        axisLines.setCoordinate( 5, new Point3f( maxX - 0.1f, -0.1f,-0.1f));
        // set Y-axis coordinates
        axisLines.setCoordinate( 6, new Point3f( 0.0f,-1.0f, 0.0f));
        axisLines.setCoordinate( 7, new Point3f( 0.0f, maxY, 0.0f));
        axisLines.setCoordinate( 8, new Point3f( 0.1f, maxY - 0.1f, 0.1f));
        axisLines.setCoordinate( 9, new Point3f(-0.1f, maxY - 0.1f, 0.1f));
        axisLines.setCoordinate(10, new Point3f( 0.1f, maxY - 0.1f, -0.1f));
        axisLines.setCoordinate(11, new Point3f(-0.1f, maxY - 0.1f, -0.1f));

        // set Z-axis coordinates
        axisLines.setCoordinate(12, new Point3f( 0.0f, 0.0f,-1.0f));
        axisLines.setCoordinate(13, new Point3f( 0.0f, 0.0f, maxZ));
        axisLines.setCoordinate(14, new Point3f( 0.1f, 0.1f, maxZ - 0.1f));
        axisLines.setCoordinate(15, new Point3f(-0.1f, 0.1f, maxZ - 0.1f));
        axisLines.setCoordinate(16, new Point3f( 0.1f,-0.1f, maxZ - 0.1f));
        axisLines.setCoordinate(17, new Point3f(-0.1f,-0.1f, maxZ - 0.1f));

        // set coordinate indices
        axisLines.setCoordinateIndex( 0, 0);
        axisLines.setCoordinateIndex( 1, 1);
        axisLines.setCoordinateIndex( 2, 2);
        axisLines.setCoordinateIndex( 3, 1);
        axisLines.setCoordinateIndex( 4, 3);
        axisLines.setCoordinateIndex( 5, 1);
        axisLines.setCoordinateIndex( 6, 4);
        axisLines.setCoordinateIndex( 7, 1);
        axisLines.setCoordinateIndex( 8, 5);
        axisLines.setCoordinateIndex( 9, 1);
        axisLines.setCoordinateIndex(10, 6);
        axisLines.setCoordinateIndex(11, 7);
        axisLines.setCoordinateIndex(12, 8);
        axisLines.setCoordinateIndex(13, 7);
        axisLines.setCoordinateIndex(14, 9);
        axisLines.setCoordinateIndex(15, 7);
        axisLines.setCoordinateIndex(16,10);
        axisLines.setCoordinateIndex(17, 7);
        axisLines.setCoordinateIndex(18,11);
        axisLines.setCoordinateIndex(19, 7);
        axisLines.setCoordinateIndex(20,12);
        axisLines.setCoordinateIndex(21,13);
        axisLines.setCoordinateIndex(22,14);
        axisLines.setCoordinateIndex(23,13);
        axisLines.setCoordinateIndex(24,15);
        axisLines.setCoordinateIndex(25,13);
        axisLines.setCoordinateIndex(26,16);
        axisLines.setCoordinateIndex(27,13);
        axisLines.setCoordinateIndex(28,17);
        axisLines.setCoordinateIndex(29,13);

        return axisLines;
    }
}
```

ChartFile.java

Like in SoundChart, the `ChartFile` class is used to read data from a file. However, it is also possible to write data to a file, since SoundChart3D enables the user to save the 3D scene. The `readData` method reads one line from the file, and stores the four values in the corresponding variables. The last two methods are only called once, since they initialize the file pointers which will be used to access the files denoted by `sPath`.

```
import javax.swing.*;
import java.awt.*;
import java.util.StringTokenizer;
import java.util.*;
import java.io.*;
import java.lang.Double;


public class ChartFile
{
    public static void readData(String s)
    {
        StringTokenizer t = new StringTokenizer(s, "|");

        read0 = Float.parseFloat(t.nextToken());
        read1 = Float.parseFloat(t.nextToken());
        read2 = Float.parseFloat(t.nextToken());
        read3 = Float.parseFloat(t.nextToken());
    }


    public static void readLine(String s)
    {
        StringTokenizer t = new StringTokenizer(s, "\n");

        line = (String)(t.nextToken());
    }


    public static void createBufferReader(String sPath)
    {
        try
        {
            in = new BufferedReader(new FileReader(sPath));
        }
        catch (IOException e)
        {
            System.out.println("ERROR in createBufferReader(): " + e);
            System.exit(0);
        }
    }
    public static void createBufferWriter(String sPath)
    {
        try
        {
            out = new BufferedWriter(new FileWriter(sPath));
        }
        catch (IOException e)
        {
            System.out.println("ERROR in createBufferWriter(): " + e);
            System.exit(0);
        }
    }

    public static float read0, read1, read2, read3;
    public static String line;
    public static BufferedReader in;
    public static BufferedWriter out;
}
```

KbListener.java

The `KbListener` class enables SoundChart3D to be notified when the user presses a certain key. The two methods are automatically called by Java when a keypress occurs. The `keyTyped` method is called when "normal" keys are pressed (such as letters), whereas `keyPressed` is used for special keys (such as function keys).

```
import java.awt.event.*;
import java.awt.event.KeyEvent;
import com.sun.j3d.utils.applet.MainFrame;
import com.sun.j3d.utils.universe.*;
import com.sun.j3d.utils.geometry.ColorCube;
import com.sun.j3d.utils.geometry.Text2D;
import javax.media.j3d.*;
import javax.vecmath.*;
import java.awt.Font;
import javax.media.j3d.Canvas3D;
import java.awt.Component;


class KbListener implements KeyListener
{
    public void keyTyped(KeyEvent e)
    {
        char source = e.getKeyChar();

        if (source == 'r' || source == 'R')
        {
            // refresh the scene
            Method3D.refresh();
        }
    }


    public void keyPressed(KeyEvent e)
    {
        int source = e.getKeyCode();

        if (source == KeyEvent.VK_F3)
        {
            // set next shading mode
            Method3D.setNextShadeMode();
        }

        if (source == KeyEvent.VK_F4)
        {
            // set next rendering mode
            Method3D.setNextRenderMode();
        }

        if (source == KeyEvent.VK_ESCAPE)
        {
            // deselect PolySound and corners
            TransformGroup TgChild = (TransformGroup)(Globals.scene.getChild(0));
            if (TgChild.numChildren() == 10)
            {
                Method3D.delete("CORNERS");

                // update Edition panel
                Globals.tfX.setText("");
                Globals.tfY.setText("");
                Globals.tfZ.setText("");
                Globals.tfY.setEnabled(false);
                Globals.bSet.setEnabled(false);
                Globals.ltfSelection.setText("Ctrl + Left click to select");

                Globals.idSelectedPoly = -1;
                Globals.idSelectedCorner = -1;
            }
        }
    }

    // must be declared, but not used by SoundChart3D
    public void keyReleased(KeyEvent e) {}
}
```

MyPickBehavior.java

The `MyPickBehavior` class enables the user to pick a PolySound (or some of its corners) by clicking on it with the mouse. Since the mouse is also used to move and rotate the 3D scene, additional keys are needed when selecting a PolySound (i.e. Alt and Control keys).

```
import com.sun.j3d.utils.picking.*;
import com.sun.j3d.utils.behaviors.mouse.*;
import java.awt.*;
import java.awt.event.*;
import java.util.*;
import javax.media.j3d.*;
import javax.vecmath.*;
import com.sun.j3d.utils.picking.behaviors.*;
import com.sun.j3d.utils.geometry.Text2D;


public class MyPickBehavior extends PickMouseBehavior
{
    int pickMode = PickTool.GEOMETRY;
    private PickingCallback callback = null;
    private TransformGroup currentTG;

    public MyPickBehavior(BranchGroup root, Canvas3D canvas, Bounds bounds)
    {
        super(canvas, root, bounds);
        this.setSchedulingBounds(bounds);
    }


    public void updateScene(int xpos, int ypos)
    {
        PolySound poly = null;
        Shape3D corner = null;

        if (!mevent.isMetaDown() && !mevent.isAltDown() && mevent.isControlDown())
        {
            pickCanvas.setShapeLocation(mevent);
            pickCanvas.setMode(PickTool.GEOMETRY_INTERSECT_INFO);
            pickCanvas.setTolerance(0.0f);

            TransformGroup TgChild = (TransformGroup)(Globals.scene.getChild(0));

            PickResult pr = pickCanvas.pickClosest();
            String classPicked = new String();
            if (pr ≠ null) classPicked = pr.getNode(PickResult.SHAPE3D).getClass().getName();

            /** POLYSOUNDS PICKING **/

            if ((pr ≠ null) &&
                (classPicked.equals("PolySound")) &&
        ((poly = (PolySound)pr.getNode(PickResult.SHAPE3D)) ≠ null) &&
        (poly.getCapability(BranchGroup.ALLOW_CHILDREN_READ)) &&
        (poly.getCapability(BranchGroup.ALLOW_CHILDREN_WRITE)))
    {
        // display corners of this PolySound
        if (TgChild.numChildren() < 10)
        {
            Method3D.addCorners(poly, Globals.magenta);
        }
        else
        {
            Method3D.delete("CORNERS");
            Method3D.addCorners(poly, Globals.magenta);
        }

        // update Edition panel
        Globals.tfX.setText((new Float(poly.gcenter.x)).toString());
        Globals.tfY.setText((new Float(poly.gcenter.y)).toString());
        Globals.tfZ.setText((new Float(poly.gcenter.z)).toString());

        Globals.ltfSelection.setText("PolySound's center");

        Globals.tfY.setEnabled(true);
        Globals.bSet.setEnabled(true);
        Globals.tfY.requestFocus();
        Globals.tfY.selectAll();

        // update selection variables
        Globals.idSelectedPoly = poly.id;
        Globals.idSelectedCorner = -1;
    }

    /** CORNER PICKING **/

    else if ((pr ≠ null) &&
            (classPicked.equals("javax.media.j3d.Shape3D")) &&
            ((corner = (Shape3D)pr.getNode(PickResult.SHAPE3D)) ≠ null))
    {
        Globals.taStatus.append("Corner selected ->");

        BranchGroup BgChild = (BranchGroup)(TgChild.getChild(9));
        int index = -1;

        for (int i = 0; i < BgChild.numChildren(); i++)
        {
            if ((BgChild.getChild(i)).equals(corner))
                index = i;
        }
        Globals.taStatus.append("point # " + index + "\n");

        for (int i = 0; i < BgChild.numChildren(); i++)
        {
            Shape3D shape = (Shape3D)BgChild.getChild(i);

            Color3f color = new Color3f();

            if (i ≠ index) color = new Color3f(0.46f, 0.0f, 0.34f);
            else color = Globals.magenta;
```

```
                QuadArray qa = (QuadArray)shape.getGeometry();
                qa.setColor(0, color);
                qa.setColor(1, color);
                qa.setColor(2, color);
                qa.setColor(3, color);

                shape.setGeometry(qa);
            }

            // update Edition panel
            poly = Method3D.getPolySound(Globals.idSelectedPoly);
            Globals.tfX.setText((new Float(poly.points[index].x)).toString());
            Globals.tfY.setText((new Float(poly.points[index].y)).toString());
            Globals.tfZ.setText((new Float(poly.points[index].z)).toString());

            Globals.ltfSelection.setText("PolySound's point #" + index);

            Globals.tfY.setEnabled(true);
            Globals.bSet.setEnabled(true);
            Globals.tfY.requestFocus();
            Globals.tfY.selectAll();

            // update global var
            Globals.idSelectedCorner = index;

        }

        else if (callback ≠ null)
        {
            callback.transformChanged(PickingCallback.NO_PICK, null);
        }
    }
}


// not needed, so doesn't do anything
public void transformChanged(int type, Transform3D transform) {}
}
```

Sonification.java

The SoundChart3D `Sonification` class is very similar to its two-dimensional counterpart. The only big difference is the creation of sound data. Indeed, the 3D scene can be travelled in three ways, resulting in three different methods for creating the data (horizontal, vertical and diagonal travelling). Another difference is that extreme values are not detected.

```
import javax.swing.*;
import javax.swing.border.*;
import javax.swing.JFileChooser.*;
import javax.sound.midi.*;
import java.awt.event.*;
import java.awt.*;
import java.awt.Graphics.*;
import java.awt.geom.*;
import java.util.Vector;
import java.lang.Double;
import java.io.*;
import java.util.*;
import javax.media.j3d.*;


class SequenceEndListener implements ActionListener
{
    public void actionPerformed(ActionEvent e)
    {
        // are we finished playing the current sequence ?
        if (!Globals.midiSequencer.isRunning())
        {
            // stop sonification
            Sonification.stop();

            // update GUI elements
            Sonification.enableSonificationComponents(true);
            Globals.bPlay.setEnabled(true);
            Globals.bPause.setEnabled(false);
            Globals.bStop.setEnabled(false);
            if (Globals.idSelectedPoly ≠ -1)
            {
                Globals.tfY.setEnabled(true);
                Globals.bSet.setEnabled(true);
            }
            Globals.bSave.setEnabled(true);
            Globals.bDelete.setEnabled(true);
        }
    }
}


public class Sonification
{
```

```
private static double dMaxY = Double.MIN_VALUE, dMinY = Double.MAX_VALUE;

private static void updateMaxMinY(float y)
{
    if (y > dMaxY) dMaxY = (double)y;
    if (y < dMinY) dMinY = (double)y;
}


private static long Msec2Tick(long msec)
{
    return (long)(msec * 0.2);
}


private static MidiEvent createMidiEvent(int channel, int command,
    int data1, int data2, long msec)
{
    MidiEvent event = null;

    try
    {
        // create message
        ShortMessage msg = new ShortMessage();
        msg.setMessage(command, channel, data1, data2);

        // create event
        event = new MidiEvent(msg, Msec2Tick(msec));
    }
    catch (InvalidMidiDataException ex)
    {
        System.out.println("\nERROR in createMidiEvent(): " + ex + "\n");
        System.exit(0);
    }

    return event;
}


private static Vector createHorizontalData()
{
    Vector vData = new Vector(), vLine = null;
    PolySound ps;
    float coords[], lastY = Float.MIN_VALUE, lastZ = Float.MIN_VALUE;
    int nbPolys = -1, lastLineChangeIndex = 0;

    // get nb of polysounds
    TransformGroup TgChild = (TransformGroup)(Globals.scene.getChild(0));
    BranchGroup BgChild = (BranchGroup)(TgChild.getChild(0));
    nbPolys = BgChild.numChildren();

    for (int i = 0; i < nbPolys; i++)
    {
        // get ith PolySound
        ps = Method3D.getPolySound(i);

        // get coords from 3rd point
        coords = ps.getPointCoord(3);
        // are we still on same line ?
        if (coords[2] == lastZ)
        {
            // add Y coord to current line
            vLine.add(new Float(coords[1]));
            updateMaxMinY(coords[1]);
        }
        else
        {
            lastLineChangeIndex = i;

            // finish last line
            if (lastY ≠ Float.MIN_VALUE)
            {
                vLine.add(new Float(lastY));
                updateMaxMinY(lastY);
                vData.add(vLine);
            }

            // create new line
            vLine = new Vector();

            // add Y coord to new line
            vLine.add(new Float(coords[1]));
            updateMaxMinY(coords[1]);

            // update last Z coord
            lastZ = coords[2];
        }

        // save last Y coord from 2nd point on poly
        float coords2[] = ps.getPointCoord(2);
        lastY = coords2[1];
    }

    // finish current line
    if (lastY ≠ Float.MIN_VALUE)
    {
        vLine.add(new Float(lastY));
        updateMaxMinY(lastY);
        vData.add(vLine);
    }

    // create new line
    vLine = new Vector();

    // handle last line
    for (int i = lastLineChangeIndex; i < nbPolys; i++)
    {
        // get ith PolySound
        ps = Method3D.getPolySound(i);

        // get coords from 0th point
        coords = ps.getPointCoord(0);

        // add Y coord to current line
        vLine.add(new Float(coords[1]));
        updateMaxMinY(coords[1]);
```

```
            // save last Y coord from 1st point on poly
            float coords2[] = ps.getPointCoord(1);
            lastY = coords2[1];
        }

        // finish current line
        if (lastY ≠ Float.MIN_VALUE)
        {
            vLine.add(new Float(lastY));
            updateMaxMinY(lastY);
            vData.add(vLine);
        }

        return vData;
    }


    private static Vector createVerticalData()
    {
        Vector vData = new Vector();
        Vector vHorData = createHorizontalData();
        Vector vLine = null;

        int nbHorRows = vHorData.size();
        int nbHorCols = ((Vector)vHorData.elementAt(0)).size();

        // each horizontal columns must become a vertical row
        for (int i = 0; i < nbHorCols; i++)
        {
            vLine = new Vector();

            for (int j = 0; j < nbHorRows; j++)
            {
                Vector vHorLine = (Vector)vHorData.elementAt(j);
                vLine.add(vHorLine.elementAt(i));
            }

            vData.add(vLine);
        }

        return vData;
    }


    private static Vector createDiagonalData()
    {
        Vector vData = new Vector();
        Vector vHorData = createHorizontalData();
        Vector vLine = null;
        int dx, dy;

        int nbHorRows = vHorData.size();
        int nbHorCols = ((Vector)vHorData.elementAt(0)).size();

        for (int i = 0; i < nbHorRows; i++)
        {
            vLine = new Vector();
            Vector vHorLine = (Vector)vHorData.elementAt(i);
            vLine.add(vHorLine.elementAt(0));
        dx = 1;
        dy = i - 1;

        while (dx < nbHorCols && dy ≥ 0)
        {
            Vector vLineY = (Vector)vHorData.elementAt(dy);
            vLine.add(vLineY.elementAt(dx));
            dx++;
            dy–;
        }

        vData.add(vLine);
    }

    // get last hor row
    Vector vHorLine = (Vector)vHorData.elementAt(nbHorRows-1);

    // handle last horizontal row
    for (int j = 1; j < nbHorCols; j++)
    {
        vLine = new Vector();
        vLine.add(vHorLine.elementAt(j));

        dx = j + 1;
        dy = nbHorRows - 2;

        while (dx < nbHorCols && dy ≥ 0)
        {
            Vector vLineY = (Vector)vHorData.elementAt(dy);
            vLine.add(vLineY.elementAt(dx));
            dx++;
            dy–;
        }

        vData.add(vLine);
    }

    return vData;
}


private static void createMidiSequence()
{
    int iProgram, iInterval, iMinNote, iMaxNote, iNote, iTime = 0, iLastDrum = -5000;
    int iEolProgram = 116, iDrumProgram = 47;
    int lastLineEndTime = 0;
    Vector vDataSonify = null;

    Globals.taStatus.append("\nCreating MIDI sequence:\n");

    // get the program number
    iProgram = Globals.cbProgram.getSelectedIndex();
    if (iProgram == 0 || (iProgram > 8 && iProgram  9 == 0)) iProgram++;
    if (iProgram < 9) iProgram–;
    else iProgram = iProgram - (iProgram / 9) - 1;
    Globals.taStatus.append("\tprogram number: " + iProgram + "\n");

    if (Globals.chbDrumBeats.isSelected())
```

```
{
    // get the drum program number
    iDrumProgram = Globals.cbDrumProgram.getSelectedIndex();
    if (iDrumProgram == 0 || (iDrumProgram > 8 && iDrumProgram  9 ==
        0)) iDrumProgram++;
    if (iDrumProgram < 9) iDrumProgram–;
    else iDrumProgram = iDrumProgram - (iDrumProgram / 9) - 1;
    Globals.taStatus.append("\tdrum program number: " + iDrumProgram + "\n");
}

if (Globals.chbEndLine.isSelected())
{
    // get the warnings program number
    iEolProgram = Globals.cbEolProgram.getSelectedIndex();
    if (iEolProgram == 0 || (iEolProgram > 8 && iEolProgram  9 == 0)) iEolProgram++;
    if (iEolProgram < 9) iEolProgram–;
    else iEolProgram = iEolProgram - (iEolProgram / 9) - 1;
    Globals.taStatus.append("\tEnd of line program number: " + iEolProgram + "\n");
}

// get data to sonify based on sonification type
String SItem = (String)(Globals.cbSonificationType.getSelectedItem());
if (SItem.equals("Horizontal Travelling (along X-axis)"))
{
    vDataSonify = createHorizontalData();
    Globals.taStatus.append("\tselected data: Horizontal Travelling\n");
}
else if (SItem.equals("Vertical Travelling (along Z-axis)"))
{
    vDataSonify = createVerticalData();
    Globals.taStatus.append("\tselected data: Vertical Travelling\n");
}
else if (SItem.equals("Diagonal Travelling"))
{
    vDataSonify = createDiagonalData();
    Globals.taStatus.append("\tselected data: Diagonal Travelling\n");
}
else
{
    System.out.println("\nERROR in createMidiSequence(): Wrong sonification type\n");
    System.exit(0);
}

// get sonification settings
iInterval = Globals.slTime.getValue();
iMinNote = Globals.slMinNote.getValue();
iMaxNote = Globals.slMaxNote.getValue();
Globals.taStatus.append("\tinterval: " + iInterval + "\n");
Globals.taStatus.append("\tminnote: " + iMinNote + "\n");
Globals.taStatus.append("\tmaxnote: " + iMaxNote + "\n");

// create MIDI sequence
try
{
    Globals.midiSequence = new Sequence(Sequence.PPQ, 100);
    Track track = Globals.midiSequence.createTrack();

    // set program numbers
    track.add(createMidiEvent(0, ShortMessage.PROGRAM_CHANGE, iProgram, 0, 0));
track.add(createMidiEvent(1, ShortMessage.PROGRAM_CHANGE, iEolProgram, 0, 0));
track.add(createMidiEvent(2, ShortMessage.PROGRAM_CHANGE, iDrumProgram, 0, 0));

// create MIDI events for each line
for (int line = 0; line < vDataSonify.size(); line++)
{
    // get line
    Vector vLine = (Vector)(vDataSonify.elementAt(line));

    // add MIDI events for this line
    for (int i = 0; i < vLine.size(); i++)
    {
        double dY = ((Float)(vLine.elementAt(i))).floatValue();

        if (Globals.rbLinearMapping.isSelected())
        {
            // LINEAR MAPPING
            iNote = (int)(((dY - dMinY) * (iMaxNote - iMinNote))
                / (dMaxY - dMinY));
            iNote += iMinNote;
        }
        else
        {
            // CHROMATIC SCALE MAPPING
            double exp = (dY - dMinY) / (dMaxY - dMinY);
            iNote = (int)(iMinNote * Math.pow(iMaxNote / iMinNote, exp));
        }

        iTime = lastLineEndTime + i * iInterval;

        // add NOTE ON event
        track.add(createMidiEvent(0, ShortMessage.NOTE_ON, iNote, 127, iTime));

        // add NOTE OFF event
        track.add(createMidiEvent(0, ShortMessage.NOTE_OFF,
            iNote, 127, iTime + iInterval));
    }

    // add EOL sound (on channel 1)
    if (Globals.chbEndLine.isSelected())
    {
        track.add(createMidiEvent(1, ShortMessage.NOTE_ON,
            60, 127, iTime + iInterval));
        track.add(createMidiEvent(1, ShortMessage.NOTE_OFF,
            60, 127, iTime + 2 * iInterval));
    }

    // add drum beats (on channel 2)
    if (Globals.chbDrumBeats.isSelected())
    {
        iLastDrum = -5000;

        for (int i = 1; i < vLine.size(); i++)
        {
            double dY = ((Float)(vLine.elementAt(i))).floatValue();
            double dY0 = ((Float)(vLine.elementAt(i-1))).floatValue();

            double slope = (dMinY == 0) ? (dY - dY0) : (dY - dY0) * dMinY;
            Globals.taStatus.append("slope = " + slope + "\n");
```

```
                    int drumInterval = (slope == 0.0)
                        ? 1500 : (int)Math.abs(150.0 / slope);
                    Globals.taStatus.append("drumInterval = " + drumInterval + "\n");

                    int iY0 = (i-1) * iInterval;
                    int iY = i * iInterval;

                    int iStart = iLastDrum + drumInterval;
                    if (iStart < iY0) iStart = iY0;

                    for (int j = iStart; j < iY; j+=drumInterval)
                    {
                        Globals.taStatus.append("drum added at " +
                            (lastLineEndTime + j) + "\n");
                        track.add(createMidiEvent(2, ShortMessage.NOTE_ON,
                            60, 127, lastLineEndTime + j));
                        track.add(createMidiEvent(2, ShortMessage.NOTE_OFF,
                            60, 127, lastLineEndTime + j + 250));
                        iLastDrum = j;
                    }
                }
            }

            // update lastLineEndTime
            lastLineEndTime = iTime + 3 * iInterval;
        }

        Globals.midiSequenceLength = Globals.midiSequence.getMicrosecondLength();
        Globals.taStatus.append("Sequence length in seconds: " +
            Globals.midiSequenceLength / (float)1e6 + "\n");
        Globals.taStatus.append("                     ticks:   " +
            Globals.midiSequence.getTickLength() + "\n\n");

        // add sequence to sequencer
        Globals.midiSequencer.setSequence(Globals.midiSequence);
    }
    catch (NumberFormatException ex)
    {
        System.out.println("\nERROR in createMidiSequence(): " + ex + "\n");
        System.exit(0);
    }
    catch (InvalidMidiDataException ex)
    {
        System.out.println("\nERROR in createMidiSequence(): " + ex + "\n");
        System.exit(0);
    }

    Globals.midiSequenceCreated = true;
}


public static void openMidiSequencer()
{
    try
    {
        Globals.taStatus.append("\nOpening MIDI sequencer...");
        Globals.midiSequencer.open();
        Globals.taStatus.append(" ok\n");
    }
```

```
        catch (MidiUnavailableException ex)
        {
            System.out.println("ERROR in openMidiSequencer(): " + ex + "\n");
            Globals.taStatus.append("\nERROR in openMidiSequencer(): " + ex + "\n\n");
            return;
        }

        // create timer for sequence-end tests
        Globals.timerSequence = new javax.swing.Timer(100, new SequenceEndListener());
    }


    public static void closeMidiSequencer()
    {
        Globals.taStatus.append("\nClosing MIDI sequencer...");
        Globals.midiSequencer.close();
        Globals.taStatus.append(" ok\n");
    }


    public static void play()
    {
        if (!Globals.midiSequenceCreated) createMidiSequence();

        Globals.timerSequence.start();
        Globals.midiSequencer.start();
    }


    public static void pause()
    {
        Globals.timerSequence.stop();
        if (Globals.midiSequencer.isOpen()) Globals.midiSequencer.stop();
    }


    public static void stop()
    {
        Globals.timerSequence.stop();
        if (Globals.midiSequencer.isOpen()) Globals.midiSequencer.stop();
        Globals.midiSequenceCreated = false;
    }


    public static void enableSonificationComponents(boolean bool)
    {
        Globals.cbProgram.setEnabled(bool);
        Globals.slTime.setEnabled(bool);
        Globals.lTime.setEnabled(bool);
        Globals.slMinNote.setEnabled(bool);
        Globals.lValMinNote.setEnabled(bool);
        Globals.slMaxNote.setEnabled(bool);
        Globals.lValMaxNote.setEnabled(bool);
        Globals.rbLinearMapping.setEnabled(bool);
        Globals.rbChromaticMapping.setEnabled(bool);
        Globals.cbSonificationType.setEnabled(bool);
        Globals.bOptions.setEnabled(bool);
    }
}
```

SliderListener.java

The goal of the `SliderListener` class is to supervise the position of the sliders and to update the corresponding textfields with the appropriate value. The `stateChanged` method is called whenever the position of a slider changes. Based on the source of the change, the corresponding label is updated.

```
import java.awt.event.*;
import javax.swing.JSlider;
import java.awt.*;
import javax.swing.*;
import javax.swing.event.*;


class SliderListener implements ChangeListener
{
    public void stateChanged(ChangeEvent e)
    {
        JSlider source = (JSlider)e.getSource();

        if (source == Globals.slTime)
        {
            int itmp3 = source.getValue();
            String Stmp3 = new String();
            Stmp3 = String.valueOf(itmp3);
            Globals.lTime.setText(Stmp3 + " ms");
        }
        else if (source == Globals.slMinNote)
        {
            int itmp4 = source.getValue();
            String Stmp4 = new String();
            Stmp4 = String.valueOf(itmp4);
            Globals.lValMinNote.setText(Stmp4);
        }
        else if (source == Globals.slMaxNote)
        {
            int itmp5 = source.getValue();
            String Stmp5 = new String();
            Stmp5 = String.valueOf(itmp5);
            Globals.lValMaxNote.setText(Stmp5);
        }
    }
}
```