TECHNISCHE
UNIVERSITÄT
WIEN

VIENNA
UNIVERSITY OF
TECHNOLOGY

MAGISTERARBEIT

Interactive Visualization of Expressive Piano Performance

Ausgeführt am Institut für

Computational Perception

der Johannes-Kepler-Universität Linz

unter der Anleitung von

Univ-Prof. Dipl.-Ing. Dr. Gerhard Widmer

durch

Martin Gasser

Klosterneuburgerstraße 66/30
1200 Wien

Wien, Mai 2005

_____
Unterschrift

Abstract

Computer supported performance analysis aims at musicians and musicologists investigating performance-related aspects of music. Since a key component of every supportive computer environment is the user interface, presentational and interactive aspects are of fundamental concern for the design and the implementation of a computer supported performance analysis system.

In this thesis, an overview of music performance analysis, and of existing visual representations of musical performance is given. Moreover, well-known methods from Information Visualization (InfoVis) and User Interface Design (UID) are reviewed and its applications to the present problem space are discussed.

Finally, a prototypical implementation of an extensible framework for visualization of expressive piano performance is reviewed, and a typical application scenario is described in a step-by-step fashion.

Kurzfassung

Computerunterstützte Performanceanalyse soll Musikern und Forschern bei der Untersuchung von aufführungspraktischen Aspekten von Musik helfen. Da die Benutzerschnittstelle die Schlüsselkomponente von Computersystemen, welche in komplexe menschliche Arbeitsabläufe eingebunden sind - und diese teilweise übernehmnen sollen - ist, sind visuelle und interaktive Aspekte von fundamentaler Bedeutung für das Design und die Implementierung eines solchen Systems.

Diese Arbeit gibt einen Überblick über zentrale Aspekte der Performance-Forschung, wobei der Schwerpunkt auf visuelle Repräsentationen von musikalischen Aufführungen gelegt wird.

Methoden aus den Forschungsbereichen der Informationsvisualisierung (Info-Vis) und des User Interface Design (UID) werden betrachtet, ihre Anwendbarkeit in der computerunterstützten Performanceanalyse wird diskutiert.

Schließlich wird ein Prototyp eines erweiterbaren Frameworks für die interaktive Visualisierung von ausdrucksvollen Klavierinterpretationen beschrieben, und ein konkreter Anwendungsfall wird Schritt für Schritt durchgespielt.

Acknowledgements

I want to thank the following people, who contributed to the success of this thesis in different ways:

First of all, I want to express my gratitude to my supervisor Prof. Gerhard Widmer for his support throughout the preparation, the implementation, and the documentation phase of this project. I also want to thank Werner Goebl and Simon Dixon for providing me with valuable information about music performance research.

I am deeply grateful to my parents Annemarie and Karl Gasser for their strong support throughout my studies. Special thanks go to my girlfriend Petra Reinwald, for tolerating all my ups and downs during the last months.

Last but not least, I want to thank my former colleagues and mentors at the VRVis Research Center Vienna - especially Helwig Hauser and Helmut Doleisch - for infecting me with the addiction to computer graphics and visualization, which was and will be an inspiring source of inspiration for my work.

List of Figures

Contents

viii

Chapter 1

Introduction

> Take a music bath once or twice a week for a few seasons, and you
> will find that it is to the soul what the water bath is to the body.
>
> *Oliver Wendell Holmes (1809 - 1894)*

Music performance can be regarded as the interpretation, structuring, and physical realization of composed music. Music performance research - which tries to identify and analyze more or less subtle variations between different realizations of composed pieces - has emerged as a major research area in musicology.

Computer supported performance research provides tools for musicians and musicologists, who want to investigate performance-related aspects of music. Since a key component of every supportive computer environment is the user interface, presentational and interactive aspects are of fundamental concern for the design and the development of such a system.

This thesis describes the requirement analysis, the design, and the implementation of an interactive performance analysis tool. We explicitly restrict ourselves to classical piano music, i.e. music that has been composed during the artistic eras of *classicism* and *romanticism*. We further assume symbolic input data in the form of MIDI [13] data, which has been aligned to the score notation in a preprocessing step. The data acquisition process involves a pianist playing on a Bösendorfer SE 290 computer-controlled grand piano [7], which records performances in a proprietary data format. The collected performance data is then fed to a score-performance matching module, that aligns the performance data to the corresponding score information on a note-by-note basis. The matched data is then fed to the analysis system, which is subject to this thesis. See figure 1.1 for an overview of this workflow.
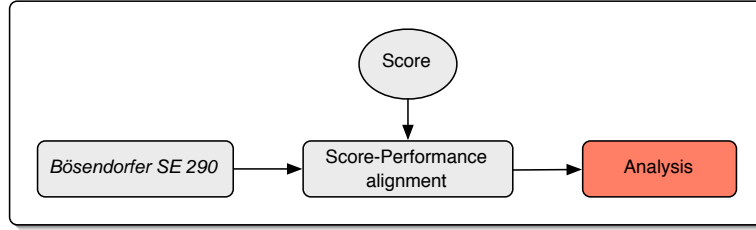
Figure 1.1: Data acquisition, preprocessing, and analysis

The key demands on the user interface of the performance analysis tool that has been designed in the course of this thesis are *usability*, *comprehensibility*, and *interactivity*. Thus, well-known techniques from Information Visualization (InfoVis), and User Interface Design (UID) are reviewed and its application in a performance analysis framework is discussed. Finally, the functionality of the system is demonstrated in an interactive analysis session, comparing two performances of Chopins Etude Op. 10, No. 3, and some conclusions and a brief outlook on future work is given.

Chapter 2

Music performance research

Those who cannot remember the past are condemned to repeat it.

George Santayana (1863 - 1952)

Music performance research aims at the identification of general rules, that characterize the artistic style of specific performers. Performance research can be classified into two major approches. While the *analysis-by-measurement* approach is based on the analysis of diversities and commonalities in existing performances, the *analysis-by-synthesis* approach modifies performance parameters in a synthetic performance, until a setting is found that optimally models the audible result. Interactive visualization methods can be used to extend an *analysis-by-measurement* approach, by providing the user with meaningful visual representations of the measured data, thus alleviating the process of finding reasonable results.

This thesis will focus on the *expressive* aspects of music performance. *Expressive performance* is characterized by the deliberate shaping of parameters like tempo, dynamics, timing microstructure, and articulation, by a performer. Bresin [22] calls those parameters, that make up the expressive structure of a performance, *expressive cues*.

In the following, a short historical overview is given, and four examples of computational models of expressive music performance are mentioned.

2.1 History

In the late 19th century, A. Binet and J. Courtier (1895) and K. Ebhardt (1898) started to measure musical performances by recording the key depressions of pianos [30]. In the course of their investigations, they observed that it is possible to clearly

discriminate amateurs from professional pianists by comparing the performances with respect to trills, scales, accents and crescendo-decrescendo patterns. Sears (1902) analyzed key movements on a reed organ, Heinlein (1929, 1930) studied pedal usage of famous pianists performing Schumann's *Träumerei* on a Duo-Art reproducing piano [30].

Hartmann (1932) carried out a thorough analysis of two pianists' performances of the first movement of Beethoven's *Moonlight Sonata*. By measuring note lengths on paper roll recordings for player pianos, he found considerable variation of tempo, articulation and chord asynchrony specific to the composers.

At the University of Iowa, a large group of researchers headed by C.E. Seashore [47] studied expressive piano performance by filming the movements of the hammers. Henderson [37] analyzed two performances of Chopin's *Nocturne*. In measures containing three quarter notes, the second was relatively shortened, whereas measures consisting of a half note plus a quarter, the latter was relatively lengthened. It was hypothesized that the lengthening was made in order to achieve more melodic equality between the long and the short notes.

Phrasing was another subject of Henderson's investigations. A general trend common to both performances was an acceleration followed by a ritard an a decrescendo towards the end of each phrase. Besides those commonalities many differences became visible; the pianists varied considerably in the tempo chosen for the performance. Chords were played asynchronously in both performances, showing *melody lead* (the melody was played before all other notes) in one case, whereas in the other case the bass notes where played first (presumably to emphasize its countermelodic character).

2.2 Contemporary research

Recently, several computational models of expressive performance have been proposed [58]. In the following, four representatives are briefly mentioned.

2.2.1 The KTH model

This model is based on a set of context-sensitive performance rules, that describe aspects of timing, dynamics, and tempo. The model follows the *analysis-by-synthesis* principle, where a professional musician evaluates tentatively parameterized rules brought forward by the researcher. Researcher and musician are in

a constant feedback loop, trying to find the best parameter setting for each rule [51].

2.2.2 The Todd model

Whereas the KTH model is a typical representative of the *analysis-by-synthesis* approach, Todds model [53] obtains its empirical evidence directly from human performances. This model is based on the assumption that there is a direct relation between musical structure and the performance, which can be modeled by a one simple rule. It is hypothesized by Todd that the hierarchical grouping structure of music identifies various points of stability in the music, and that a performer slows down at those points of stability in order to enable the listener to perceive the hierarchical structure of the music. Apart from the timing structure, Todd's model also incorporates dynamics, by directly relating the played tempo to the perceived intensity. Hammer velocity has been modeled as approximately proportional to the squared tempo.

2.2.3 The Mazzola model

A model that is based on mathematical considerations is the Mathematical Music Theory by Guerino Mazzola [42]. It covers analytical as well as performance aspects, and it has been implemented in the Rubato system. This system consists of several modules, the so-called Rubettes. One of them, the EspressoRUBETTE, implements the *inverse performance theory* (a subset of Mazzola's theory), and analyses performance data by matching the input data to scores and extracting expressive parameters.

2.2.4 The machine learning model

An alternative way of building rules for expressive performance is to start from large collections of measured performance data and to use inductive rule learning methods to discover significant regularities in the data. Widmer et al. [57] have used note-level as well as higher-level models to generate performance rules that can be used to predict both note-level expressive deviations as well as patterns at the phrase level.

It should be stated, that computational models try to model the *commonalities* between performances, whereas this thesis focuses on dissimilarities, i.e, the per-

sonal artistic style of specific performers. More precisely, methods for interactive audiovisual exploration of performance data are investigated.

2.3   Data acquisition and preprocessing

The first serious step in performance analysis is the acquisition of data to be analyzed. Performance data can be measured at various levels: At the low signal level, the symbolic level (by using MIDI), or even by incorporating information that has no direct association to the performed music, e.g., by interpreting visual gestures of performers. This thesis primarily deals with symbolic note-level data that has been acquired with a computer-monitored Bösendorfer grand piano and converted to MIDI data. However, for completeness, the following section also reviews or mentions methods that deal with audio data.

Data acquisition only analyzes parameters like pitch and dynamics. For performance analysis, it is of the utmost importance that note-level semantics with respect to the score information are known at analysis time. Thus, acquired data has to be aligned with score information in a preprocessing step.

2.3.1   Score-Performance matching

A fundamental task in performance research is to find the note-level connections between performances and the corresponding scores. In theory, it is possible to construct a perfect one-to-one mapping from scores to performances.

However, due to human imperfection, this is not feasible. There are three main reasons for this:

- Musicians leave out notes by error

- Musicians add notes by error

- Scores contain cues that offer performers a limited degree of freedom (i.e. glissandi, trills and grace notes)

In order to analyze performances, it is necessary to classify performed notes with respect to the above mentioned categories.

Several approaches to the solution of the score-performance matching problem exist. When the input is symbolic (e.g., MIDI messages) the problem is considerably simplified, since onset, offset, and pitch are known, whereas audio information

needs a preprocessing step that transcribes the audio signal into a symbolic data stream by applying various DSP methods. Dannenberg [19] has developed a score following algorithm that aligns MIDI streams and scores in realtime. It is based on a dynamic programming algorithm (dynamic time warping) that finds an optimal path through a similarity matrix built from the score and performance in an incremental fashion. At IRCAM, the Suivi [46] MIDI score following system has been developed, which models the performance with Hidden Markov Models (a specialization of Bayesian networks).

Beat tracking is another approach, that tries to map the real (performance) time to score time. However, this method does not provide for note-level information. The *Beat Root* beat tracking system [26] is built on a robust multi-agent algorithm that identifies beat information from peaks in the RMS value of the audio signal.

Remarkable progress has been done in these fields, however, the problem of automatic score-performance matching has not been completely solved yet, since errors of the piano player can failure of the matching. The data that has been used in this thesis has been produced in the course of a large project headed by Gerhard Widmer, data from a Bösendorfer SE 290 grand piano has been automatically matched to score files and manually corrected afterwards.

2.4 Feature extraction from audio data

In this section DSP-related methods that directly process audio data are shortly reviewed.

2.4.1 DSP methods

The spectrum of a stationary signal can be calculated with well-known tools from mathematics and digital signal processing. Continuous signals can be transformed into fourier series, its discrete counterpart, the discrete fourier transform, can be used to represent sampled signals in the frequency domain. The fourier coefficients represent the spectral content of the signal (in fact, both fourier series and the DFT are discrete in the frequency domain, while in the time domain, the series expansion yields a continuous function while the DFT is only defined for discrete time values).

Since real sampled signals are not stationary, the DFT - which cannot localize features in time - will not yield accurate results. A combination of time-based and

frequency-based is the STFT (*Short Time Fourier Transform* [50]). However, the STFT, which is a windowed version of the discrete fourier transform, suffers from time-frequency resolution problems related to the window width, which is equal for all frequency bands, and causes either blurring in the frequency or in the time domain. An alternative that tries to overcome those drawbacks is the discrete wavelet transform [55], which has been proven useful for various purposes, such as beat tracking or speech recognition [43, 54].

To compute perceived *loudness* accurately, it is inevitable to analyze the audio signal with methods from signal processing. Various models exist that emulate the physiology of the human ear (see [59]).

DSP based methods can serve as a foundation to other methods, that derive higher-level, performance-related information. Dixon [27] has shown the applicability of the STFT for an audio transcription system. If used in an visualization environment, the STFT can be produce a two dimensional plot, which is called the *spectrogram*. From a spectrogram, *pitch* can be visually estimated by determining the predominant frequencies (the peaks in the spectrum).

2.5 Feature extraction from MIDI data

The expressive dimensions that are going to be investigated in this thesis are *timing*, *tempo*, *dynamics*, *articulation*, and *pedal usage*.

Several approaches related to the extraction of expressive parameters from symbolic performance data are reviewed. These features are classified into *first-order* and *second-order* data. First order data do not need further processing (i.e., they are already present in the input data), while second-order data are calculated from first-order data according to pre-defined rules.

2.5.1 First-order data

Dynamics

Dynamics is generally referred to as "loudness". Since it is not possible to calculate loudness values from MIDI data, *MIDI velocity* (see [13]) is taken as the closest possible estimate of dynamics in the course of this work.

Note onset/offset

Note onset/offset time specifies the time when a played note starts/stops. Onset and offset are given in units of MIDI ticks, and can be converted to microseconds according to the configuration of the MIDI sequencing device in use.

Pedal

Pedal values are stored in MIDI data as controller values and describe the pedal pressure at certain points in time. Pedal is used by performers to extend the duration of played notes and to control dynamics.

2.5.2 Second-order data

Note duration

Note duration designates the time interval between note onset and note offset.

Beat&tempo

Beat can be described as a perceived pulse of beat times which are approximately equally spaced in time. Played notes are aligned with the resulting *beat track*. Beat tracks can be perceived (and therefore computed) at different *metrical levels*. The default metrical level corresponds to the denominator of the time signature of the musical piece under consideration (i.e., in $\frac{3}{4}$ time, a beat corresponds to a quarter note, while in $\frac{6}{8}$ time, a beat corresponds to an eighth note).

Local tempo is reciprocally related to the time interval between two beats. An important concept that specifies the way local tempo can be calculated, is the principle of causality. That is, the *exact* calculation of local tempo must take place *after* the occurence of the corresponding beat. This has important consequences for real time applications, where tempo must be calculated instantaneously. Real time applications cannot rely on non-causal decisions because that would require knowledge about the future. An area of ongoing research is the question how humans estimate future beat times (e.g., in a live performance of improvised music) and how a priori knowledge about the rhythmical structure of a musical piece (e.g., in live performances of classical music) can be represented. Several researchers have attacked the problem of causal beat tracking, for instance by proposing agent-based architectures [34, 33, 26], or bayesian methods [24].
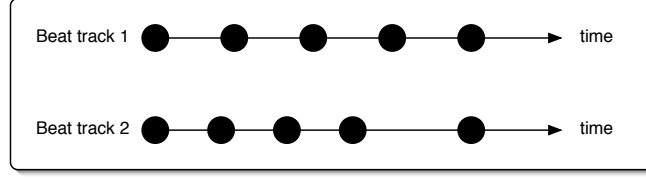
Figure 2.1: Two beat tracks

A different approach relies on a priori knowledge of the complete score-performance matching (see section 2.3.1). Such methods generally can be classified as being non-causal, since for every point in time, the past and the future is known. In this thesis, a non-causal tempo calculation model has been implemented, because the main focus is on the analysis of performances (which implies the presence of a complete performance).

In figure 2.1, two beat tracks are shown; while in the first track, the beats are spaced regularly in time, the second track exposes a slight accelerando during the first four beats.

An *inter-onset interval* (IOI) is the time interval between the onsets of two notes.

$$IOI_p(N_x, N_y) = onset_p(N_y) - onset_p(N_x), y > x \qquad (2.1)$$

$$IOI_s(N_x, N_y) = onset_s(N_y) - onset_s(N_x), y > x \qquad (2.2)$$

where the functions $onset_p(N_x)$ and $onset_s(N_x)$ return the played onset time and the score onset time, respectively, of a given note N_x. Score time is in *beats*, while played time is assumed to be in *microseconds*.

An *inter-beat interval* (IBI) refers to the inter-onset interval of two *beats*.

$$IBI_{p,m}(x, y) = IOI_p(B_{x,m}, B_{y,m}) \qquad (2.3)$$

$$IBI_{s,m}(x, y) = IOI_s(B_{x,m}, B_{y,m}) \qquad (2.4)$$

In this formalization, a beat is a specialization of a note, with the restriction that its onset time is a multiple of the chosen metrical level.

Local tempo (in beats per minute) can be calculated as

$$tempo(B_{x,m}) = 60000000.0 \times \frac{IBI_{s,m}(x, x+1)}{IBI_{p,m}(x, x+1)} \tag{2.5}$$

where B_1 and B_2 are two successive beats at the desired metrical level.

In case that a score note corresponding to a beat time does not exist (e.g., in the case of rests), played beat times are linearly interpolated from neighboring beats.

$$onset_p(B_{x,m}) = onset_p(B_{x-1,m}) + IBI_{p,m}(x-1, x+1) \times \frac{IBI_{s,m}(x-1, x)}{IBI_{s,m}(x-1, x+1)} \tag{2.6}$$

Articulation

Articulation is modeled as the amount of overlap of two successively played notes, normalized by the theoretical duration of the first note when played in the current local tempo.

$$overlap(N_x) = offset_p(N_x) - onset_p(N_{x+1}) \tag{2.7}$$

$$articulation(N_x) = \frac{overlap(N_x)}{duration(N_1) \times \frac{60000000.0}{tempo(B_{x,m})}} \tag{2.8}$$

In this model, a note is played in *legato*, if its articulation value is greater than 0. If it is smaller than 0, it is assumed to be *staccato*. As human performances always bear a certain amount of imprecision, an articulation value of 0 is extremely unlikely. It should be explicitly mentioned, that this model is only a very rough approximation to articulation, as it is defined in musicology.

Chapter 3

Performance visualization

> One picture is worth a thousand words.
>
> *Fred R. Barnard*

3.1 Related work

Recently, music visualization has become an attractive field of research. While the term *music visualization* has not been given a strict definition, this thesis will focus on *performance visualization*, i.e. the visualization of expressive parameters that shape the audible appearance of a performance. Schoonderwaldt et al. [45] have explored the usability of visualization methods in an interactive feedback learning system aimed at improving the expressive skills of performers. Hiraga et al. [38, 39, 44] have proposed an interactive performance visualization system, which allows the user to explore performance data in a 3D visualization system. Dixon et al. [25] have used interactive visualization of expressive performances in tempo/dynamics-space to discover commonalities and dissimilarities in performances.

3.2 Direct visualization of sampled audio data

Audio data can be visualized in the time domain and/or the frequency domain. The time domain representations (waveform views) of audio signals give clues about the amplitude structure of the signal. However, to draw conclusions from such a visualization, the analyzed signal has to be strictly band-limited and monophonic, otherwise spectral masking effects make it very difficult to identify meaningful patterns.
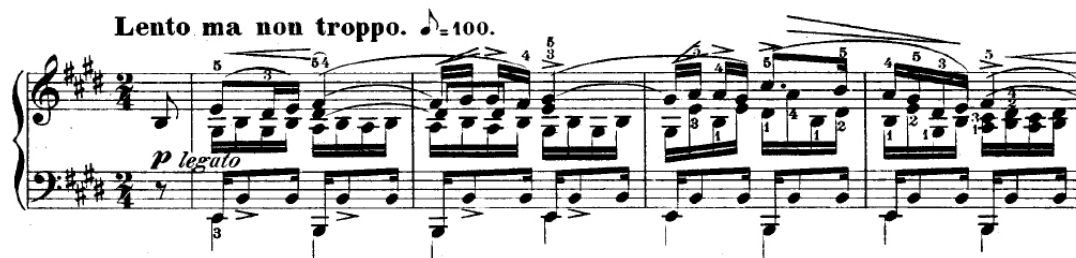
Figure 3.1: A score of the beginning of a Chopin etude

Frequency domain visualizations are based on a signal transform and show the spectral energy of the signal. A combination of time-based and frequency-based is the STFT (*Short Time Fourier Transform* [50]) that can be visualized in a *spectrogram*, that shows the amount of spectral energy in the sub-bands of the audio data.

3.3 Visualization of symbolic data

Since this thesis focuses on piano music that has been recorded on a computer monitored grand piano, symbolic information consists of note onset, note offset, pitch, and MIDI velocity (which is related to the loudness of the played note).

3.3.1 Score

In classical western music, the score representation has become the most widespread means for communicating musical intentions of composers. Therefore, a natural choice for music visualization would be a score. A drawback of the score representation is that it isn't well suited for the visualization of musical performance, since its main intention is to transport the composer's idea, rather than the performer's interpretation (see figure 3.1).

3.3.2 PianoRoll

A visualization metaphor that has gained widespread application is the PianoRoll representation. A PianoRoll is basically a two-dimensional chart, where the x axis denotes time, and the y axis pitch. Key depressions are represented as bars with constant constant height and variable width (according to the duration of
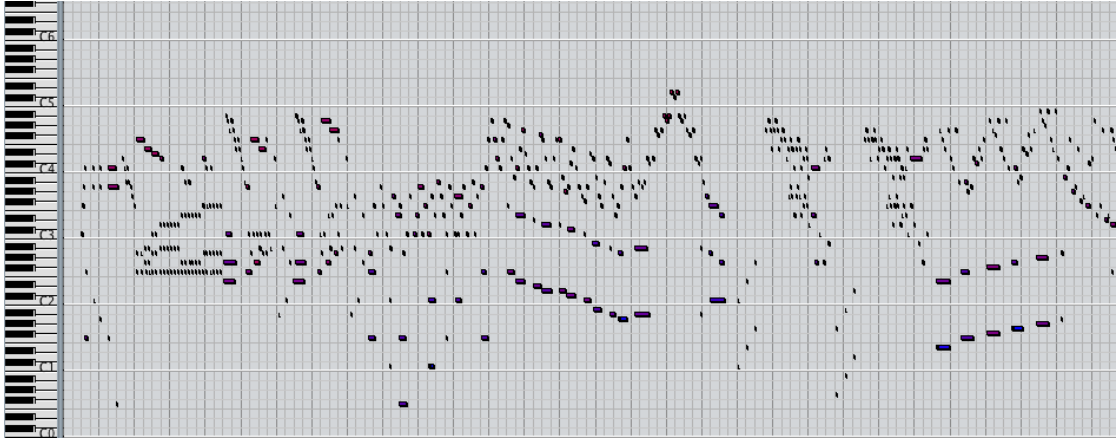
Figure 3.2: A PianoRoll in Cubase ^{TM}
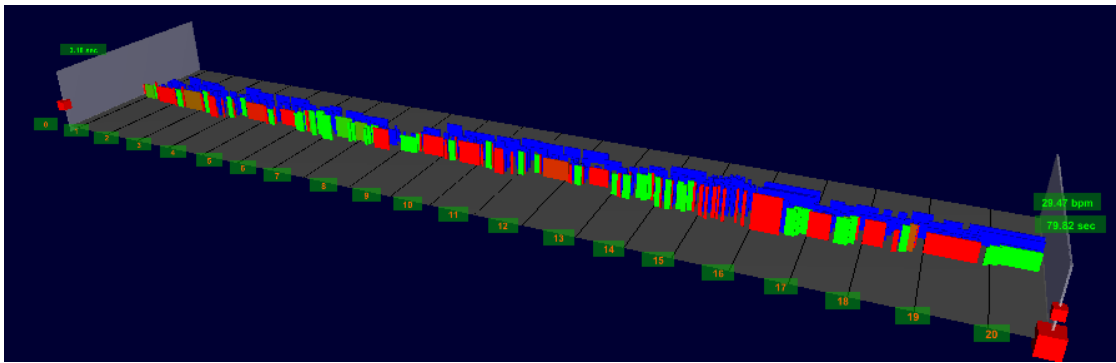
Figure 3.3: A 3D PianoRoll

the key press). As mentioned in the historical overview (see section 2.1), its origin are player pianos, where paper rolls where used to record key depressions of a piano player, in order to replay performances at a later point in time. Ingenious researchers very soon discovered that those paper rolls - the *PianoRolls* - can be used to analyze piano performances very precisely.

PianoRolls are a powerful visualization metaphor. Apart from the onset, offset, and pitch information, articulation - which is related to the amount of overlap of two successive notes - is clearly visible in PianoRolls. Figure 3.2 shows a PianoRoll representation in the commercial software package Cubase [11].

An extension based on the PianoRoll representation are 3D PianoRolls (see 3.3), which have been developed in the course of this thesis. In a right-handed coordinate system, time is denoted by the x-axis, pitch by the z-axis, and a third data dimension, commonly loudness or a related measure, by the y axis.

14

Figure 3.4: Tempo diagrams at the bar/beat/halfbeat/quarterbeat metrical levels
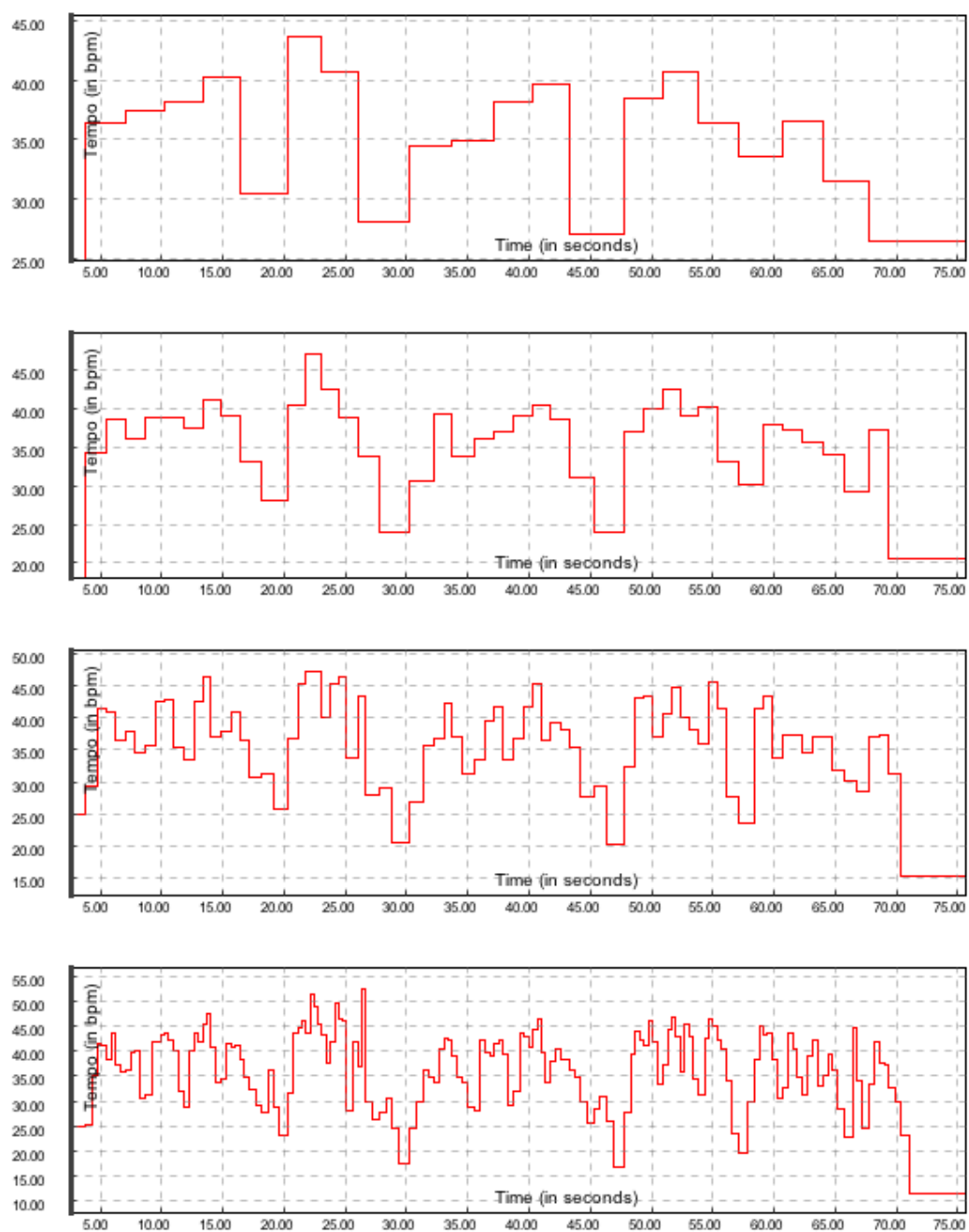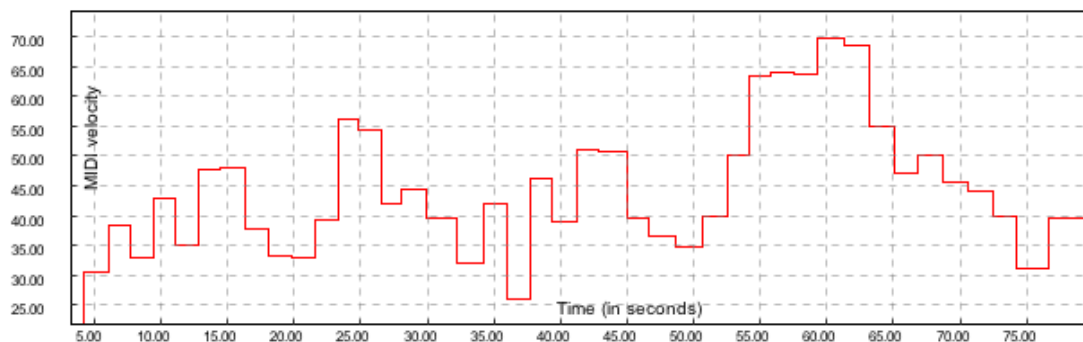
15

Figure 3.5: A velocity curve at the beat level

3.3.3 Tempo diagram

Apart from the visualization of first-order features (note onset, note offset, MIDI pitch, MIDI velocity), second-order features (tempo, articulation) also can be directly visualized. Multi-level tempo diagrams - which have been implemented in this thesis - (see figure 3.4) show the local tempo at various metrical levels (bar, beat, sub-beat). The smallest rhythmical unit in a musical piece determines the bottom level of the piece. Starting from this level, the local tempo is calculated at successively larger time windows, until the bar level is reached. An extension to higher structural levels (phrase-level) has been considered, but not been implemented, due to lack of structural information above the bar level.

Formally, a tempo diagram can be defined as

$$tempogram(B_x) = tempo(B_{x,m}, B_{x+1,m}) \tag{3.1}$$

for all possible values of the metrical level m.

3.3.4 Dynamics diagram

Similarly to multi-level tempo diagrams, a hierarchical visualization of dynamics can be constructed (see figure 3.5). When working with MIDI data, loudness can be only estimated from velocity values. This is not very accurate, since hammer velocity and perceived loudness are not linearly related. The current approach models dynamics by summing up MIDI velocities of all played notes in a given time window. As in the tempo diagrams, the time windows are aligned with a given metrical level.

16
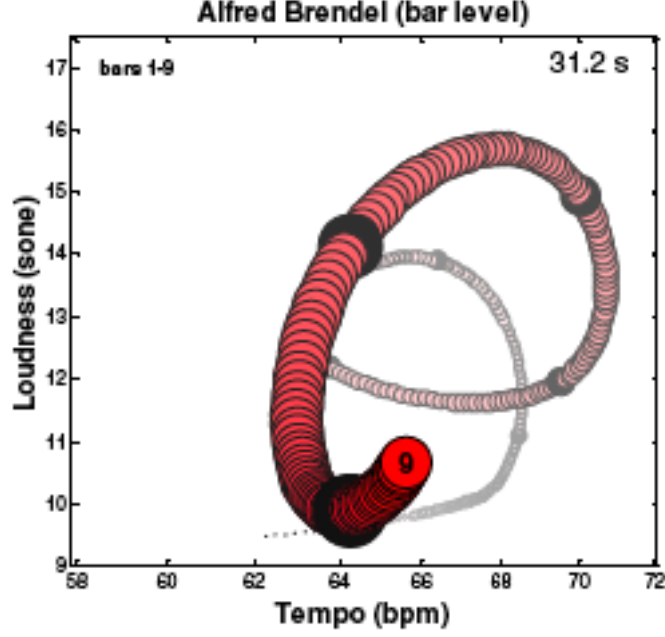
Figure 3.6: The performance worm

Formally, the dynamics measure is calculated as

$$dynamics(B_{x,m}) = \frac{\sum_{id_p(B_{x,m})}^{id_p(B_{x+1,m})} velocity(N_y)}{IBI_{s,m}(x, x+1)} \tag{3.2}$$

where the function $id(B_{x,m})$ returns a position in the performance that corresponds to the beat time $B_{x,m}$.

3.3.5 Performance worm

The *performance worm* is a metaphor that has been developed by Langner [40] and implemented as a real time animation by Dixon et al. [25]. It represents tempo and dynamics curves at the beat-level in an integrated way as trajectories in two-dimensional tempo-loudness space. Tempo and dynamics points are convolved with a gaussian kernel, and the resulting points are interpolated to form a smooth curve (see figure 3.6). The width of the gaussian kernel determines the degree of variation that is visible in the resulting trajectory.

In this thesis, some of the here presented visual representations (PianoRoll, PerformanceWork, tempo/dynamics curves) will be extended with special emphasis on interactivity. Moreover, the individual visual representations will be integrated in an environment that links them with respect to their common parameters, which will enable the simultaneous analysis of performances in several visual representations.

Chapter 4

Requirement analysis

> To accomplish great things, we must dream as well as act.
>
> *Anatole France (1844 - 1924)*

This chapter describes the main requirements that were posed to the system that has been designed.

4.1   Early considerations

The first fundamental question that arose was whether the system should be capable of real-time operation. An obvious realization was that such a thing as "real time analysis" is rather unrealistic. Since *analysis* is the emphasis of this work, the idea of an online visualization system has been abandoned in favor of an offline system with more sophisticated analysis capabilities.

Another presumption of the author is, that a real-time system would have biased the choice of the implementation environment (from Java towards C++) in the first place, because an efficient multi-threaded message passing infrastructure would have been essential to ensure reliable operation of such a system.

4.2   Functional requirements

In this section, the main functional requirements for an interactive performance visualization and analysis system are reviewed.

4.2.1 Data processing

Data input

As input to the system, data files that store the score-performance matching information, should be accepted. Data files in our own *match file* format, which has been designed in the course of a large project headed by Gerhard Widmer, store score data, performance data, and score-performance matching information. Thus, the association between score notes and corresponding performed notes is known.

In addition to match files, standard MIDI [13] files have to be loaded, since match files don't provide information about pedal controller values.

Data derivation

The expressive cues (tempo, articulation, dynamics), as described in 2.5, should be computed and provided in a data structure for further usage in the application.

4.2.2 User interaction/visualization

In the following, the UI elements that should be implemented are defined.

A central controlling instrument

Similar to common MIDI sequencing applications, the system should offer an interface that allows the user to

- playback the loaded performance

- change the current position in time

- modify the lower and upper bounds of the time range under investigation

- alter the overall MIDI tempo

The PianoRoll visualization

The system should provide a 3D PianoRoll view, that acts as a navigational element as well as a visualization of MIDI velocity, tempo, and articulation in the

time/pitch/dynamics space. User interaction should occur directly in the 3D visualization via common drag&drop metaphors. The user interface should provide basic operations, like translation, rotation and zooming.

Apart from the actual visualization of the played notes, primitives that designate the current position in time, as well as the lower and upper bounds of the current time range *in focus*, should be provided.

Changes of the current time position that emanate from the user should be propagated to the other views that show the same performance, but with respect to different expressive parameters.

The PerformanceWorm visualization

As an extension to the PerformanceWorm, as described in section 3.3.5, a 3D PerformanceWorm view should be integrated into the system. The basic user interface of the worm view is modeled after the piano roll view, as described in the last subsection, but the worm is embedded into time/tempo/dynamics space. The primitive used in the worm visualization should be modeled as a continuous space curve, in contradiction to Dixon's implementation, where the worm is a series of points in tempo/dynamics space. Additionally to the visualization of tempo and dynamics (with respect to time), melody articulation should be shown in this visualization by modulating the shape of the worm according to smoothed articulation values.

Tempo and dynamics visualization

Tempo and dynamics should be visualized with the methods described in 3 (tempo diagram and dynamics diagram). Additionally to the visualization of the performance parameter at a given point in time, the lower and upper time range bounds should be visualized in the diagrams by employing the *fisheye view* [29] concept. Fisheye views apply a non-linear scaling function, that enlarges the area of interest, to the visualized data.

The metrical level for the computation of performance parameters should be adjustable by subdividing the bar level. For consistency, the following time subdivision rule has been defined:

$$\text{Subdivide according to} \begin{cases} PFD(ML) & \text{if } (Beat \leq ML \leq Bar) \\ BS(ML) & \text{if } (ML < Beat) \end{cases} \tag{4.1}$$

where $PFD(ML)$ refers to the prime factor decomposition and $BS(ML)$ refers to the binary subdivision of the metrical level ML.

Thus, for example, in a $\frac{3}{4}$ time signature, the possible metrical levels would be $\frac{3}{4}, \frac{1}{4}, \frac{1}{8}, \frac{1}{16}$, in $\frac{6}{8}$ time signature, it would be $\frac{6}{8}, \frac{3}{8}, \frac{1}{8}, \frac{1}{16}$.

Pedal visualization

Pedal visualization should be based on a user interface similar to the on of the tempo/dynamics diagrams. The two pedal types used in expressive piano performance are *sustain pedal*, which elongates a played note to the duration of the pedal pressure, and the *damper pedal*, which dampens played notes according to the pressure of the pedal. It should be noted that usually MIDI synthesizers don't reproduce the pedal characteristics of real pianos (in fact, many synthesizers use a cutoff value of 64 to divide pedal controller values (which can take values between 0 and 127) into a **On** (if **value** $>= 64$) and **Off** values (if **value** < 64). Thus, the visualization that is provided does not actually correspond to the audible MIDI reproduction of the performance.

Like the tempo/velocity diagrams, the pedal visualization should support the *fisheye* concept, which supports the investigation of the microstructure of pedal usage.

4.2.3   Linking

Linking of multiple views

Until now, several requirements to the individual views that should be offered to the user have been defined. However, the key hypothesis of this thesis is that it is possible to alleviate the work of researchers by linking those views with respect to certain parameters. In the current implementation, this concept should be demonstrated by linking the views via their most common parameter, which is *time*. In such a setting, all views that are associated with a specific performance show the same point in time, but with respect to the performance parameters that are visualized in the individual views.

Additionally to the linking at the *performance level* (linking via performance/real time), linking at the *score* level should be supported. That is, the system should support the analysis of several performances of the same musical piece.

4.3   Non-functional requirements

4.3.1   Portability

A key assertion was that the implemented system can be run on the three major platforms (Linux, MacOS X, MS Windows).

4.3.2   Usability

The system should provide an intuitive user interface. Its functionality should be modeled after workflows that are well-known to musicologists.

4.3.3   Performance

The system should perform seamlessly on common PC hardware (since laptop computers are a target environment for the application, the performance bounds are even tighter).

Chapter 5

Design and implementation

> Imagination is the beginning of creation. You imagine what you
> desire, you will what you imagine and at last you create what you
> will.
>
> *George Bernard Shaw (1856 - 1950)*

5.1 Choice of the implementation environment

Initially, a combination of C++, OpenGL [9], the QT GUI toolkit [14], and PortAudio [10] for MIDI I/O has been the favored environment. However, due to certain circumstances, Java has been chosen. In particular, the author planned to reuse existing code, and the application had to be cross-platform without recompilation.

At development time, Java 1.4.2 was the version of the Java Development Environment that was available for the target platforms (Linux, MacOS X, Windows XP), and was therefore chosen for the implementation of the application. As a critical remark, the author would like to state that Java is definitely cross-plattform in the sense that applications *run* on all platforms; however, Java doesn't *perform* equally well on all platforms. Especially on the MacOS X platform, Java suffers from severe performance problems - unfortunately, the author is a passionate Apple user. However, the situation is likely to improve in the future.

5.1.1 Hardware accelerated rendering in Java

Since a pure software rendering environment would not support real-time rendering of arbitrarily complex scenes, access to the native rendering capabilities of the

graphics hardware was necessary. Java3D [4] provides hardware accelerated rendering of scenegraphs (see section 5.1.2), however, due to the internal thread-safe design of Java3D, rendering speed is not always satisfying. In particular, Java3D allows several threads to modify the scenegraph structure. The library synchronizes updates to to the scenegraph and rendering, which results in perceptible performance overhead.

In theory, it would have been possible to access an OpenGL context via the Java Native Interface [5], and perform the core rendering in native code, but that would have introduced considerable implementation overhead. Luckily, Sun now officially supports a Java interface to OpenGL (JOGL, [6]), which enables direct access to hardware accelerated rendering from Java. Nevertheless, since OpenGL is a hierarchical, state-oriented API, it does not fit well into the object-oriented design of Java. Thus, on top of JOGL, several OpenSource projects which implement a scenegraph interface similar to Java3D, have been introduced (Xith3D [16], Aviatrix3D [12]). While providing the object-oriented advantages of Java3D, thread-safety has been abandoned in favor of rendering speed. Thus, the developer has to synchronize scenegraph updates and scenegraph traversal.

For the current implementation, the scenegraph library Xith3D has been chosen. Although its developers follow a rather game-oriented approach (they put strong emphasis on technologies that are heavily used by game developers, like collision detection, level loaders, etc.), it is currently the most powerful Open-Source Java scenegraph library.

5.1.2   Scenegraph basics

A scenegraph is a directed acyclic graph that organizes objects in a three-dimensional universe in a hierarchical fashion. Figure 5.1 shows a simple scenegraph.

Each scene graph has a single VirtualUniverse. The VirtualUniverse object has a list of Locale objects. A Locale object provides a reference point in the virtual universe, being a landmark used to determine the location of visual objects in the virtual universe.

A BranchGroup (the **BG** nodes) object is the root of a subgraph. There are two different categories of scene subgraphs: the view branch graph and the content branch graph. The content branch graph (the left one in the figure) specifies the contents of the virtual universe - geometry, appearance, behavior, location, sound, and lights. The view branch graph specifies the viewing parameters such as the
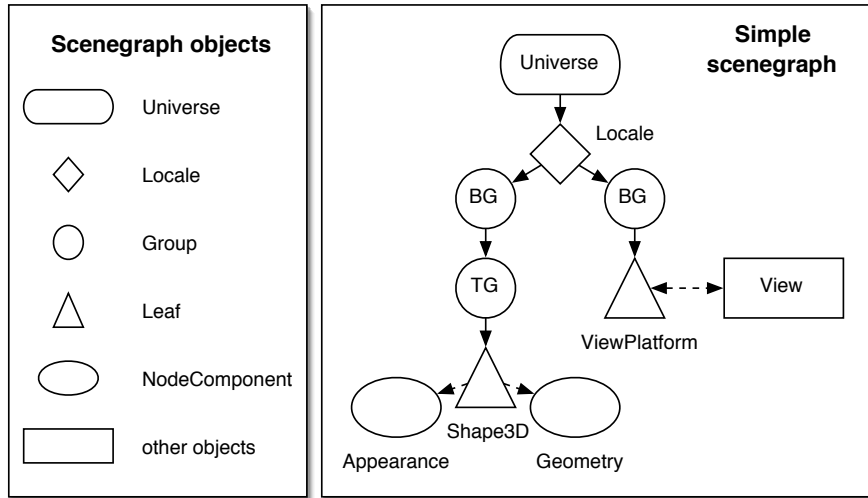
Figure 5.1: A simple scenegraph

viewing location and direction. Together, the two branches specify much of the work the renderer has to do.

A TransformGroup (**TG** nodes) specifies geometric transformations to be applied to all sub-objects. Basically, a TransformGroup encapsulates a transformation matrix, that is multiplied onto the internal OpenGL stack during traversal of the scenegraph.

Leaf nodes specify the actual content of the universe. Shape3D nodes are specific leaf nodes that have a certain appearance (material properties, texture) and geometry data.

5.1.3 JavaSound

JavaSound is Java's abstraction layer to audio and MIDI capabilities of the operating system. Since the JavaSound implementation in Java 1.4.2 is rather buggy and provided very unstable MIDI playback, the Tritonus sequencer plug-in library [15] has been used for MIDI sequencing.

5.2 Region of interest specification

As mentioned in section 5.3, visual information seeking requires the user to specify a region of interest (ROI) in the data. This ROI specification can then be used to link several visualizations of the data.
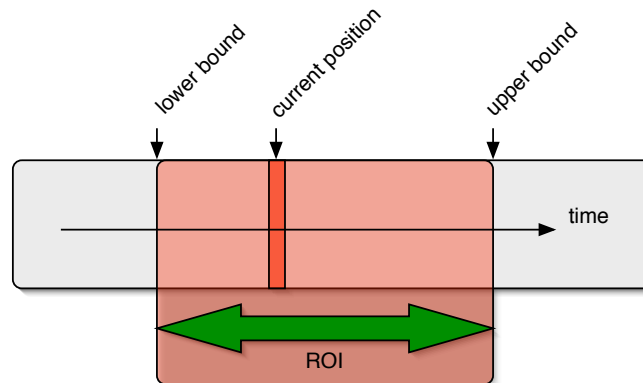
26

Figure 5.2: Region of interest concept

In the current implementation, a ROI is specified in the time domain, by providing a lower and an upper time bound in the performance under investigation. (Linked) views can then visually emphasize data items which lie between the lower and the upper bound. User interfaces may or may not provide locator user interface elements (widgets) to manipulate the actual ROI, therefore they are divided into *active* and *passive* user interfaces.

Figure 5.2 visualizes the concept of the ROI.

5.3 Visual information seeking

Shneiderman [48] proposed the *Visual Information Seeking Mantra* as a general guideline for the implementation and evaluation of interactive information visualization systems.

The basic principles of Shneiderman's proposal - first overview, then zoom&filter, then details on demand - are reviewed and its application to the design of a user interface for interactive visualization of piano performance is discussed.

5.3.1 Overview

Visualizations should provide an entry point to the information that is visualized. By presenting an overview, the user should be able to get a grasp of the basic structure of the data.

A common way to handle large amounts of highly detailed data is to increase the space that is available in the visualization domain and to embed the data into

this space. An intuitive way to do this is to raise the two-dimensional screen space to three dimensions. By providing appropriate navigational capabilities - which allow the user to change his or her viewpoint - it is easy to get an overview of the basic data layout.

Zooming

The user should be able to zoom into regions of interest.

In a three-dimensional environment, *zooming* can be implemented with a change of the viewpoint. If the user is interested in a certain feature, he or she just moves close to it.

Since human beings are used to living in a three-dimensional world, the possibilities for user interaction are less intuitive when the available space is restricted to two dimensions. In this setting, distortion techniques like the fisheye view [29] can be employed, where a subset of the available space is used to show the area of interest enlarged, while the remaining space is used to show the rest of the data as context information (in order to keep the amount of shown information constant, the context information has to be scaled down).

Filtering

It should be possible to filter out uninteresting/unimportant data regions interactively by making them invisible on demand, thereby making it easier for the user to concentrate on important data items.

Details on demand

The user interface should provide a means to select a subset of the data and to show detailed information about the data items present in that subset. This interaction pattern can be exemplified by considering a PerformanceWorm visualization, which provides overview information and points out potentially interesting performance parts, and a tempo curve representation, that shows exact tempo values at user-definable metrical levels of detail.

5.3.2   Focus+context

Shneiderman's concepts can be effectively supported by the idea of *focus+context* visualization [35, 1]. In focus+context visualization, data items of interest (the *focus*) are visually emphasized, while the remaining data items are still shown as

context information. This approach supports the user in perceiving the important data characteristics while still keeping the contextual relations in mind.

A common concept is the degree-of-interest function [35], which assigns an importance measure to each data item. The interactive designation of data items that are of interest is commonly referred to as *brushing* [56], since the idea of "painting" on the data region of interest seems to be very intuitive.

5.3.3   Multiple linked views

Linking of several views to propagate changes in one view to all other views showing a different representation of the same data has been proven to be a powerful concept in various application areas [23, 28].

5.3.4   Direct interaction widgets

Another proposal by Ben Shneiderman [49] are direct interaction widgets. The term *widget* has its origins in the words *window* and *gadget*, and has been used since the introduction of the X Window environment for arbitrary user interface elements (Menus, Windows, Buttons, etc.). Direct interaction widgets are - in contradiction to traditional widgets, which are based on the Look&Feel of the windowing system used - directly embedded into the visualization domain, which makes it possible for the user to manipulate them very intuitively with an appropriate input device (e.g., a mouse, a touch screen, or even haptic devices like a force-feedback data glove).

5.3.5   Application concept

The main subject of this thesis is the design and the implementation of a system based on focus+context visualization and multiple linked views. There are several implementations of systems making use of these techniques in the area of visualization of abstract data (commonly referred to as *information visualization*, InfoViz) and the visualization of medical and flow data (*scientific visualization*, SciViz), e.g., the XmdvTool [41, 56], XGobi [52, 52], Spotfire [17], IVEE [18], and SimVis [28, 32]. However, to the author's knowledge, there doesn't exist a system that explicitly employs those techniques for the visualization of musical data.

In "classical" musicology, researchers would use a combination of several applications to analyze musical performance; a sequencing application (e.g., Apple Logic Audio [2], Steinberg Cubase [11]) to record/edit and pinpoint sections of

interest in a performance. Thereafter, a data analysis/visualization package (e.g., MATLAB [8]) would be employed to create plots of certain interesting parameters for visual comparison.

This workflow should be effectively supported by the application. To meet this requirement, the following main components have been identified.

File loader

The file loader is capable of loading *match files*, which map performances to corresponding scores, into the application's internal data structures (see section 5.4).

Application core

The application core implements the handling of the score-performance matching information and the linking of associated views via a centralized time base (see section 5.5), that keeps track of the current region of interest.

View classes

View classes visualize the performance data and implement user interaction (see section 5.6). Changes to the region of interest are communicated to the time base, which propagates them to the other views.

Inter-application linking module

The inter-application linking modules is capable of propagating region of interest/position updates emanating from user interactions over a TCP/IP (see section 5.5.1) to other instances of the application, which can either reside at the same or at a remote machine. The goal of the inter-application linking facility is to enable simultaneous analysis of several performances of the same score. Linking of several performances can only occur on the score time level (whereas linking of several visualizations of the *same* performances occurs on the performance time level). Therefore, it is necessary to convert the local performance time to the corresponding score time before sending it to another application instance)

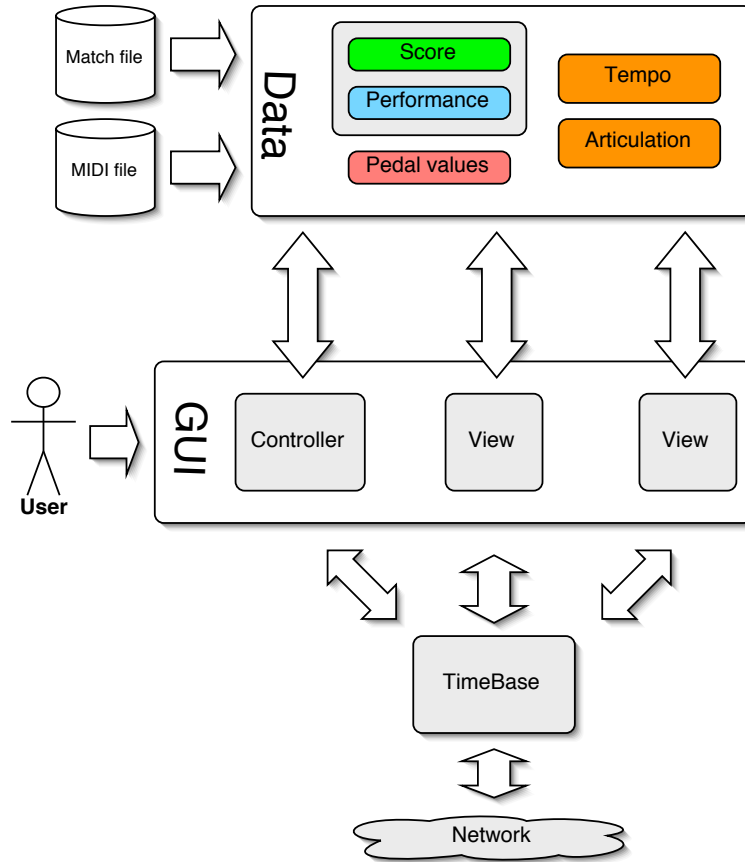Figure 5.3 sketches the application architecture and the data flow between the main components.

Figure 5.3: Application architecture

5.4 File reader

The file reader is responsible for parsing files in a proprietary data format, that stores score-performance matching information (*match files*). In addition to match files, MIDI files, which provide pedal controller information, are loaded and the corresponding data is fed into the internal data structures.

In figure 5.4, a short section from a match file is shown. The first part of a line (before the dash sign) corresponds to a score note:

```
snote(Anchor,[NoteName,Modifier],Octave,Bar:Beat,Onset,
        Duration,BeatNumber,DurationInBeats,ScoreAttributesList)
```

while the second part denotes a played note:

```
...
snote(n1,[e,n],5,1:1,0,1/4,0,1.0,[arp])-note(1,[e,n],5,4831,5421,5421,60).
snote(n2,[g,n],5,1:1,0,1/4,0,1.0,[arp])-note(2,[g,n],5,4884,5402,5402,74).
snote(n3,[c,n],4,1:1,0,1/16,0,0.25,[])-note(3,[c,n],4,4954,5012,5012,65).
snote(n4,[c,n],6,1:1,0,1/4,0,1.0,[s,arp])-deletion.
snote(n5,[c,n],5,1:1,1/16,1/16,0.25,0.5,[])-note(5,[c,n],5,5097,5251,5251,67).
snote(n6,[b,n],4,1:1,2/16,1/16,0.5,0.75,[])-note(6,[b,n],4,5221,5269,5269,63).
snote(n7,[c,n],5,1:1,3/16,1/16,0.75,1.0,[])-note(7,[c,n],5,5346,5395,5395,56).
insertion-note(8,[c,n],4,5469,5680,5769,62).
...
```

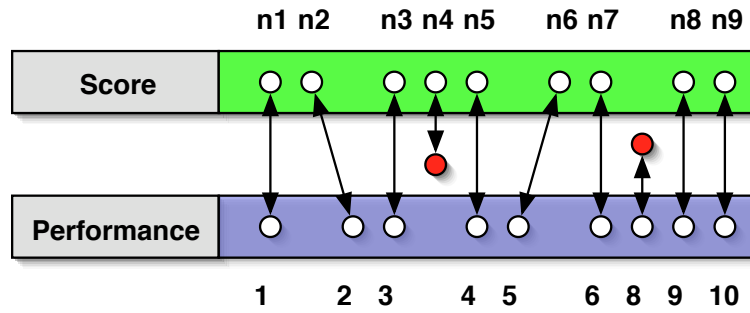Figure 5.4: A part of a match file

Figure 5.5: Score-performance matching

```
note(Number,[NoteName,Modifier],Octave,
        Onset,Offset,AdjOffset,Velocity)
```

The data present in a match file is mapped to an internal data structure (see figure 5.5). Note that score note n4 does not have a corresponding played note (in the match file, this situation is designated as *deletion*), whereas played note 8 does not correspond to any score note (referred to as *insertion* in the match file).

5.5 Application core

The application core consists of data provider components, and it exposes a central time base to all user interface components that are present in the system. The time base maintains the current position in time as well as the lower and upper bound of the region of interest, that is, the smallest and greatest value the current position locator can take.
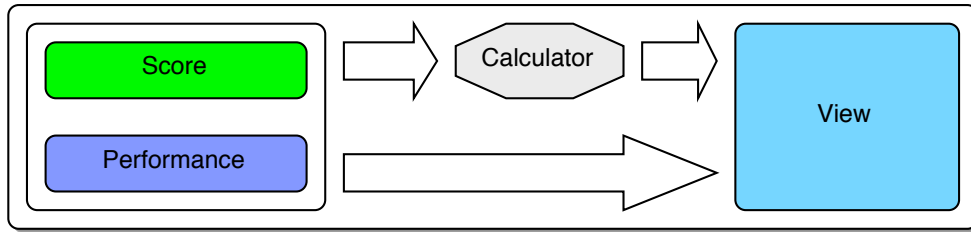
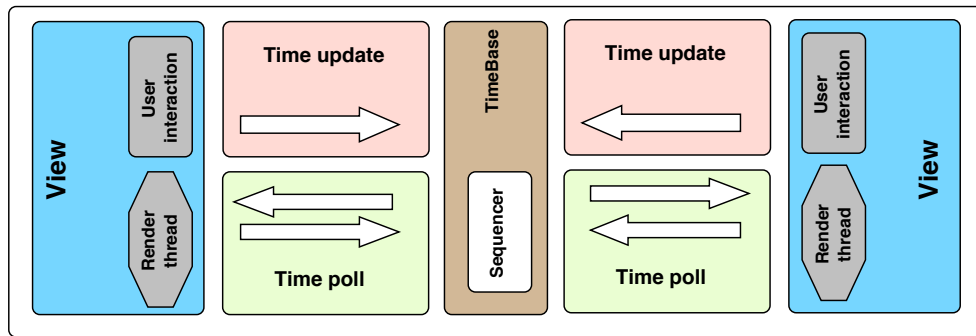Figure 5.6: Calculator classes

Figure 5.7: Time base

The data provider components supply first order data (see section 2.5.1) as well as second order features, as described in 2.5.2. First order data are stored statically, whereas second order features are computed dynamically via specialized *calculator* classes (see figure 5.6).

The time base encapsulates a MIDI sequencer. Performance data is converted to a MIDI data stream and fed to the sequencer. The current time position is also kept by the sequencer, therefore timing information is always accurate, whether the sequencer is running or stopped.

To directly support the control of the region of interest, the current time position, playback state, and warping of the MIDI clock, a controller GUI has been implemented (see figure 5.8).

5.5.1 Inter-application linking

The application core also exposes an external linking interface to other instances of the application via TCP/IP (Java RMI [3]). The network linking architecture is based on a simple publish-subscribe mechanism; the application instances
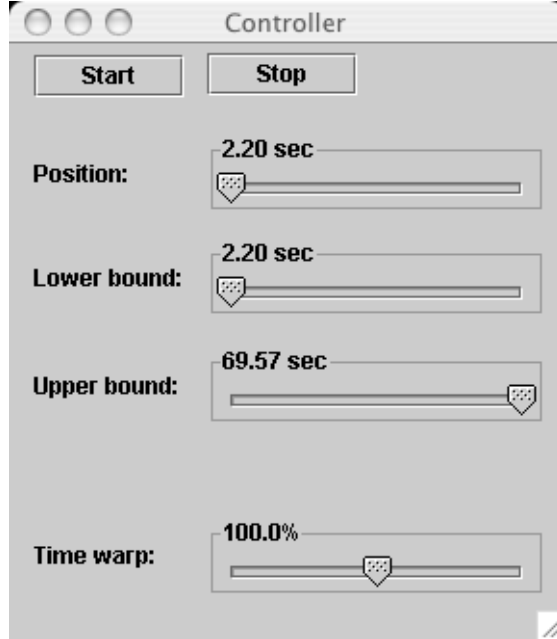
Figure 5.8: Controller panel

subscribe for time changes, thus they act as *observers* [31], that are notified by the server (the *observable*) whenever updates have been sent to the server by an individual client.

In the actual implementation, one application has to start a server. Then the application connects to this server. Other applications can also connect to this server (under the precondition that they have already loaded a compatible performance, i.e., one that is based on the same score data). From this point on, all changes to the actual time position and the lower and upper bound are propagated not only to the local views of each application, but also propagated over the network to all applications that have logged into the server. Figure 5.9 sketches the distributed linking facility.

Conceptually, linking of two applications works like local linking. The current locator position and the region of interest are sent to a server, which in turn propagates updates to all subscribed clients. However, since the clients are associated with different performances (of the same score), updates have to occur in *score time*, which is given in beats, in contradiction to the local linking, which is done with regard to *performance time* (in seconds). Therefore, clients have to translate score time back into their own performance time after receipt of a time update.

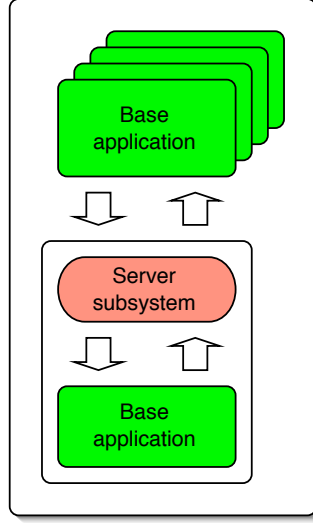Score time t_s can be converted to performance time t_p by linear interpolation:

Figure 5.9: Distributed linking

$$P = findPrev_s(t_s)$$
$$N = findNext_s(t_s)$$
$$\alpha = \frac{onset_s(t_s - onset_s(P))}{IOI_s(P, N)}$$
$$convert_s(t_s) = onset_p(P) + \alpha IOI_p(P, N) \tag{5.1}$$

and performance time t_p can be converted to score time t_s:

$$P = findPrev_p(t_p)$$
$$N = findNext_p(t_p)$$
$$\alpha = \frac{onset_p(t_p - onset_p(P))}{IOI_p(P, N)}$$
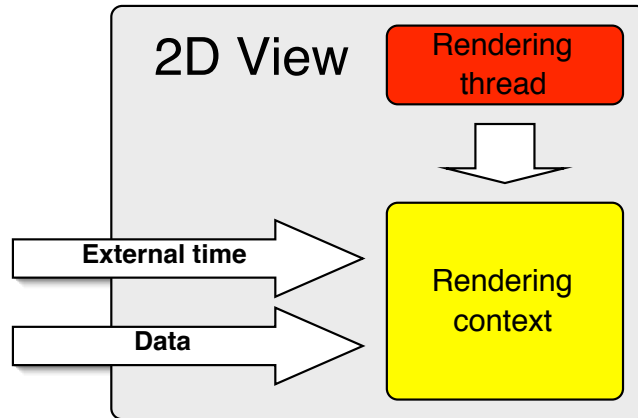$$convert_p(t_p) = onset_s(P) + \alpha IOI_s(P, N) \tag{5.2}$$

Figure 5.10: 2D view architecture

5.6 Visualization views

This section describes the implementation of the visualization views present in the system.

5.6.1 2D views

In the 2D views, a simplified concept has been followed. Since 2D visualizations are comparably simple, no scenegraph structure has been used, incoming data is rendered directly. Moreover, no direct interaction widgets have been implemented (see figure 5.10).

5.6.2 3D views

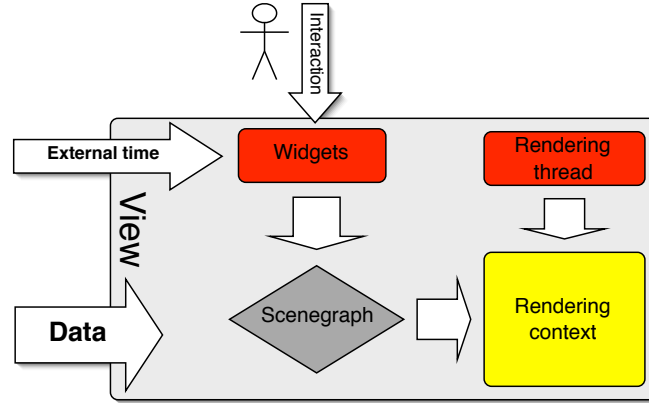3D views are responsible for visualization and user interaction.

Figure 5.11: 3D View architecture

Lifecycle

In the current architecture, every view contains its own scenegraph, which is created during initialization of the view (see figure 5.11). After that, a rendering thread is started. The rendering thread loops over the following steps:

Algorithm 5.6.1: RENDERLOOP()

repeat
$\begin{cases} \text{CHECKINPUT()} \\ \text{UPDATESCENEGRAPH()} \\ \text{TRAVERSESCENEGRAPH()} \end{cases}$
until ($exit = $ **true**)

5.6.3 Widget framework

To manipulate the bounds of the ROI, *time plane* widgets have been implemented (see figure 5.12).

Time plane widgets have been implemented as direct interaction widgets. That means that they can be directly manipulated in the 3D domain by dragging the red cube at the bottom part of the plane with the mouse.

Besides the time plane widgets, there are two more interaction tools available. First, by dragging the ground plane of the visualization, it can be translated along the time dimension. Second, the viewpoint can be freely changed by common rotate/translate interaction patterns. In combination, those techniques effectively implement overview- and zoom-functionality.
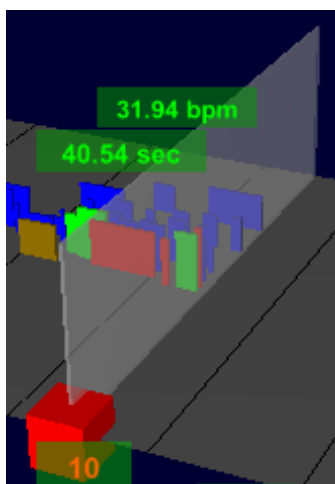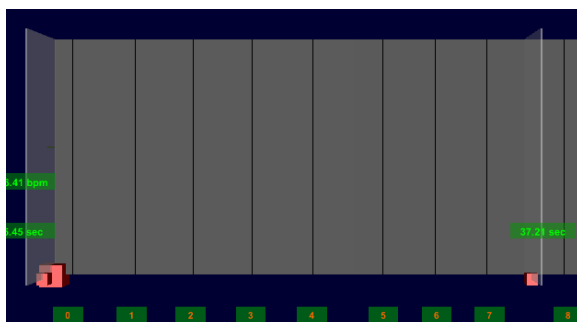
Figure 5.12: A time plane widget
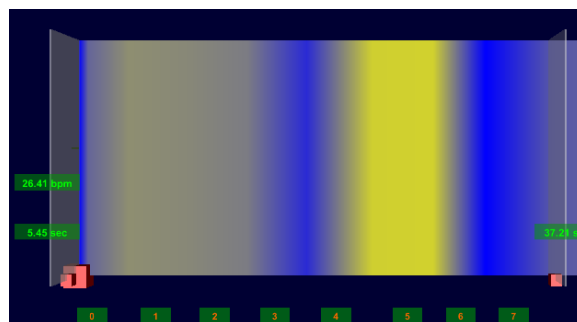

Figure 5.13: *Line* tempo cues

Figure 5.14: *Color* tempo cues

The basic widgets (time planes, ground plane) have been also extended to directly incorporate visualization elements. To display numerical parameters (exact time values, tempo at a certain point in time), text labels have been attached to the time planes.

The bottom plane has been also enriched visually. Two different types of *tempo cues*, which sketch the beat/tempo structure of the performance, have been implemented (see figure 5.6.3. In the line visualization, the distance of beats (at a selectable metrical level) is visible, while the color visualization shows tempo mapped to color values (blue corresponds to a low, while red corresponds to a high relative tempo).

Alongside the widget elements, bar numbers are displayed.

5.6.4   Implementation of the 3D widgets

All 3D widgets are implemented as nodes in the scene graph. Thus, user interaction implies an update to the scenegraph structure.

Since the current implementation is based on traditional input and output devices (mouse and screen), a mapping from two-dimensional input space to three-dimensional world space has to be defined.

Picking is a well-known concept that enables users to select objects in a three-dimensional world by pointing on its two-dimensional representation in screen space. The actual conversion from 2D input coordinates to 3D world is done by constructing a picking ray that emanates from the viewer and passes through the point on the screen that has been designated by the input device, and testing the objects in the world representation for intersection with this ray. However, this has solved only a part of the problem.

The open question is how objects should react to user gestures. In two-dimensional user interfaces, a *drag* is performed by the following steps:

Algorithm 5.6.2: DRAG()

$selected \leftarrow$ **true**
repeat
$\begin{cases} coords \leftarrow \text{GETCOORDS}() \\ \text{MOVETO}(object, coords) \\ selected \leftarrow \text{ISSELECTED}() \end{cases}$
until $(selected =$ **false** $)$

It is easy to compute object positions during a drag operation, since the object is flat and completely embedded in input space. However, in three-dimensional object space this is not the case. It is therefore necessary to map the two-dimensional screen input space to a plane, which is perpendicular to the picking ray (a *drag plane*). If the intersection point of the picking ray with the picked object's geometry and the plane are coplanar, new positions can be computed - like in the

two-dimensional case - by intersection the picking ray with the plane. Thus, the necessary steps to perform a drag operation in 3D are:

Algorithm 5.6.3: Drag3D()

$selected \leftarrow$ **true**
$ray \leftarrow$ CreateRay()
$objCoords \leftarrow$ ObjectIntersect(ray)
$plane \leftarrow$ CreatePlane($objCoords, ray$)
repeat
$\begin{cases} newCoords \leftarrow \text{PlaneIntersect}(plane, ray) \\ \text{MoveTo}(object, newCoords) \\ selected \leftarrow \text{IsSelected}() \end{cases}$
until ($selected =$ **false**)

The lastly described algorithms and components provide a framework for the visualizations that will be described in the following sections.

5.6.5   PianoRoll implementation

In this section, the piano roll implementation is described.

Visualization mapping

3D piano rolls are an extension to the classical 2D piano roll representation. In 3D piano rolls, time, pitch, and velocity are mapped to 3D space, as depicted in figure 5.15.

Additionally to the spatial mapping, a color mapping has been established: *Legato* notes are drawn in greenish tones, while *staccato* notes are drawn in reddish colors. This gives the user a quick impression of the pianist's articulation style.

Additional GUI components

Besides the direct interaction widgets, a set of traditional GUI components, allowing the interactive selection of the beat cue mode, the metrical level used for the beat cues, and default views, is provided (see figure 5.16). Additionally, it is possible to enable/disable visualization of notes that are ahead of the actual position of the current time locator widget, and a special mode forces the camera to follow the actual position of the time locator widget (*sticky camera*).
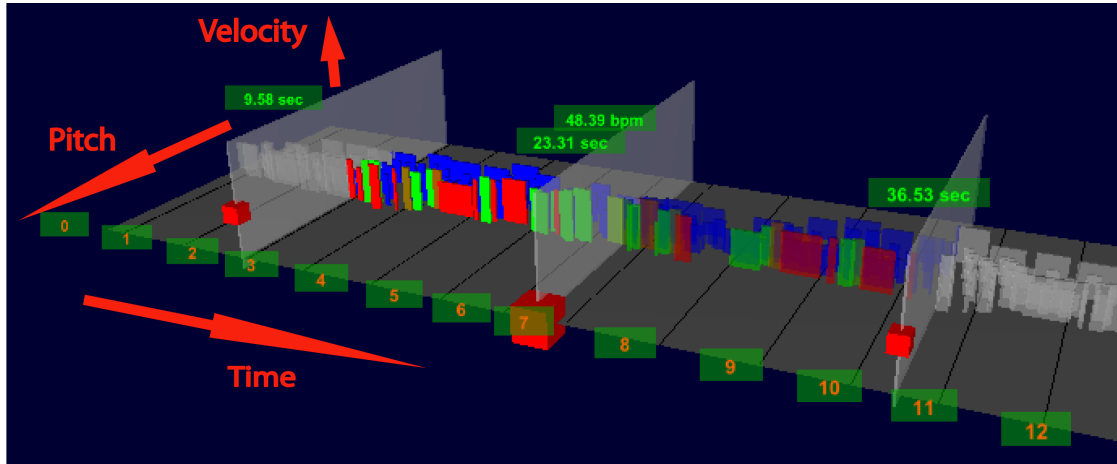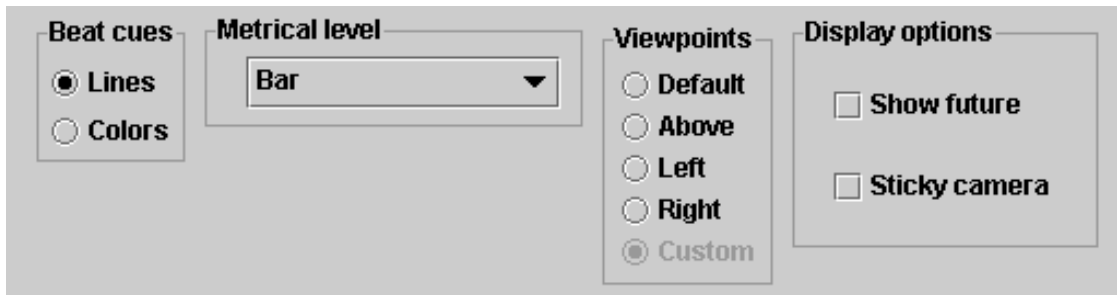
Figure 5.15: 3D piano roll

Figure 5.16: PianoRoll 2D GUI components

5.6.6 PerformanceWorm implementation

Visualization mapping

Similar to the PianoRoll, the PerformanceWorm visualization is integrated with the framework that has been described in section 5.6.3. The worm is modeled as a sweep curve in 3D-space, that interpolates discrete points in time/tempo/dynamics-space (see figure 5.17).

Excursus: Space curves

As in the implementation by Dixon et al. [25], the PerformanceWorm is calculated by measuring tempo and dynamics at regularly spaced points in score time. However, the calculation has been extended to work not only at the beat level, but
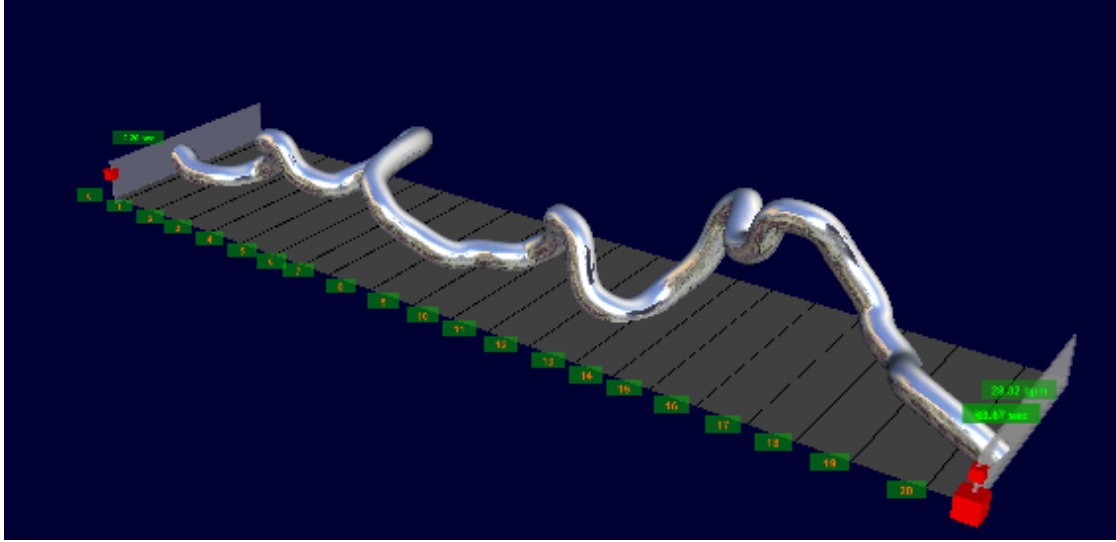
41

Figure 5.17: The 3D PerformanceWorm visualization

also at the bar- and the sub-beat levels. Thus, it is possible to filter out variations according to a desired level of detail.

The measured (**tempo**, **dynamics**) points are then converted to points in three-dimensional (**tempo**, **dynamics**, **performanceTime**) space and interpolated by fitting a third-order polynomial (a cardinal spline [36]) to the data. The resulting continuous space curve is the center line of the PerformanceWorm.

A *sweep representation* [36] is constructed by moving a two-dimensional geometric shape along a sweep path. Sweep paths can be straight lines, periodic curves, or even freely defined space curves. A sweep representation has been used to model the visual appearance of the 3D PerformanceWorm by sweeping 2D geometric primitives along a cardinal spline curve, which interpolates points in (**tempo**, **dynamics**, **performanceTime**) space.

In order to determine the orientation of the shapes that make up the cross-sections of the sweep, local reference frames - a tool from differential geometry - are used. A common reference frame has been invented by Frenet. It is an orthonormal system made up from the tangent, the normal, and the binormal vector of the space curve.

Given a parametrized space curve $\mathbf{f}(s)$, the Frenet reference frame $\mathbf{T}, \mathbf{N}, \mathbf{B}$ is made up of the *tangent vector* $\mathbf{T}(s)$, the *principle normal vector* $\mathbf{N}(s)$, and the *binormal vector* $\mathbf{B}(s)$.

42

$$\mathbf{T}(s) = \frac{\mathbf{f}'(s)}{|\mathbf{f}'(s)|} \tag{5.3}$$

$$\mathbf{B}(s) = \frac{\mathbf{f}'(s) \times \mathbf{f}''(s)}{|\mathbf{f}'(s) \times \mathbf{f}''(s)|} \tag{5.4}$$

$$\mathbf{N}(s) = \mathbf{B}(s) \times \mathbf{T}(s) \tag{5.5}$$

However, this method is very sensitive to zeros in the second derivative, i.e., it becomes unstable whenever the space curve approaches a straight line. In [21], a more stable incremental calculation rule has been proposed:

$$\mathbf{T}(s_{n+1}) = \mathbf{T}(s_{n+1})) \tag{5.6}$$

$$\mathbf{N}(s_{n+1}) = \mathbf{B}(s) \times \mathbf{T}(s_{n+1}) \tag{5.7}$$

$$\mathbf{B}(s_{n+1}) = \mathbf{T}(s_{n+1}) \times \mathbf{N}(s_{n+1}) \tag{5.8}$$

Thus, the binormal vector $\mathbf{B}(s)$ is used to calculate the reference frame at the parameter value $s + 1$.

From the column vectors $\mathbf{T}$, $\mathbf{N}$, and $\mathbf{B}$, a rotation matrix, that transforms the cross-sections of the worm into the right orientation with respect to world coordinates, can be computed:

$$\mathbf{F}(s) = [\mathbf{T}(s)\mathbf{N}(s)\mathbf{B}(s)] \tag{5.9}$$

If $\mathbf{f}(s)$ represents a space curve and $\mathbf{c}(u)$ represents a two-dimensional cross-section in a reference orientation (e.g., $\mathbf{c}(u) = [c_x(u)c_y(u)0]^T$), the points lying on the surface of the sweep curve $\mathbf{w}$ can be computed as

$$\mathbf{w}(s, u) = \mathbf{F}(s)\mathbf{c}(u) \tag{5.10}$$

If the cross-section $\mathbf{c}(u)$ is differentiable with respect to u, it is also easy to calculate normal vectors for the individual points:

$$\mathbf{n_w}(s, u) = \mathbf{F}(s)\mathbf{c}'(u) \tag{5.11}$$

By taking the above considerations into account, it is possible to construct a connected set of points (a *mesh*) that approximates the ideal surface of a sweep curve. This representation can then be rendered using a graphics API like OpenGL.
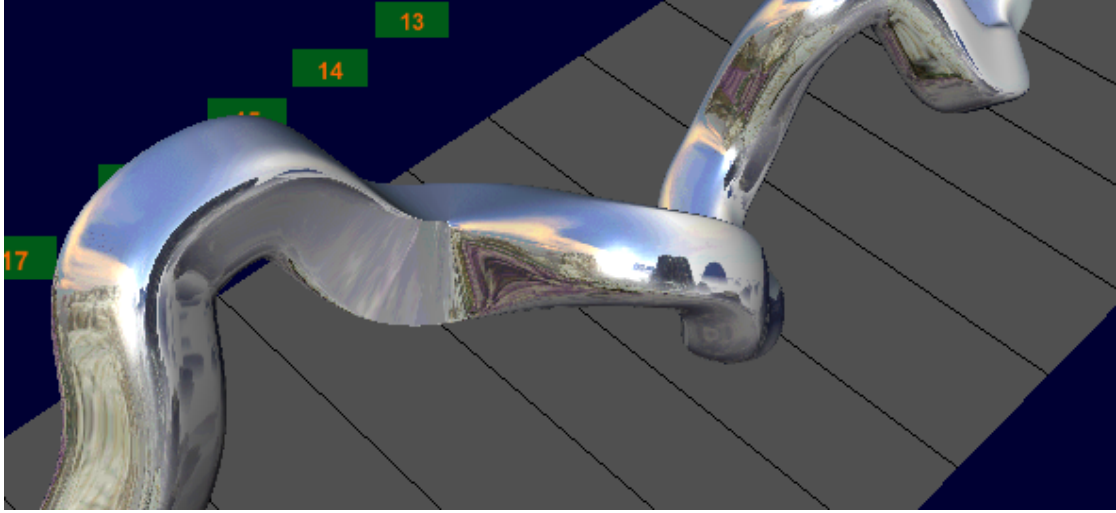
Figure 5.18: Articulation mapping with superelliptic cross-sections

An extension: The articulation PerformanceWorm

A visualization that shows articulation values by modulating the cross-sectional shapes of the PerformanceWorm has been implemented. The idea was that *legato* articulation style should be reflected with rather round shapes, while *staccato* articulation should produce rather angular shapes. In search of an adequate cross-sectional shape, the superellipse (see section 5.6.6) was chosen, since it provides the ability to continuously morph from an angular to a round shape, which was necessary to maintain the smooth appearance of the PerformanceWorm. Figure 5.18 shows how the cross-section of the worm modulates over time, thereby indicating articulation style at that point in time.

Excursus: Superellipses

The *superellipse* [20] is a generalization of the ellipse. It can be defined in an implicit equation as

$$\left| \frac{x}{r_x} \right|^{2/s} + \left| \frac{y}{r_y} \right|^{2/s} = 1 \tag{5.12}$$

When $r_x = r_y$, one may speak of "supercircles". Then, variation of the shape parameter s produces the shapes in figure 5.19.

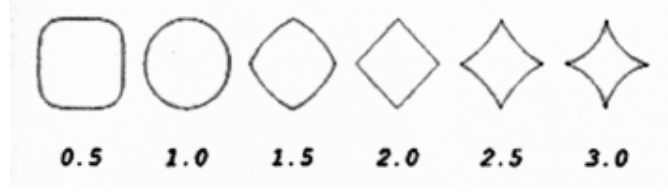Equation 5.12 can be rewritten in parametric form as:

Figure 5.19: Supercircles for values from 0.5 to 3.0

$$x(u) = r_x * sgn(\cos(u)) * |\cos(u)|^s \tag{5.13}$$

$$y(u) = r_y * sgn(\sin(u)) * |\sin(u)|^s \tag{5.14}$$

$$z(u) = 0; \tag{5.15}$$

The normals can be calculated as:

$$n_x(u) = r_x * sgn(\cos(u)) * |\cos(u)|^{2-s} \tag{5.16}$$

$$n_y(u) = r_y * sgn(\sin(u)) * |\sin(u)|^{2-s} \tag{5.17}$$

$$n_z(u) = 0 \tag{5.18}$$

5.6.7 Additional GUI components

Like the PianoRoll visualization, the PerformanceWorm provides additional 2D UI elements (see figure 5.20). The worm shape can be adjusted by selecting the desired metrical level and scaling values for tempo and dynamics values. Display of the future can be turned on and off, the camera can be switched to *sticky*, articulation values can be used to modulate the shape that makes up the cross sections of the worm (see section 5.6.6). Furthermore, pre-defined viewpoints can be selected.

5.6.8 Tempo curve, dynamics curve, and pedal curve

The tempo and dynamics curves have been implemented as multi-level, linked focus+context charts. The current focus is defined as the time span between the lower and upper time bounds that can be adjusted in the 3D views or in the controller panel. The focus+context visualization is achieved by applying a nonlinear
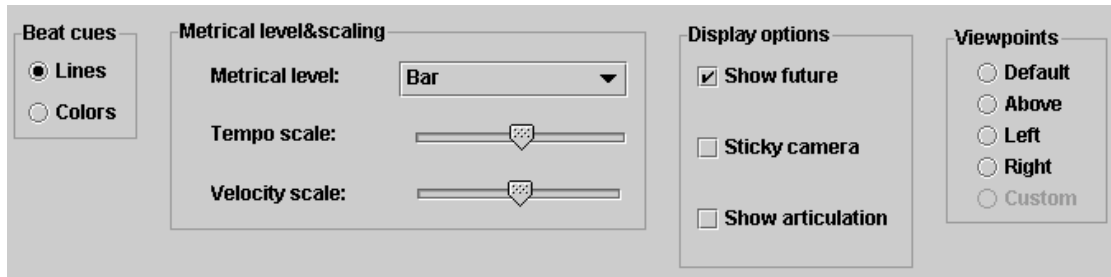
Figure 5.20: PerformanceWorm 2D GUI components

distortion function to the data, thus creating a fisheye view-like effect, that magnifies the *focus* region and scales down the *context* (the region outside the time bounds). Figure 5.21 demonstrates the usage of the fisheye view technique to magnify regions of interest in the tempo microstructure of a performance. Additionally to the lower and upper time bounds, the current position in time is indicated by a vertical line, and bar lines are denoted by dashed vertical lines.

The tempo/dynamics microstructure can be analyzed at different metrical levels of detail by selecting an appropriate level in the menu (see figure 5.22).

Like the tempo and the dynamics visualizations, the pedal visualization offers a fisheye view interface (see figure 5.23); however, no multi-level extension is provided, since the use of a multi-level expansion of pedal usage is generally questionable.
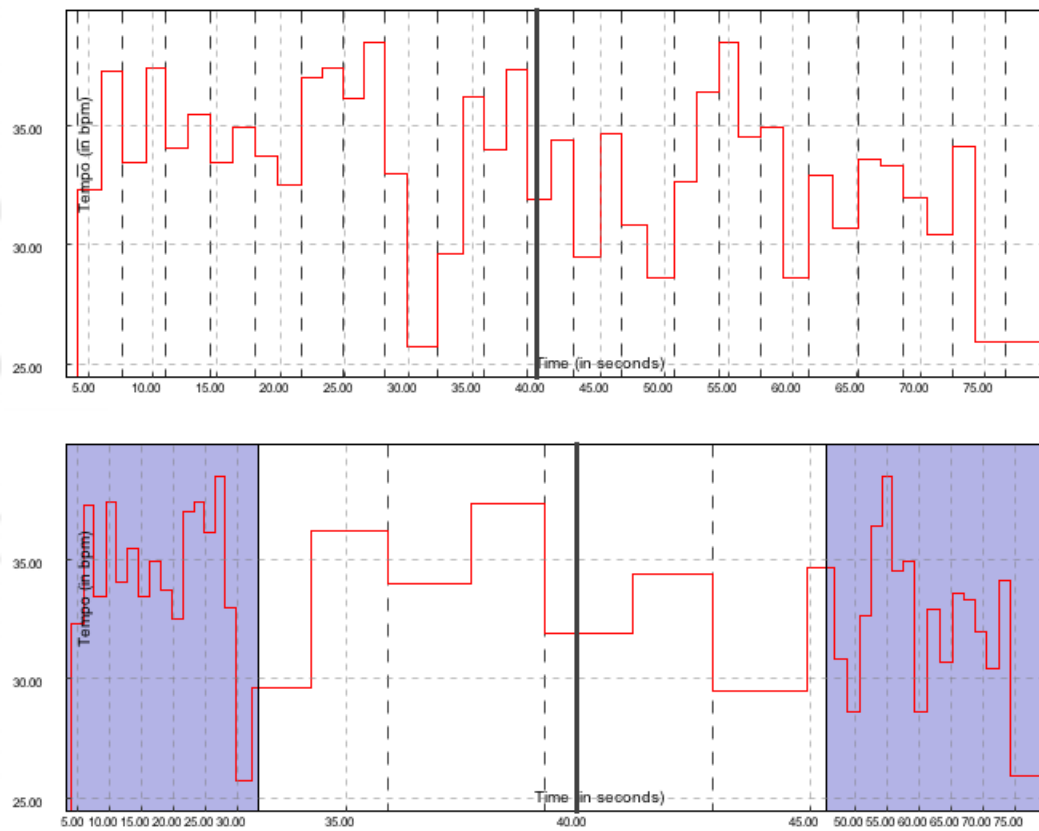
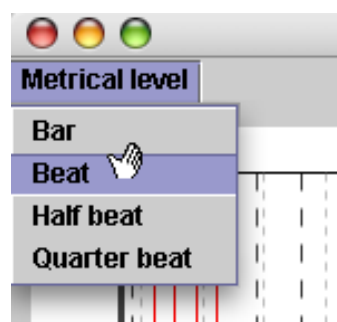Figure 5.21: Focus+context in the tempo visualization

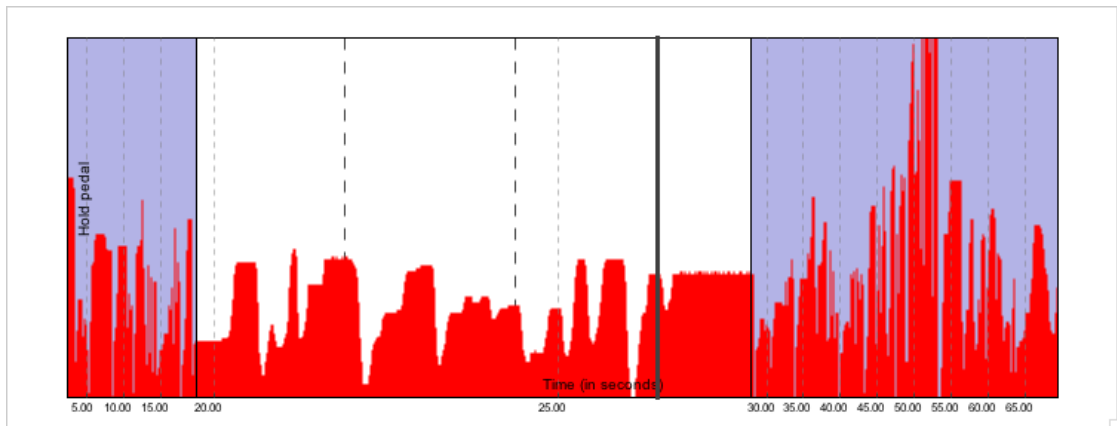Figure 5.22: Selection of metrical level

Figure 5.23: Focus+context visualization of pedal values

48

Chapter 6

An interactive analysis session

> I am among those who think that science has great beauty. A
> scientist in his laboratory is not only a technician: he is also a child
> placed before natural phenomena which impress him like a fairy tale.
>
> *Marie Curie (1867 - 1934)*

In this chapter, a demonstration of the application's main features is given. Two performances of the first movement of Chopin's Etude Op.10, No. 3 are analyzed with special emphasis on the tempo shaping and the timing microstructure.

6.1 Application setup

To show commonalities and differences in two performances of the same score, the inter-application linking feature of the application is used in this session. Thus, in a first step, two instances of the application are started. Then, the files `op10_3_1#09.match` and `op10_3_1#18.match` are loaded.

Applications can be either linked locally or over a remotely over a network connection. By using the Java Remote Method Invocation mechanism, it was possible to implement inter-application communication over TCP/IP with minimal implementation overhead.

After startup and file loading, the application console, which provides the user with important messages about application state and errors, is presented to the user (figure 6.1).

After loading the score-performance matching information, the two application instances are linked. By linking the applications, updates to lower/upper score time bounds (region of interest) and current score time position are shared among
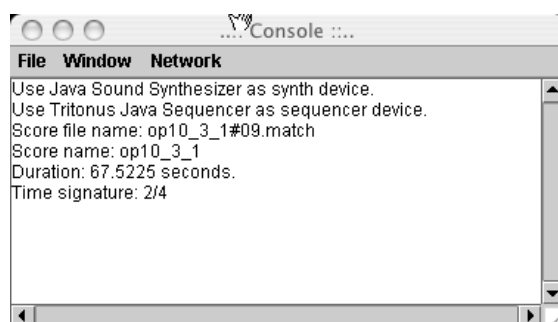
Figure 6.1: The application console

the instances. All associated views show the same time values, either by moving the direct interaction widgets to the correct location, or, as in the case of the 2D view, by adjusting the current focus region and positioning the time locator. Thus, it is possible to directly compare performances interactively.

6.2 Overview

Piano roll visualizations are used to grasp the overall structure of the two performances (6.2). Individual notes, visualized with block-like shapes, are enhanced with color coding, showing the articulation style of the performer. Articulation has been modeled as note overlap: Overlapping (legato) notes are greenish, while non-overlapping (staccato) notes are colored in red tones. Only the melody voice - which has been marked up in the input data - has been subject to articulation calculation.

The lower and upper time bounds have been adjusted to bars 6 to 14, since this region seems to be interesting with respect to the tempo shaping, which will be subject to further investigation in the following steps.

Tempo values have been encoded as colors at the bottom plane of the piano roll. Those *tempo cues* suggest that the performer **B** varies tempo more expressively than performer **A**. Yellowish tones denote faster tempos, while blueish tones denote slower tempos; performer **A** plays a slight ritardando at bar 7, while performer **B** plays two strong ritardandi at bar 7 and bars 11 and 12. It is also possible to change the metrical level that is used for the calculation of local tempi in the piano roll. However, this doesn't seem appropriate for an *overview* visualization; tempo/timing microstructure can be visualized with high accuracy in the detail visualization, that will be described in the following.
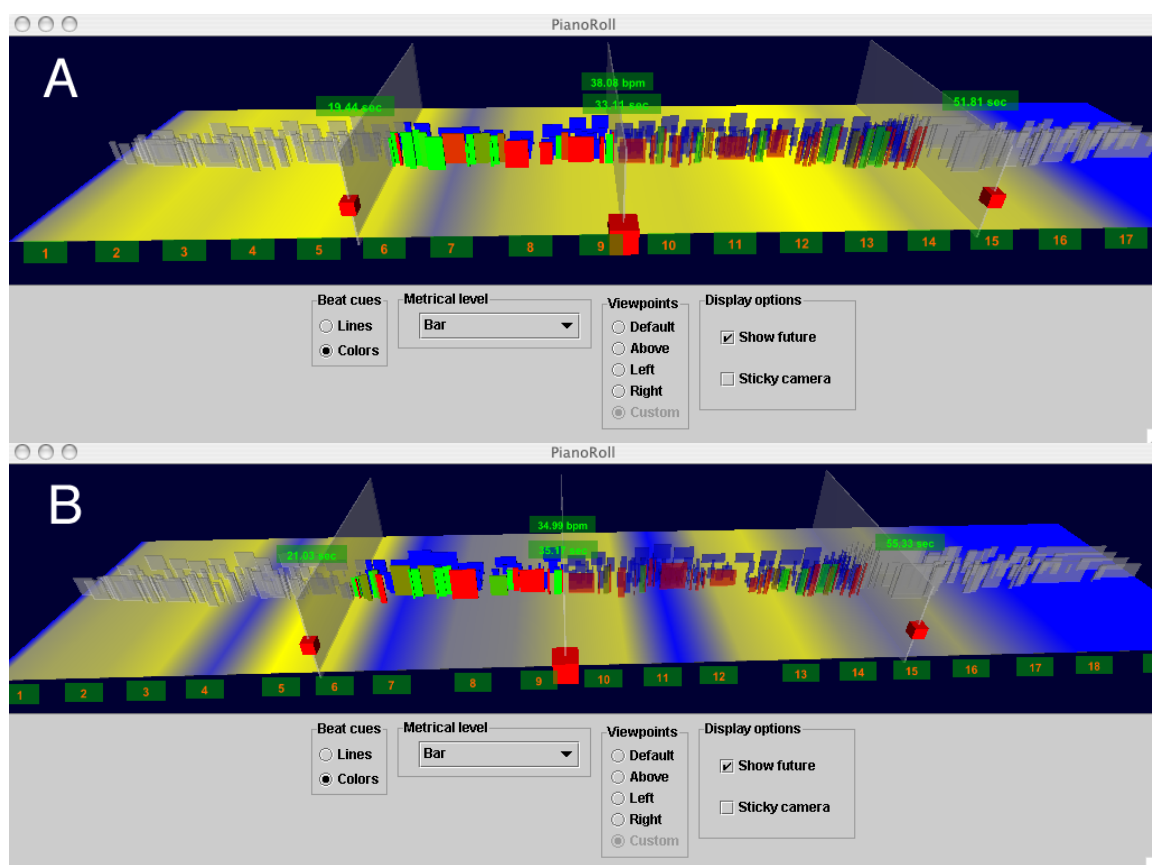
50

Figure 6.2: Piano roll visualizations of two performances

6.3 Timing and dynamics microstructure

By opening **tempo curve** views from both instances, the timing (micro)structure of the performances can be compared. In figure 6.3, which displays tempo values averaged over bar durations, it can be observed, that the second pianist puts much more emphasis on the shaping of the tempo structure. The tempo varies from 28 to 40 beats per minute in the second performance, while in the first performance, it varies from 33 to 41 BPM.

A linked dynamics visualization 6.3 bares slight correlations between tempo and dynamics in both performances (accelerando-crescendo and ritardando-decrescendo patterns). This observation is well known in performance research, and it gives evidence for a "the faster, the louder" rule. However, performers with advanced expressive skills often handle tempo and dynamics more independently.
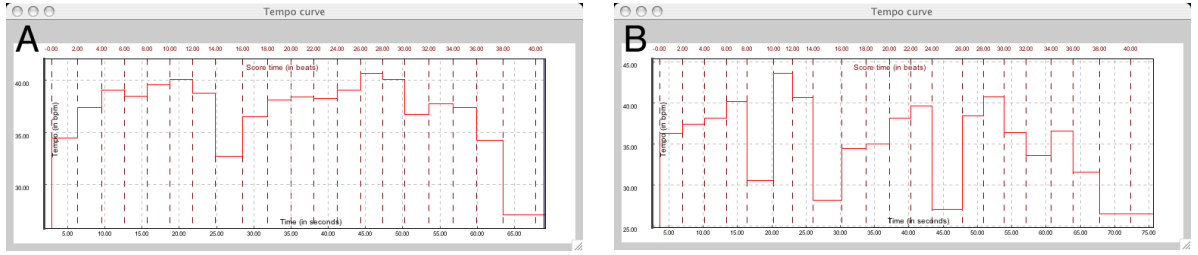
Figure 6.3: Tempo curves of two performances at the bar level
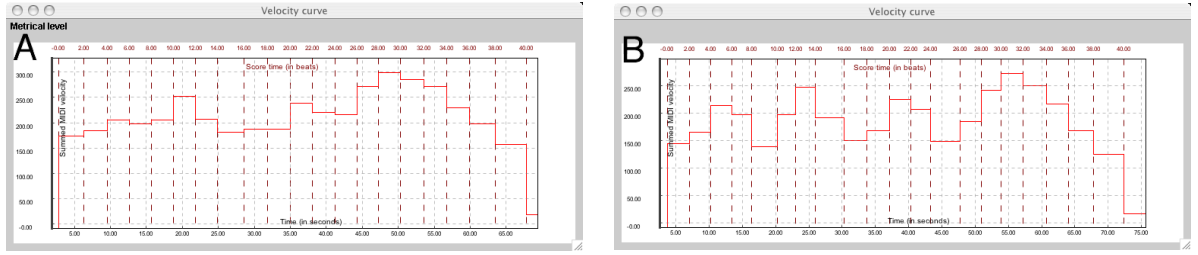
Figure 6.4: Dynamics curves of two performances at the bar level

At the sixteenth note level (which is the lowest metrical level that appears in the musical piece under investigation), the local variation of tempo values becomes clearly visible. Figure 6.3 shows the timing microstructure of the two performances. Here, it can be clearly observed, that performer **A** plays in a rather monotonic style (with the exception of a strong ritardando in bar 7) while performer **B** puts more emphasis on expressive timing.

Figure 6.5: Tempo curves of two performances at the sixteenth note level

Figure 6.6: Tempo projection of two performances

6.4   Integrated tempo-dynamics visualization

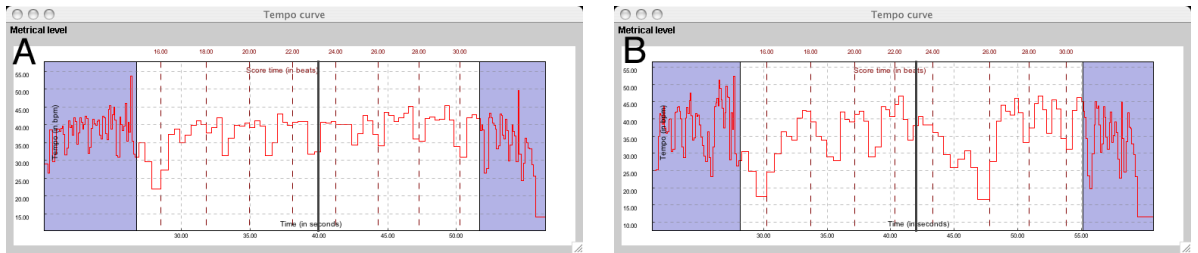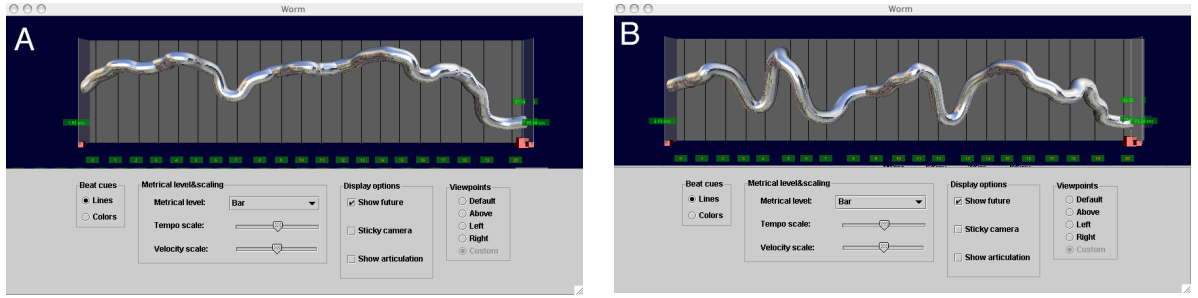The PerformanceWorm provides an integrated visualization of tempo and dynamics in 3D space. By rotating the view, projections of the data can be investigated. Figure 6.2 shows the evolution of tempo over time. Expressive tempo shaping becomes apparent in performance **B**; apart from a ritardando in bars 6 and 7, the visualization of performance **A** does not provide an indication of tempo variation.

As the PianoRoll visualization, the PerformanceWorm is intended to provide an overview of the data, therefore it is not very convenient to construct it at metrical levels below the bar level. It would be possible to compute tempo/dynamics points at lower levels and smooth it afterwards, but that would yield approximately identical results to the here implemented piecewise cubic interpolation of tempo/dynamics values which have been computed at the bar level. However, this problem has not been fully investigated and will be subject to future work.

When viewed from adequate viewpoints, the PerformanceWorm visualization metaphor clearly shows characteristic expressive aspects of performances. In figure 6.4, one can clearly observe the counter-clockwise patterns in performance **B**, which indicates highly expressive shaping of tempo and dynamics. In performance **A**, tempo is relatively constant throughout the whole piece. Variations occur primarily in dynamics, which results in a less smooth appearance of the PerformanceWorm visualization. An interesting conclusion is, that visually appealing worm shapes in general correspond to highly musical performances, while "unbalanced" PerformanceWorm shapes indicate rather "mechanic" performances that do not sound very pleasant to the human ear.
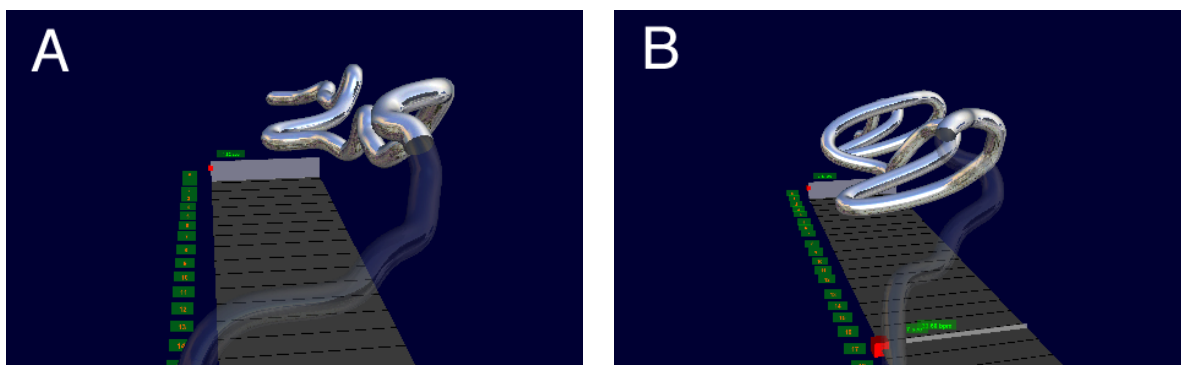
Figure 6.7: Worm visualizations of two performances

6.5 Audio-visual exploration

To fully support exploration in the visual as well as the audible domain, the application supports synchronized animated visualization and playback of performances. All visualization views have been extended with a user interface element that lets the user locate the current position in time during exploration or playback. Furthermore, lower and upper time bounds are interpreted as start/stop markers for the sequencer that plays back the MIDI file representation of the performance. Thus, a user can easily specify a region of interest and playback this region multiple times. To support audio-visual analysis of fast passages, the reference speed of the MIDI sequencer can be also controlled, effectively providing a global time warping of the performance. All parameters (current time position, lower/upper bound, playback state, time warping) can be adjusted with a controller view (see 5, 5.8).

Chapter 7

Conclusions and future work

> The purpose of computing is insight, not numbers.
>
> *R.W.Hamming*

In this thesis, a framework for interactive visualization of expressive piano performances has been presented. The design of an easy-to-use graphical interface and the support for common workflows in music performance research has been emphasized. Well-know techniques from the field of information visualization, like focus+context visualization, linked views, and interactive data selection have been used to provide the user with abilities to specify and investigate regions of interest in the performance data.

The application of the *fisheye view* concept to multilevel tempo-, dynamics-, and pedal-diagrams has been proven as a very useful extension, enabling the user to investigate performance data at various metrical levels. Focus+context methods enable interactive zooming into regions of interest while still keeping the contextual information in mind.

The extension of the well-known visualization metaphors *PianoRoll* and *PerformanceWorm* to 3D visualizations provides a visually compelling appearance. However, to actually use those visualizations in performance research, thorough evaluation and feedback by researchers would be necessary. An exciting project would be the implementation of the 3D visualizations in immersive environments like a back-projection table, that enables the user to interact with the direct interaction widgets by pointing directly on them (without the use of a spatially dislocated pointing device like a mouse).

MIDI data has been proven to be useful for the analysis of discrete features (pitch, onset, offset), however, it is inferior when it comes to the representation of

perceptual features (dynamics). Thus, an extension that handles audio *and* MIDI data could substantially enhance the visualizations of dynamics.

The visual system is the most powerful sensory organ of humans, providing a broadband data channel to the human brain, thus visualization methods promise to be able to transport larger amounts of data than traditional data representation methods. *Visual data mining* - the explorative visual detection of previously unseen structures in large amounts of data - is currently a hot topic in the visualization and in the data mining community. Non-visual data mining methods aim at answering a set of questions by executing pre-defined algorithms, which always carry the danger of the introduction of methodological or model errors (in machine learning speech, this is called a *bias*). Thus, machine learning methods are always build on a limited amount of a priori knowledge.

Explorative methods aim at human-understandable data representations, that allow humans to perform reasoning based on visual representations of the data, without a priori knowledge of the data characteristics. The framework that has been presented in this thesis fulfills basic requirements to a visual data mining system - it supports multiple linked views, interactive data selection, a region-of-interest concept, and new visualization methods can be easily integrated.

Bibliography

[1] *Readings in Information Visualization: Using Vision to Think.* Morgan Kaufmann Publishers, 1998.

[2] Apple Logic Audio. `http://www.apple.com/logic/`, April 2005.

[3] http://java.sun.com/products/jdk/rmi/. `http://java.sun.com/products/jdk/rmi/`, April 2005.

[4] Java3D. `http://java.sun.com/products/java-media/3D/`, April 2005.

[5] JNI - Java Native Interface. `http://java.sun.com/j2se/1.4.2/docs/guide/jni/`, April 2005.

[6] JOGL. `https://jogl.dev.java.net`, April 2005.

[7] L. Bösendorfer Klavierfabrik Gmbh. `http://www.boesendorfer.com/index.html`, 2005.

[8] MATLAB - The language of Technical Computing. `http://www.mathworks.com/products/matlab/`, April 2005.

[9] OpenGL - The Industry Standard for High Performance Graphics. `http://www.opengl.org/`, April 2005.

[10] PortAudio - portable cross-platform Audio API. `http://www.portaudio.com/`, April 2005.

[11] Steinberg Cubase. `http://www.steinberg.de/Category_sb3ae5-2.html`, April 2005.

[12] The Aviatrix3D project. `http://aviatrix3d.j3d.org/`, April 2005.

[13] The MIDI manufacturer's association. `http://www.midi.org`, April 2005.

[14] The QT toolkit. `http://www.trolltech.com`, April 2005.

[15] Tritonus JavaSound. `http://www.tritonus.org`, April 2005.

[16] Xith3D: Powerful 3D Scenegraph and Renderer for Java. `http://xith.org/tiki-index.php`, April 2005.

[17] C. Ahlberg. Spotfire: An information exploration environment. *SIGMOD Record (ACM Special Interest Group on Management of Data)*, 25(4):25–29, 1996.

[18] C. Ahlberg and E. Wistrand. IVEE: An information visualization and exploration environment. In *Proc. IEEE Symposioum on Information Visualization 1995*, pages 66–73, 1995.

[19] P. Allen and R. B. Dannenberg. Tracking musical beats in real time. In *Proceedings of the International Computer Music Conference*, pages 140–143, International Computer Music Association, 1984.

[20] A. Barr. Superquadrics and angle-preserving transfomations. *IEEE Computer Graphics and Applications*, (1):11–22, 1981.

[21] Jules Bloomenthal. *Calculation of reference frames along a space curve*, pages 567–571. Academic Press Professional, Inc., San Diego, CA, USA, 1990.

[22] R. Bresin and A. Friberg. Emotional coloring of computer-controlled music performances. *Computer Music Journal*, 24(4):44–63, 2000.

[23] A. Buja, J. McDonald, J. Michalek, and W. Stuetzle. Interactive data visualization using focusing and linking. In *Proceedings IEEE Visualization '91*, pages 156–163, 1991.

[24] A. T. Cemgil, H. J. Kappen, P. Desain, and H. Honing. On tempo tracking: Tempogram representation and Kalman filtering. *Journal of New Music Research*, 28:4:259–273, 2001.

[25] S. Dixon, W. Goebl, and G. Widmer. The performance worm: Real time visualisation based on langner's representation. In M. Nordahl, editor, *Proceedings of the 2002 International Computer Music Conference*, pages 361–364, San Francisco, CA, 2002. International Computer Music Association.

[26] Simon E. Dixon. Automatic extraction of tempo and beat from expressive performances. *Journal of New Music Research*, 30(1):39–58, 2001.

[27] Simon E. Dixon. *Analysis of Musical Content in Digital Audio*. 2003.

[28] Helmut Doleisch, Martin Gasser, and Helwig Hauser. Interactive feature specification for focus+context visualization of complex simulation data. In *VisSym*, 2003.

[29] G. Furnas. The fisheye view: A new look at structured files. Technical memorandum, Bell Laboratories, Murray Hill, New Jersey 07974, U.S.A., 1981, 1981.

[30] Alf Gabrielsson. The performance of music. In D. Deutsch, editor, *Psychology of Music*, pages 501–602, San Diego, CA, 1999. Academic Press.

[31] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. Design patterns - elements of reusable object-oriented software, 1995.

[32] Martin Gasser. Fast focus+context visualization of large scientific data. In *Proceedings of the 2004 Central European Seminar on Computer Graphics*, Budmerice, Slovakia, 2004.

[33] Masataka Goto. An audio-based real-time beat tracking system for music with or without drum-sounds. *Journal of New Music Research*, 30(2):159–171, 2001.

[34] Masataka Goto and Yoichi Muraoka. A real-time beat tracking system for audio signals. pages 171–4, 1995.

[35] H. Hauser. Generalizing focus+context visualization. In *Dagstuhl Seminar 03231: Scientific Visualization: Extracting Information and Knowledge from Scientific Data Sets*, 2003.

[36] D. Hearn and P. Baker. *Computer Graphics - The C Version*. Prentice-Hall, Inc., 1997.

[37] M. Henderson. *University of Iowa studies in the psychology of music*, chapter Rhytmic organization in artistic piano performance, pages 281–305. University of Iowa, Iowa, 1937.

[38] Rumi Hiraga. Case study: a look of performance expression. In *VIS '02: Proceedings of the conference on Visualization '02*, pages 501–504. IEEE Computer Society, 2002.

[39] Rumi Hiraga and Noriyuki Matsuda. Visualization of music performance as an aid to listener's comprehension. In *AVI '04: Proceedings of the working conference on Advanced visual interfaces*, pages 103–106. ACM Press, 2004.

[40] J. Langner and W. Goebl. Visualizing expressive performance in tempo-loudness space. *Computer Music Journal*, 27:69–83, 2003.

[41] A. Martin and M. Ward. High dimensional brushing for interactive exploration of multivariate data. In *Proceedings IEEE Visualization 1995*, pages 271–278, 1995.

[42] Guerino Mazzola and Oliver Zahorka. The RUBATO performance workstation on NEXTSTEP. pages 102–108, 1994.

[43] Neil P. McAngus Todd. Wavelet analysis of rhythm. 93(4 pt 2):2290, 1993. (abstract).

[44] Reiko Miyazaki, Issei Fujishiro, and Rumi Hiraga. Exploring midi datasets. In *GRAPH '03: Proceedings of the SIGGRAPH 2003 conference on Sketches & applications*, pages 1–1, New York, NY, USA, 2003. ACM Press.

[45] E. Schoonderwaldt, A. Friberg, R. Bresin, and P. Juslin. A system for improving the communication of emotion in music performance by feedback learning. *Journal of the Acoustical Society of America*, 111(5), 2002.

[46] D. Schwarz, N. Orio, and N. Schnell. Robust polyphonic midi score following with hidden markov models. In *Proceedings of the International Computer Music Conference*, Miami, Florida, 2004.

[47] C.E. Seashore. *Psychology of Music*. McGraw-Hill, 1938.

[48] Ben Shneiderman. The eyes have it: A task by data type taxonomy for information visualizations. Technical Report UMCP-CSD CS-TR-3665, College Park, Maryland 20742, U.S.A., 1996.

[49] Ben Shneiderman. Direct manipulation for comprehensible, predictable and controllable user interfaces. In *IUI '97: Proceedings of the 2nd international conference on Intelligent user interfaces*, pages 33–39. ACM Press, 1997.

[50] Ken Steiglitz. *A Digital Signal Processing Primer - with Applications to Digital Audio and Computer Music.* Addison-Wesley Publishing Company, Inc., 2725 Sand Hill Road, Menlo Park, CA, 1995.

[51] Johan Sundberg, Anders Askenfelt, and Lars Frydén. Musical performance: A synthesis-by-rule approach. 7(1):37–43, 1983.

[52] D. Swayne, D. Cook, and A. Buja. Xgobi: Interactive dynamic graphics in the x window system with a link to snap-dragging. In *Proceedings of the ASA section on Statistical Graphics*, pages 1–8. American Statistical Association, 1991.

[53] N. Todd. A model of expressive timing in tonal music. *Music Perception*, 3:33–58, 1985.

[54] George Tzanetakis, Georg Essl, and Perry Cook. Automatic musical genre classification of audio signals. In *"Proceedings International Symposium for Audio Information Retrieval (ISMIR)"*, Princeton, NJ, October 2001.

[55] Brani Vidaković and Peter Müller. Wavelets for kids: A tutorial introduction. Technical report, Duke University, 1991.

[56] M. Ward. Xmdvtool: Integrating multiple methods for visualizing multivariate data. In *Proceedings IEEE Visualization '94*, pages 326–336, 1994.

[57] G. Widmer. Machine discoveries: A few simple, robust local expression principles. *Journal of New Music Research*, 31:37–50, 2002.

[58] Gerhard Widmer and Werner Goebl. Computational models of expressive music performance: The state of the art. *Journal of New Music Research*, 33(3), 2004.

[59] E. Zwicker and H. Fastl. *Psychoacoustics: Facts and Models.* Springer, 1999.