Visualization Research Center of the University of Stuttgart (VISUS)

Masterarbeit

Visual MIDI Data Comparison

Christian Schierle

Course of Study:	Softwaretechnik
Examiner:	Jun.-Prof. Dr. Michael Sedlmair
Supervisor:	Frank Heyen, M.Sc.
Commenced:	May 26, 2020
Completed:	November 26, 2020

Abstract

We propose a system that supports the visualization and visual comparison of MIDI files. As a series of abstract time-related events, MIDI data poses a special set of challenges to the design of visual representations. Based on a set of target user groups and corresponding tasks, we propose a concept and implementation of several visualizations. Multiple MIDI files can be viewed at once in list of cards, each holding a summary of the corresponding file. Our visualization system provides various visualizations for the contents of individual MIDI files. A heatmap is used to provide an overview of the distribution of notes across the channels. We propose a novel approach to the visualization of the occurrence count for each individual note in a hexagonal grid as an alternative to the more traditional implementation of a stacked bar chart. For the visualization of note sequences, we use a pitch-time chart and investigate the capabilities of an adaptation to the MatrixWave visualization for the representation of music. We further explored designs based on arc diagrams as a possibility to visualize similarities between string representations of sequences. To provide support for the visualization of differences between the contents of two MIDI files, we propose comparative views based on the pitch-time chart and the hexagonal grid. We implemented the designs as a browser application, which we evaluated with a usage scenario. The results show that our system supports the specified target users' tasks, but also reveal some limitations of our concept and implementation.

Kurzfassung

Wir präsentieren ein System, das die Visualisierung und den visuellen Vergleich von MIDI Dateien ermöglicht. MIDI Daten, die aus einer chronologischen Abfolge von Ereignissen bestehen, stellen eine besondere Herausforderung für den Entwurf entsprechender visueller Repräsentationen dar. Basierend auf den Bedürfnissen von Nutzern aus bestimmten Zielgruppen entwickeln wir ein Konzept und eine Implementierung einiger Visualisierungen. Beispielsweise können mehrere MIDI Dateien gleichzeitig mit Hilfe einer Liste angezeigt werden, die aus Informationskarten mit einer kurzen Zusammenfassung der jeweils zugehörigen Datei gebildet wird. Unser Visualisierungssystem stellt einige Möglichkeiten zur Visualisierung des Inhalts einzelner MIDI Dateien bereit. Eine Heatmap wird verwendet, um einen Überblick über die Verteilung von Noten in den MIDI Kanälen zu bieten. Als Alternative zu einer traditionelleren Implementierung eines gestapelten Säulendiagramms stellen wir eine neuartige Visualisierung der Anzahl der Vorkommen jeder Note vor, die auf einer Wabenstruktur aus Sechsecken basiert. Zur Visualisierung von Notensequenzen verwenden wir ein Diagramm, das die Tonhöhe und Dauer der einzelnen Noten darstellt. Außerdem untersuchen wir die Leistungsfähigkeit einer adaptierten MatrixWave Visualisierung zur Darstellung von Musik. Des Weiteren haben wir prototypische Entwürfe zur Visualisierung von Ähnlichkeiten zwischen als Zeichenketten repräsentierten Sequenzen untersucht, die auf sogenannten Arc Diagrams basieren. Um die Darstellung von Unterschieden zwischen den Inhalten zweier MIDI Dateien zu unterstützen, stellen wir auf dem Tonhöhendiagramm und der Wabenstruktur basierende Vergleichsansichten vor. Die Entwürfe wurden in Form einer Webanwendung implementiert und mit Hilfe eines Anwendungsszenarios evaluiert. Die Ergebnisse zeigen, dass das Visualisierungssystem die spezifizierten Nutzerbedürfnisse erfüllt, decken aber auch Schwachstellen in der Konzeption und Implementierung auf.

Contents

List of Figures

Listings

1 Introduction

When working with music in a professional fashion or as an amateur, it is important to understand the parts and overall structure of a piece. Repetitive patterns, changes in tempo, volume, and tonality can determine the possibility of landing a hit in the charts, or provide useful information for an estimation of the piece's complexity to someone learning an instrument. Comprehending complex musical structures simply by ear is a hard task and requires some training [TK04]. Therefore, users need to be supported through other means. Visual data exploration provides a powerful tool for the discovery of otherwise undisclosed relations and patterns within a dataset, sometimes even revealing secret messages [Jaw16] hidden in the data.

Many file formats are available for the storage and exchange of musical data. Some formats are optimized for lossless reproduction of a studio or live recording, while others accept a quality penalty in order to reduce file size. The MIDI [MIDI96] is a file format enabling cooperation between and synchronization of electronic instruments and audio devices. As a unified technical standard, it provides a common ground for communication within complex setups including devices from various manufacturers. The MIDI file format was first described in 1981, became standardized in 1983, and is managed by the MIDI Manufacturers Association (MMA)[1]. It is the most prevalent industry standard and is used in a large variety of scenarios involving the production, analysis, and playback of music. These range from music production itself over the synchronization of components in an audio setup to the planning of automatically controlled light shows for live concerts. We therefore use MIDI data as the basis for visualization design.

Musical data is encoded in the form of events that can be grouped into tracks consisting of 16 channels in the current standard, or an unlimited number of channels with the upcoming MIDI version 2.0 [MIDI20]. The two most relevant event types for this thesis represent information about the start or the end of a note with a certain pitch, so-called Note-On and Note-Off events. Other event types hold instructions, like the aftertouch or pitch bend applied to a note or channel. General metadata like the song title, the track or channel name, or the instrument in use can also be defined by using corresponding global events. Since the data is likely to be scattered across multiple tracks and channels, visual analysis can be challenging and suitable visualization techniques have to be applied. This work aims at providing multiple different visualizations to support the comparison of two or more MIDI tracks. Those tracks could represent different songs, multiple versions of the same song, or a ground truth notation and multiple recordings from instruments. Therefore, we combine previous visualization techniques for individual musical data with other approaches for various types of time series data in order to enable comparison of musical data.

We implemented a visualization system as a platform independent browser application. It supports comparative visualization of two to around 20 MIDI files at the same time. A selection of well-established visual representations for MIDI files is supported to make the system accessible to new

[1] https://www.midi.org/.

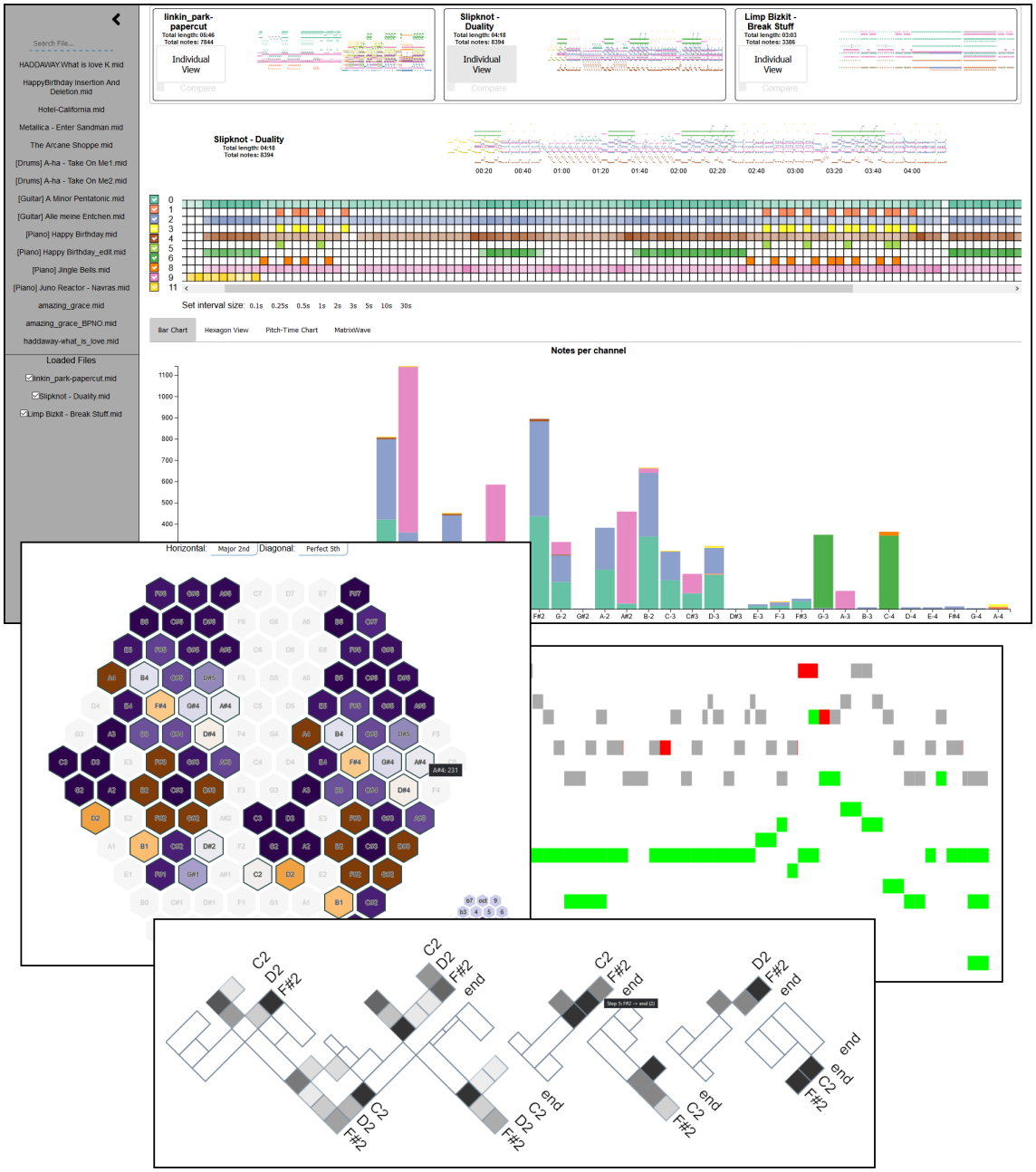

Figure 1.1: An overview of key features of the visualization system presented in this thesis.

users. The focus of this thesis lies on the exploration of various techniques for the visualization of music based on previous works from other research areas. The hexagonal grid layout and a wave-like representation of sequences built from transition matrices are the two most unconventional approaches examined in this thesis. Figure 1.1 shows a preview of the key features of our visualization system. It includes traditional designs as well as our novel approaches towards music visualization based on a hexagonal grid and an adaptation of the MatrixWave visualization.

The rest of this thesis is structured as follows: Chapter 2 presents other work dealing with research on comparative visualization of time series data and pattern recognition within the data. Additionally, an existing taxonomy for comparative visualization is explained here. An overview of the technical background concerning the MIDI standard is provided in Chapter 3. Chapter 4 presents the scope for the conception of the provided visualizations. It describes the scope of the application by outlining suitable user scenarios. The design and visual encodings finding use in the implemented visualizations are discussed in Chapter 5. Chapter 6 provides an overview of the implementation and outlines details on how MIDI data is processed in the system before being displayed. An evaluation of the presented approaches is performed in Chapter 7 under the assumptions set by the previously defined scope. It is followed by a discussion of the findings in Chapter 8. The thesis concludes with an assessment of the gathered insights, and possible areas for improvements and future research in Chapter 9.

2 Related Work

This chapter provides an overview of previous research relevant to the proceedings of this thesis. Various approaches towards the visualization of time series datasets are discussed in Section 2.1, followed by research concerning the segmentation of musical data in Section 2.2. The chapter concludes with an introduction of a set of concepts in Section 2.3 providing context and a common vocabulary for the discussion of the visualizations presented later in this thesis.

2.1 Visualizing Music as a Time Series Dataset

Musical data in the MIDI file format can be treated as time series data consisting of sequences of note events. These events can occur at different, simultaneous, or overlapping points in time throughout a track and have different durations. Various techniques for processing and visualizing such data have been developed and evaluated. Some of them build representations in the form of abstract objects, such as models generated by neural networks [Kal06], or as structured text [HL05]. The focus of this thesis lies upon visual representations instead. Aigner et al. [AMST11] provide an extensive overview of approaches that visualize various types of data in temporal context. A selection of this collection, alongside publications of other researchers, is presented in the following:

Potter et al. [PWB+09] show distribution and uncertainty of ensemble data, representing a collection of multiple time series data sets, generated through weather simulation in a variety of graphs and charts using geographical mappings. Music as a more abstract concept does not come with such inherent mappings, although there have been efforts made towards the visualization of differences in tonal content of music influenced by different traditions and ethnical backgrounds [TE06; Toi05]. This particular aspect is outside the scope of this thesis, but the authors provide an interesting approach to the visualization of large music collections as a hexagon-based map using *Self-Organizing Maps* [KHS12], the design of which is further explored. The visualization of highly multivariate data sets like these in two-dimensional space can become quite challenging, making the feasibility highly dependent on the application area [JME10]. Singh et al. [SZPH16] try to counteract these limitations by making use of three-dimensional visualizations. They claim to have developed a visualization that is equally simple and intuitive as a standard line chart. Three-dimensional views often suffer from visual obstruction and clutter induced by perspective, as can be observed in the efforts of Hiraga et al. [HMF02] and Miyazaki et al. [MFH04] towards a visualization-enhanced MIDI editor.

Highlighting trends and variations in data over time can also provide valuable insights, such as changes to the importance of themes in textual documents following temporal events [HHN00]. Using *MatrixWave* [ZLD+15], event sequences can be visualized and compared. A variation of this visualization is implemented in this thesis as a novel approach to the representation of sequences of MIDI events. Gómez and Bonada [GB05] propose an approach towards visualizing variations

in tonality over the course of a musical piece. Nevertheless, they point out difficulties with key estimation, since regions of stable key cannot be determined reliably, and the estimation requires parameters being fine tuned on a per piece basis. Similarly, the dramatic structure of classical music can be visualized through flowgrams [JN16] based on changes in loudness, tempo, and harmony. These features unfortunately do not apply as well to other music genres. This thesis aims for a more generic visualization approach covering a wide variety of genres.

2.2 Segmentation of Musical Data

The identification of similarities within or between time series datasets is another research area that is important in the context of this thesis. It can help to enhance visualizations with detected relationships between objects, possibly leading to insights being discovered that would otherwise not come to surface during data exploration. Similarities between segments of individual objects provide an estimation of their overall structure, which can be visualized in an arc diagram [Wat02]. Since arc diagrams are also applicable to musical data, this thesis explores a visualization design for individual MIDI files and visual comparison based on this idea. Many approaches exist that try to find approaches towards segmentation of such datasets from different domains. Vrotsou and Nordman [VN20], for example, propose an interactive mining approach for eye-tracking data. Using a sliding window approach to identify patterns, the data is made explorable through an interconnected tree visualization and sequence view. They do not consider simultaneously occurring events though, since human gaze can only focus on one point at a time. The string-based pattern detection within MIDI files performed by Cambouropoulos et al. [CCI+02] can encounter this issue as well. The order in which simultaneous events are inserted into the character sequence should always be the same for different sequences to not interfere with the matching results. When reducing the contents of a musical score to its skyline [RIL09], many of these overlaps can be eliminated before applying similarity measures. Instead of detecting similarity directly, Tanaka and Fujii [TF17] treat the segmentation as an optimization problem. Vectors holding possible motifs of a MIDI file are built in order to find the optimal combination of vectors that covers as many notes of a song as possible. The intended goal is to find the smallest possible building blocks of a song, which are called motifs, but their suggestions leave room for configuration of the optimization parameters.

2.3 Fundamental Concepts in Visual Comparison

While proper visualization of a complex object is difficult in itself, many more challenges arise when comparing multiple objects. Relationships need to be identified between the objects in comparison, which becomes a harder task the higher the complexity or the sheer number of objects is. These are common challenges that have to be dealt with when building comparative visualizations regardless of the types of data at hand.

Gleicher et al. [GAW+11] present a taxonomy that divides comparative designs into categories consisting of the three main types depicted in Figure 2.1. By allowing the combination of these types, their taxonomy specifies a total of six visualization categories. These categories provide

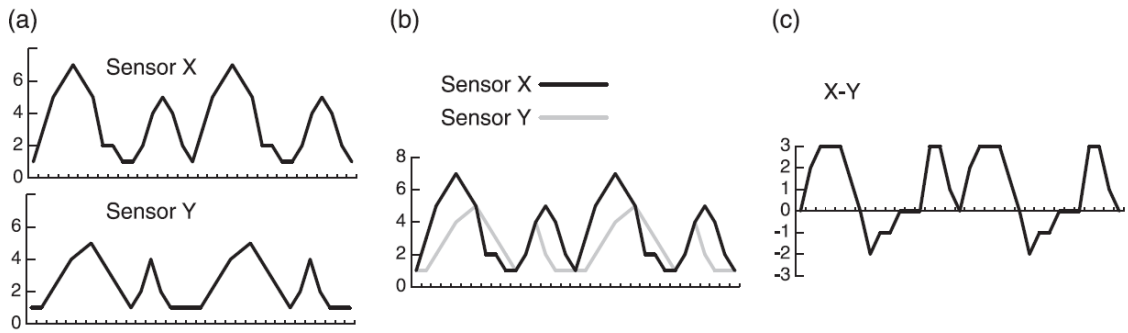

Figure 2.1: The three main categories of comparative visualizations as proposed by Gleicher et al. [GAW+11]: (a) shows an example for juxtaposition, where two graphs of sensor readings are visualized in separate views. (b) shows how overlaying the graphs in the same space through superposition can help to identify correlations between them. In (c) the sensor readings are subtracted from one another and the result is explicitly encoded in a new graph.

context and a common vocabulary for the discussion of comparative visualizations. The categories represent different approaches towards enabling comparison through visualization and are described as follows:

Juxtaposition Designs in this category (Figure 2.1a) use side-by-side views to enable matching of objects. The actual process of identifying connections between the different views is performed by the viewer. Frequently switching focus puts load on the viewer's visual working memory [War10]. Effective comparison is therefore limited by its capabilities [Fra13]. Juxtaposition scales badly as it relies on replication of displays for increased numbers of objects.

Superposition Superposition displays independent objects overlayed within the same space. The proximity helps to identify relationships where spatialization is a key component of the data, or where data is highly similar. Figure 2.1b shows the effects of superposition on the comparison of diagrams. The same space design falls short when the data is dense, resulting in high amounts of clutter in the visualization. Overlaying the data semi-transparently does not provide a scalable solution either, since colors begin to blend. Although applying techniques like color weaving [HKIH07] or attribute blocks [Mil07] can help to mitigate the effects.

Explicit Encoding In contrast to the previous two categories, explicit encoding requires prior knowledge of relations between compared objects, and about how to identify and compute them. The visualization then only shows a newly created object that encapsulates the computed relationships, like the graph depicting the difference of sensor readings in Figure 2.1c. Visualizations in this category can put their main focus on the highlighting of these relationships. Gaining insights on the individual objects only from the aggregated data can be difficult, because the global context is missing. Therefore, explicit encoding is often combined with one of the other visualization types in multiple or hybrid views.

Hybrid Visualizations Next to these three basic categories, Gleicher et al. [GAW+11] also include their combinations in the proposed taxonomy: juxtaposition combined with superposition, juxtaposition combined with explicit encoding, and superposition combined with explicit encoding. These hybrid designs cope with the trade-offs of their individual building blocks, but at the price of higher complexity as well as more potential for visual clutter.

3 Technical Background

This chapter provides an overview of the general principles of the MIDI standard. With the MIDI file format, a typical datatype encountered in the domain of music production, exchange, and playback is described. Key features of the standard are discussed, as well as current developments towards the release of a new version.

3.1 Information Encoding in MIDI

The MIDI file format [MIDI96] provides a standardized encoding to transfer information about music between software and hardware devices. All information is encapsulated as binary data in various types of events holding metadata about the musical piece being transferred, which notes are played at what time, by which instrument, and for how long, as well as technical instructions for devices in the chain of an audio setup. An extensive list of all available event types can be found in the MIDI specification [MIDI96]. A more compact overview is available in the wiki section[1] of the GitHub repository containing the MIDI parser used for the backend server of the presented visualization system (see Chapter 6 for more technical details). The events can be grouped into separate tracks, each consisting of up to 16 channels. This channel limit is lifted with the upcoming MIDI version 2.0 [MIDI20]. Although being a standardized format, MIDI allows for some degrees of freedom as to which events are actually present and what information they hold in a file. For example, information about the instrument represented by a channel is not required. Channel 10 is reserved for percussion instruments by convention, but that is also not enforced by the standard. This uncertainty makes it difficult to rely on the availability of certain features, which also affects how MIDI data can be processed within the scope of this thesis. Section 6.3 provides more details on how MIDI files are processed by the presented system.

3.2 MIDI File Types

The MIDI standard describes three different types of MIDI files:

Type 0 A type 0 MIDI file contains only one track. It holds all the information of the song including the music events as well as metadata, such as the song title.

[1] https://github.com/colxi/midi-parser-js/wiki/MIDI-File-Format-Specifications.

Type 1 There should be at least two tracks in a type 1 MIDI file. The first track, by convention, holds information concerning the entire file, like the song title, time signature, and tempo. The other tracks only contain data specific to the individual track, that is music events and optional metadata such as a title.

Type 2 A type 2 MIDI file allows for a mix of the previous types. It consists of multiple tracks that are independent from one another and do not necessarily occur at the same time. This enables storage of multiple patterns that can be processed individually, or combined to form a song.

3.3 MIDI Version 2.0

A new version of the MIDI standard [MIDI20] has been announced in the beginning of 2020 and is slowly making its way to market. It is backwards compatible with MIDI 1.0, and adds a large variety of improvements, such as bi-directional communication between devices and higher-density transport of information, e.g. through additional channels. This thesis only considers MIDI version 1.0, since it is still well-established and widely used today. Due to its superior feature set, MIDI 2.0 is expected to quickly set foot in the wild and replace the aging previous standard once enough supporting hardware is available [MIDI20].

4 Concept

This chapter discusses the conceptional baseline for the presented visualization system. Following the nested model proposed by Munzner [Mun09], requirements considering users, tasks, and data are discussed. Section 4.1 provides plausible scenarios that highlight different groups of users and their respective tasks when working with musical data. The data itself is described in Section 4.2. A general explanation of the user interface is followed by a discussion of design decisions for the visualizations presented in this thesis. This chapter concludes with an outline of the general workflow when interacting with the visualization system.

4.1 Users and Tasks

The user scenarios presented in this section highlight the need for suitable visualizations. The problem is viewed from different angles, including the one of a music producer, looking for new songs to publish under their label, and a musician's point of view. For the musician, a professional and an amateur are considered. The professional musician is working on their own musical piece, while the amateur is in the process of learning how to play a song on an instrument starting from an existing set of songs. These scenarios construct plausible real-world goals that each of these user types might want to achieve. They also determine the scope of exemplary tasks that are supported by the implementation presented in Chapter 6.

4.1.1 The Professional Music Producer

Imagine a producer working for a music label. To achieve maximum profit from new releases, these should appeal to a large listener base and ideally make it to the charts in multiple locations. Therefore, the producer always keeps track of what kinds of songs are well received by the public. By analyzing and comparing the structure and features of the most profitable songs, a common ground of patterns can be identified that generally do well from a corporate point of view. These patterns can then be combined into a new song that is likely to do similarly well, minimizing the risk of failure for the producer.

4.1.2 The Professional Musician

The interests of a professional musician working on releasing their own piece provide another interesting point of view: The composition of a song is generally an iterative process built of multiple stages [Cop13]. Melody, harmony, lyrics, rhythm, and form have to be developed and combined to form a song. The process of combining and fine-tuning these building blocks is partially driven by trial and error, where certain configurations are implemented in the hope of

resulting in a song matching the musician's requirements. Achieving satisfactory results can require many iterations of tweaking. Being able to compare different versions of a song comes in handy here. Traceability of changes can help to identify a stage, where the development process drifted in a certain direction. The corresponding changes can then be reviewed to allow reflecting the modifications and to allow planning of further adjustments, if necessary.

In the context of this thesis, the professional musician represents the perspective of a composer working on a song, while the producer holds more of a corporate point of view, aiming at reaching market share with publications under their label. This distinction does not necessarily hold up in the real world, since these roles can partially overlap.

4.1.3 The Amateur Musician

Someone picking up a new instrument is generally interested in finding songs to practice with. They often have a collection of singles and albums of their favorite artists and pick a song they enjoy as a learning goal. To evaluate the difficulty of each song, the user can dissect it to estimate the amount of repetition, determine the individual notes and chords, and see the song's tempo. Through comparison with the personal skill set of known notes, chords, and mechanical proficiency, the user can select a song that provides the desired challenge. Songs with highly repetitive patterns and low complexity are supposedly easier to pick up than pieces with little to no repetitions and high amounts of variation.

These collections can also include multiple versions of the same song, be it studio and live recordings, acoustic or cover versions by other artists, or simply different transcriptions for songs where the true notation is unknown. In that case, a learner wants to understand how these versions differ and where they share similarities. With these insights, the user can find and practice their favorite version.

4.2 Data

Many instruments, devices, and software applications support input or output in MIDI and there is an abundance of MIDI tracks available for various genres of music. Information is encoded in the form of time-related events, allowing MIDI files to be treated as time series datasets. This offers a variety of possibilities and challenges for the design of visualizations, since the typical variations of waveform visualizations for file formats based on audio signals (such as MP3) cannot be applied here. More details on the MIDI file format can be found in Chapter 3.

This thesis considers ground truth MIDI files of songs with multiple tracks for different instruments, which means we do not consider MIDI recordings of imperfect performances. Files of various complexities are included, ranging from a few simpler files with small snippets in a single channel to full fledged album versions of popular songs. For an evaluation of the presented visualization system, the song selection includes representatives from a variety of genres.

4.3 User Interface

The user interface, shown in Figure 4.1 and Figure 4.2, follows common design principles for web applications. Rarely used menus are separated and can be hidden to make more room for elements of focus, where most of the user interaction takes places. The interface is divided into three main parts:

Sidebar The sidebar on the left provides a searchable list of MIDI files known to the system. When the user selects a file with a click, it is added to the lower part of the list holding only loaded files. It can be unloaded again with another click. The sidebar can be collapsed via the chevron above the search field.

File Cards The upper section of the user interface contains a list of cards, each of which represent a file loaded into the system. The list is organized into rows of three cards each, which offers the possibility to quickly spot general differences between multiple files. Each card holds an overview of the corresponding file in the form of some metadata that is accompanied by a small pitch-time chart, which acts as a thumbnail preview of the file's content. The Ïndividual View"button in each card provides access to a more detailed visualizations of the corresponding file. Marking the "Compare"checkbox includes it in the aggregated comparative views.

Main Content The largest part of the screen is dedicated to holding the main content. It houses sophisticated versions of the designs introduced in Chapter 5. A file from the list can be viewed individually through a selection of different visualizations. Figure 4.1 shows the individual view of Slipknot's "Duality", the middle entry in the file card list. Furthermore, it can also be compared with another file from the list. Suitable aggregated views are rendered in the main content in that case.

4.4 Site Navigation

The general navigation pattern to access the individual components of the visualization system is depicted in Figure 4.1 and explained in the following: The entry point to the system is an empty slate with a sidebar holding a searchable list of available files in the backend database. By clicking an entry in the list, it is added to the loaded files and a new file card is rendered and added to the list at the top of the page. The sidebar with available and loaded files can be collapsed, if more screen space is required for the main content. Clicking the button labeled "Individual View", or on the pitch-time chart leads the user to a detailed individual view that is rendered in the main content area. The tabs in the rendered view allow to switch between different visualizations for the selected file. Above the tabs, an overview of the file's channels is provided, showing the distribution of their note events as a heatmap (see Section 5.2). The checkboxes allow to add or remove the information related to the respective channel from the rendered visualizations.

Figure 4.1: The workflow to access visualizations of individual MIDI files: Clicking on an available file in the sidebar (1) loads it into the system and creates a card for it. A detailed individual view can be accessed by clicking the corresponding button or the pitch-time chart (2). The tabs (3) allow to switch between different individual visualizations.

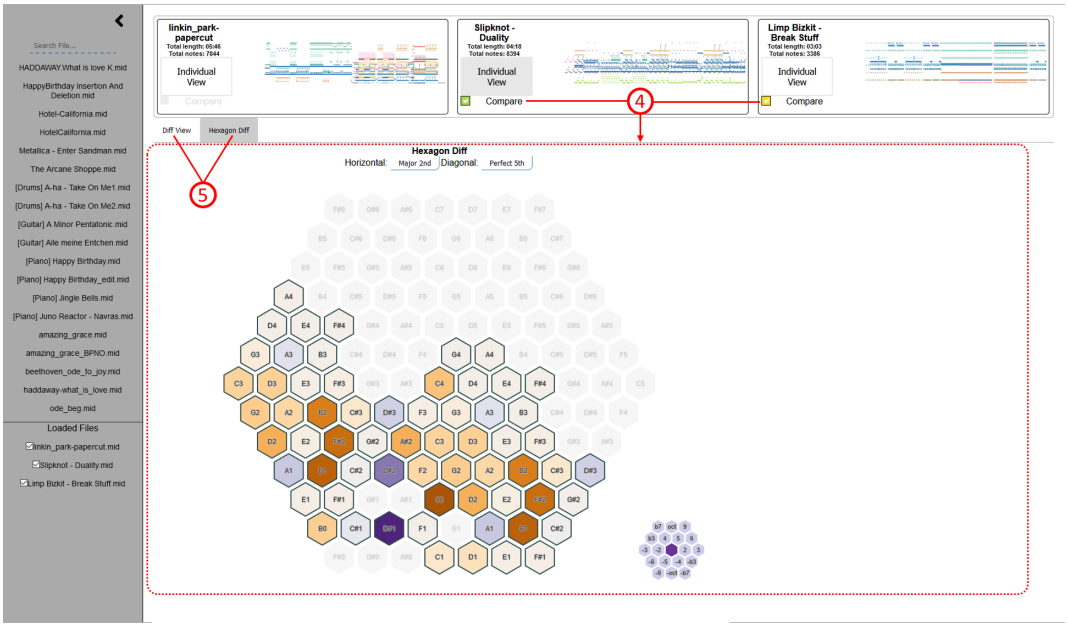

Figure 4.2: The workflow to access comparative visualizations for MIDI files: Instead of or after accessing an individual view (Figure 4.1 (2)), entries in the file card list can be marked for comparison (4). The visualizations available for the selected files can be navigated via the corresponding tabs (5).

To access aggregated views showing direct a direct comparison of MIDI files, the user needs to select the corresponding file cards by marking their “Compare“ checkbox (see Figure 4.2 (4)). If an individual view is currently rendered, an additional button appears, that also leads to the aggregated views. The user can switch between the available views by navigating to the corresponding tab.

5 Visualization Design

This chapter provides an overview of visualization designs and drafts that we developed in the process of this thesis. Information encodings in use are discussed, as well as the reasoning behind design decisions. An evaluation of the desings' performance is conducted in Chapter 7 based on the scope defined in Chapter 4.

5.1 Channel Color Scheme

MIDI files in the scope of this thesis can contain up to 16 channels, which poses a significant challenge to the design of comparative visualizations. For the visualizations presented here, information related to MIDI channels is encoded using the color scheme provided in Figure 5.1. A unique color from the scheme is assigned to each channel and is used consistently in each visualization. The scheme is a combination of two predefined categorical color schemes in the visualization framework D3[1] We concatenate the color schemes *schemeSet1*(Figure 5.1 a) and *schemeSet2*(Figure 5.1 b) from the *d3-scale-chromatic*[2] module.

Relying solely on color to encode large amounts of data is generally not recommended [War10]. Nevertheless, the limitations of the given design space do not allow for many alternatives: applying patterns or using alternative shapes is not viable for complex charts, such as pitch-time charts consisting of many small elements. The presented color scheme is not optimal, since especially the shades of yellow are sometimes difficult to discern, but it offers enough versatility for the applications presented in this thesis. Through the consistent application of channel colors throughout all building blocks of the visualization system, the user can quickly understand what color is associated to which channel.

Figure 5.1: Unique colors to assign to MIDI channels. This color scheme is formed by combining *schemeSet1*(a) and *schemeSet2*(b) from the *d3-scale-chromatic* module. These 17 unique colors are applied consistently to all visualizations involving information separated across MIDI channels.

[1] https://d3js.org/.
[2] https://github.com/d3/d3-scale-chromatic/.

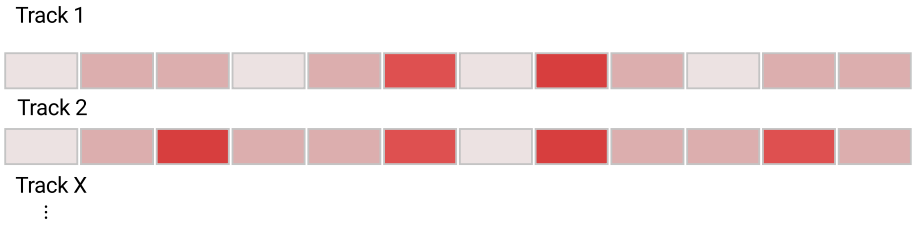

Figure 5.2: Sketch of a heatmap showing the density of note events per track. The track's sections are colored in red where the density is high and in white where it is low.

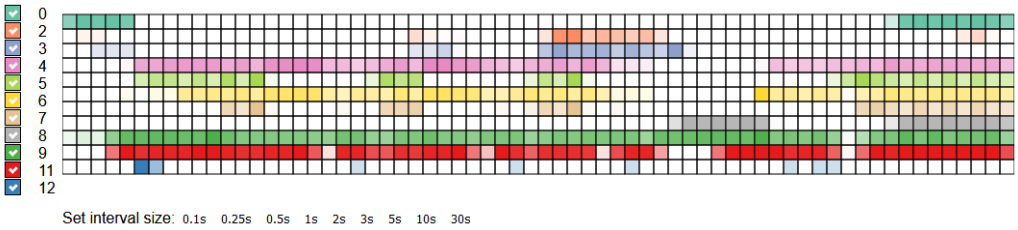

Figure 5.3: Channel sections visualizing the distribution of note events in each channel of Metallica's "Enter Sandman". The most populated channels can clearly be discerned from the relatively unoccupied rest.

5.2 Channel Heatmap

Figure 5.2 shows an early prototype of a heatmap visualization. Each track of a MIDI file is split up into sections of equal length. The sections are grouped in chronological order. The saturation of each section's color depends on the number of note events within a section compared to the global maximum of the corresponding track. Instead of using the uniform color scheme from the sketch in Figure 5.2, the color scheme from Section 5.1 is applied in the final version of this visualization.

Figure 5.3 shows, how aligning the heatmaps of all tracks enables the user to directly compare their note density. This alignment allows to discern sparsely populated tracks from tracks of high complexity. Sparsely populated tracks can indicate a solo for a specific instrument. Continuous entries in the heatmap often represent repetitive rhythm-related parts, such as the bass or rhythm guitar tracks as well as drum patterns. In the individual view containers, each track of the heatmap represents a channel of the corresponding MIDI file. Using the attached checkboxes, notes belonging to the respective channel can be included or hidden from the visualizations. The time interval determining the notes contained within each section can be adjusted with the buttons below the heatmap.

5.3 Pitch-Time Chart

One of the most commonly used visualizations for MIDI files is a line plot showing the occurring MIDI events over time. Figure 5.4 shows how the open source audio editor Audacity[3] displays a MIDI file with the melody of “Happy Birthday“ stored in the first of 16 channels. The lines, or rectangular areas in this case, match the starting time and duration of notes in the file. They are color coded to match the channel they belong to. Although this thesis explores new visualization types for musical data, this traditional representation is also included in the system. Users with experience in the field should immediately recognize this type of chart, which provides an intuitive point of entry to the system.

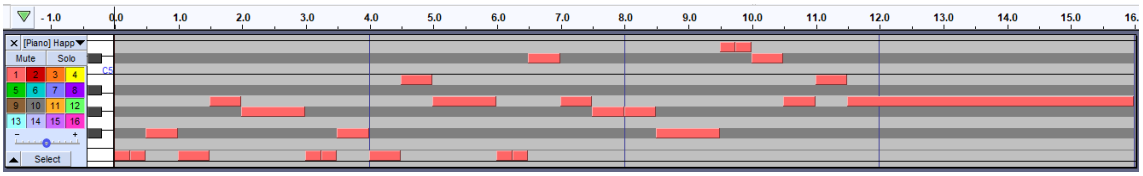

Figure 5.4: A MIDI file as it is displayed in the open source audio editor Audacity.

Figure 5.5 shows a visualization of “Happy Birthday“ in the system presented in this thesis. Instead of using a piano representation to identify the notes, an axis with the actual note names is provided on the left. The time axis is located at the bottom of the chart. In an earlier version of the chart, the height of the note rectangles was determined by the applied velocity. Depending on the given data, this lead to confusing visual clutter due to large variations in the individual heights. With a fixed

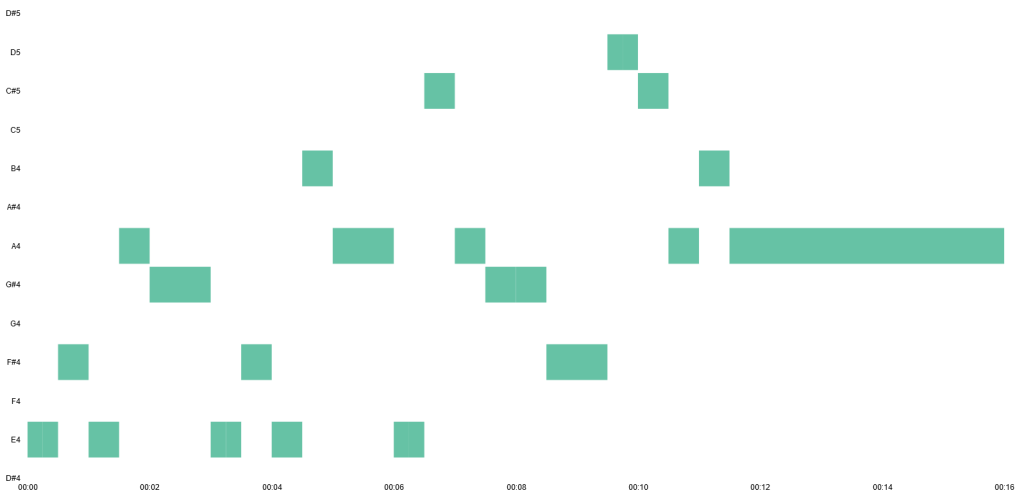

Figure 5.5: “Happy Birthday“ visualized as a pitch-time chart. The y-axis is labeled with the note names, while the x-axis provides temporal context. Each bar represents a note from the MIDI file and is colored according to the channel it belongs to. Only one channel is filled with notes in this example.

[3] https://www.audacityteam.org/.

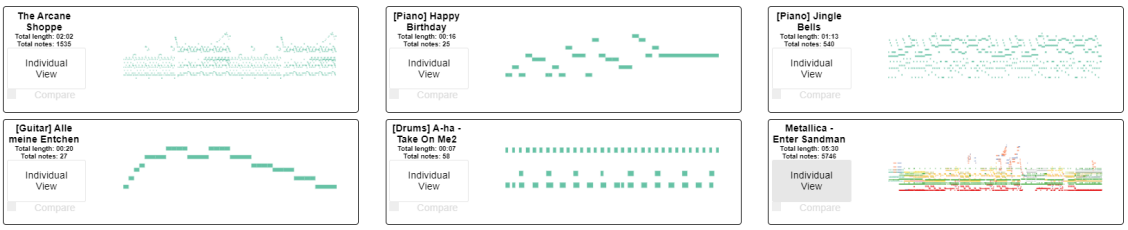

Figure 5.6: A list of file cards providing a quick overview of each MIDI file consisting of a minified pitch-time chart and general metadata. Interaction elements lead to more detailed individual, or aggregated views

height, the chart is much cleaner and easier to follow. Notes are colored according to the channel they belong to using the color scheme from Section 5.1. The simple example in Figure 5.5 only has one channel filled with notes, so only one color is applied. More complex examples can be found in Chapter 7.

5.4 File Cards

The cards shown in Figure 5.6 were designed early on and remained a central building block throughout the entire project. They contain some metadata about the corresponding MIDI file, like the filename, the length of the stored song, and its total number of note events. A small version of the Pitch-Time Chart (see Section 5.3) is also included to provide a quick overview of the song's structure. The interaction elements allow the user to jump to a more detailed individual view, or to include the file into the current comparison for aggregated views.

By arranging the cards in a list, a quick comparison of multiple MIDI files is possible. The downsized Pitch-Time Charts allow to easily identify differences in the complexity of files. Metallica's "Enter Sandman" in the bottom right of Figure 5.6 clearly has a higher note density than the German children's song "Alle meine Entchen" at the bottom left. The use of multiple colors for notes from different channels also indicates higher complexity due to notes being spread across multiple channels.

5.5 Stacked Bar Chart

Visualizing distributions can provide valuable insights when working with time series data [PWB+09]. Bar charts or histograms provide simple but effective ways for encoding comparison information [War10]. Differences in the size of bars can easily be identified by the viewer, since all bars start a common zero value. The use of areas adds more visual support to understand relations than a basic scatter plot would provide. This effect can be observed in Figure 5.7 and Figure 5.8. The distribution of notes across note groups, i.e. when omitting octave information, is visualized in Figure 5.7.

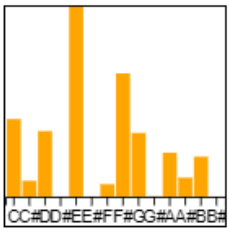

Figure 5.7: Bar chart showing the distribution of note groups within a MIDI file, with omitted octave information.

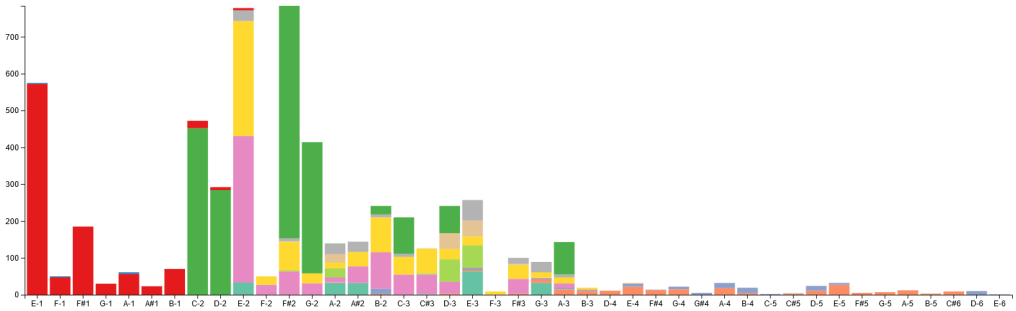

Figure 5.8: A stacked bar chart showing the distribution of note events across multiple channels of a MIDI file. Each stack is colored according to the channel color scheme in Section 5.1. The y-axis represents the total count per note. The note name is depicted on the x-axis.

The stacked bar chart in Figure 5.8 additionally makes use of the color scheme from Section 5.1 to split the individual bars into chunks representing the channel information. While this allows to convey more information to the user at once, the readability of the stacked bar chart is reduced due to the increased visual clutter. Sections belonging to the same channel not starting at the same zero value also add to the issues with readability, and make exact comparison difficult. On the other hand, highly populated channels can quickly be identified as well as outliers, like the red bars in Figure 5.8 representing notes for percussion instruments. Discerning small differences between two sparsely populated channels becomes increasingly difficult, when channels with high note counts are displayed simultaneously. The orange portion of the bar at B-2 is hardly visible at all. These limitations are tackled with the application of channel filters in the corresponding view containers.

5.6 Hexagon View

Another variation of the heatmap approach from Section 5.2 uses hexagonal shapes. This approach was inspired by the hexagon-based map visualization of large music collections by Toiviainen and Eerola [TE06] and Toiviainen [Toi05]. Figure 5.9 shows how the information is conveyed to the user in a design sketch of a similar arrangement of hexagons. Each hexagon represents a MIDI note. Using a diverging color scale, the hexagons are colored corresponding to their number of occurrences within a song. High counts are mapped to the upper end of the scale, and low counts to the lower end, while a neutral color represents the mean of occurrence counts.

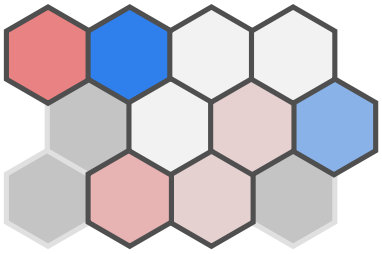

Figure 5.9: Sketch of a heatmap using hexagonal shapes. Red indicates high note counts, blue shows low note counts, and white represents the mean. Notes that do not occur in the file are greyed out.

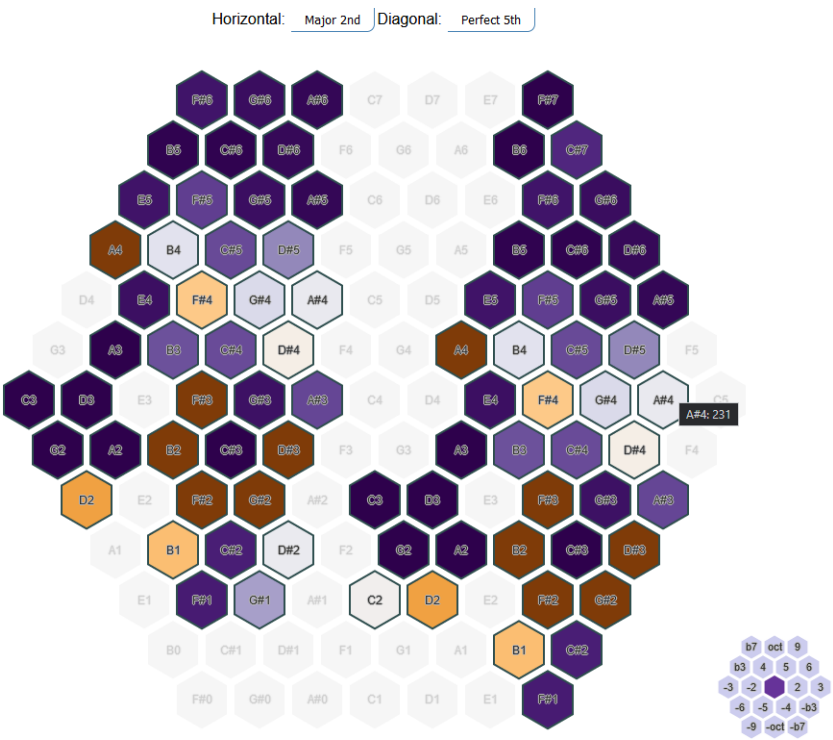

Figure 5.10: "What Is Love" by Haddaway represented in the hexagonal grid layout. Each hexagon represents a note and is colored according to its occurrence count within the song. The neutral color white represents the average, while purple and orange stand for deviations above and below average respectively. Greyed out notes do not appear in the song. The layout can be rearranged by changing the horizontal and diagonal intervals using the dropdown menus at the top. The distance between a note and its neighbors is shown in the legend.

The implementation of the final version is based on Asturiano's *Musical Hexagons*[4]. In this relative layout, each hexagon represents a note between C0 and G9. The center of the visualization depicted in Figure 5.10 is always set to the middle C (C4), which is commonly used as a point of reference in music. The other notes are placed around the center according to the selected interval between horizontally and diagonally adjacent notes. This ensures that the distance between intervals is consistent along each dimension of the visualization. Aligning the hexagons into such a regular grid pattern, forms a unique overall form for each song. We replaced the red and blue color scheme from the sketch in Figure 5.2 with an interpolation between purple and orange. Both schemes respect impairments induced by color blindness. The red and blue scheme worked well on small sketches, but was not as visually pleasant when working with actual data. Hovering over a hexagon reveals the occurrence count for the corresponding note. The legend in the bottom right of Figure 5.10 provides an overview of the intervals. The filled hexagon represents the point of reference, while each surrounding hexagon contains the interval between the point of reference and itself. Using the dropdown menus at the top allows to change the intervals for each axis, which rearranges the grid accordingly.

5.7 Arc Diagram

We explored visualization designs based on arc diagrams [Wat02]. They use a concept similar to the visualization of chronological sections in the channel heatmap from Figure 5.3. We apply the Levenshtein distance to estimate the similarity between string representations of the sections. The sections in Figure 5.11 are represented as circles with a radius matching their individual length. Arcs connect similar sections. Their thickness reflects the calculated similarity, meaning a large line width stands for a low Levenshtein distance and vice-versa.

Since we already applied the same basic concept as with the channel heatmap, a similar layout for arc diagrams supporting multiple tracks was explored as well. It provided the baseline for the prototype in Figure 5.12. The segments of multiple files are arranged in lines with arcs indicating similarities between sections of adjacent files. We already showed with the channel heatmap from Figure 5.3 that up to 16 MIDI tracks can be compared in a compact visualization. With a similar layout that accounts for the additional arcs between tracks, we expect about 30 tracks to be comparable in an implementation of the prototype in Figure 5.12. Positioning the boxes closer to each other could help to use the available space more efficiently. Loss of information due to overlapping arcs should be prevented then.

Unfortunately, the similarity measures did not perform consistently for different data types The required fine tuning per instance could be resolved by providing corresponding user settings in an individual view. We did not consider this a viable option in a comparative scenario and did not further pursue an implementation of either of these approaches.

[4] https://bl.ocks.org/vasturiano/e70e14483fe01eb0a3ea7d1d46a30571.

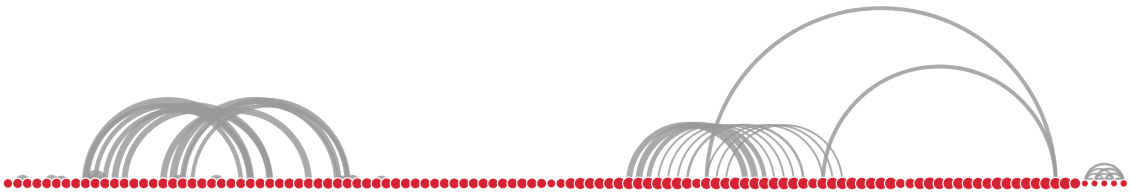

Figure 5.11: An arc diagram for "Papercut" by Linkin Park. Differently sized circles represent sections of different lengths. Similar sections are connected with an arc reflecting their calculated similarity value.

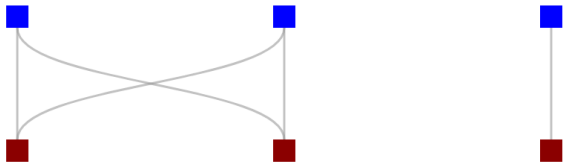

Figure 5.12: A prototype for a comparative visualization of multiple MIDI tracks using arc diagrams to indicate pairwise similarities between parts.

5.8 MatrixWave

MatrixWave [ZLD+15] provides an interesting approach towards the visualization of ordered sequences. Due to the reasons discussed in Section 8.2.2, the comparative visualization proposed by the authors was not considered useful in our implementation. However, we draw inspiration from their ideas for an implementation of an adapted design for sequences of an individual MIDI file. The visualization depicted in Figure 5.13 consists of a series of transition matrices that are aligned in a zig-zag pattern. An entry in a transition matrix is a link from a starting note to a target note in the following matrix. Each individual matrix visualization consists of three parts: a set of bars, an area with squares colored in greyscale, and a legend. The bars represent the starting notes for the links in the current step of the sequence. Their length is determined by how many links start with the given note in total. Each square represents a link in the transition matrix. It is placed at the intersection of the bars corresponding to the link's start and target. Its occurrence count is represented by the assigned greyscale color. High counts result in a square with dark shades, low counts yield light shades. Hovering over a square reveals the start, target, and count of the link and the current step. The legend holds the names of the starting notes for each step and form the axis labels for the coordinate system shared by each pair of adjacent matrices.

5.9 Pitch-Time Diff

The pitch-time chart from Section 5.3 only shows note events of an individual song. By adjusting the color coding, the same visualization can also be used to display differences between two songs. Following the color scheme of common file comparison tools, such as *git diff*, differences can be explicitly encoded. Additional notes are represented as green bars, while removed notes are colored red. Grey as a neutral color indicates the absence of change, meaning that there is no difference at the corresponding note between both versions. This approach to the encoding of changes between

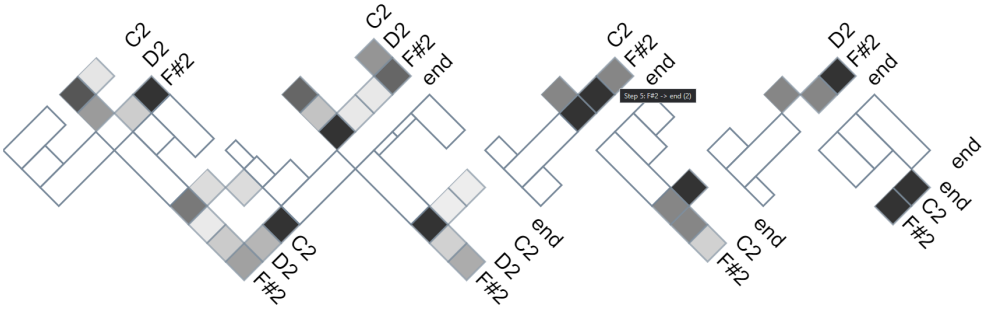

Figure 5.13: A MatrixWave representation of a snippet from "Take On Me" by A-ha. Sequences are read from left to right as a zig-zag pattern of links between notes. The white bars represent the occurrence count of each note per sequence step. Each square depicts a link between two notes and is colored in a greyscale matching its frequency (dark=high, light=low). Hovering over a square reveals start and end of a link as well as its occurrence count.

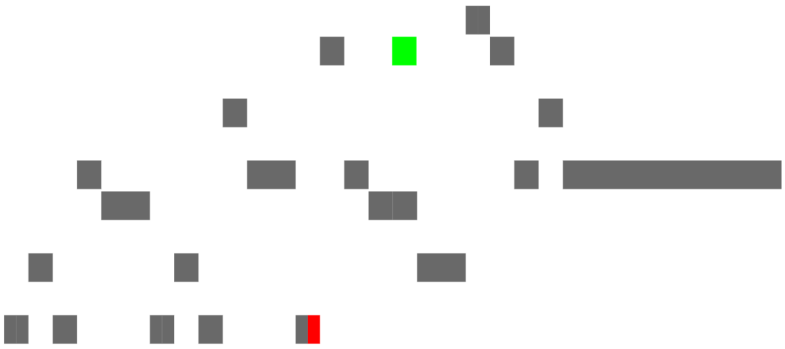

Figure 5.14: Differences between two songs visualized in a pitch-time chart. The grey bars represent notes occurring in both of the songs. Similar to the color scheme of Git's diff tool, additional notes in the second song are highlighted in green, while removed notes are highlighted in red.

two versions of a file is well-established in version control systems widely used during software development. We think, the provided insights on changes between versions are valuable for music and could be used to track different directions that a user explored during the production of a song.

In Figure 5.14, two versions of the melody of "Happy Birthday" are combined into the same visualization using superposition. Between the two versions of the melody in Figure 5.14, an instance of E4 has been removed (highlighted in red), while a C#5 was added (highlighted in green).

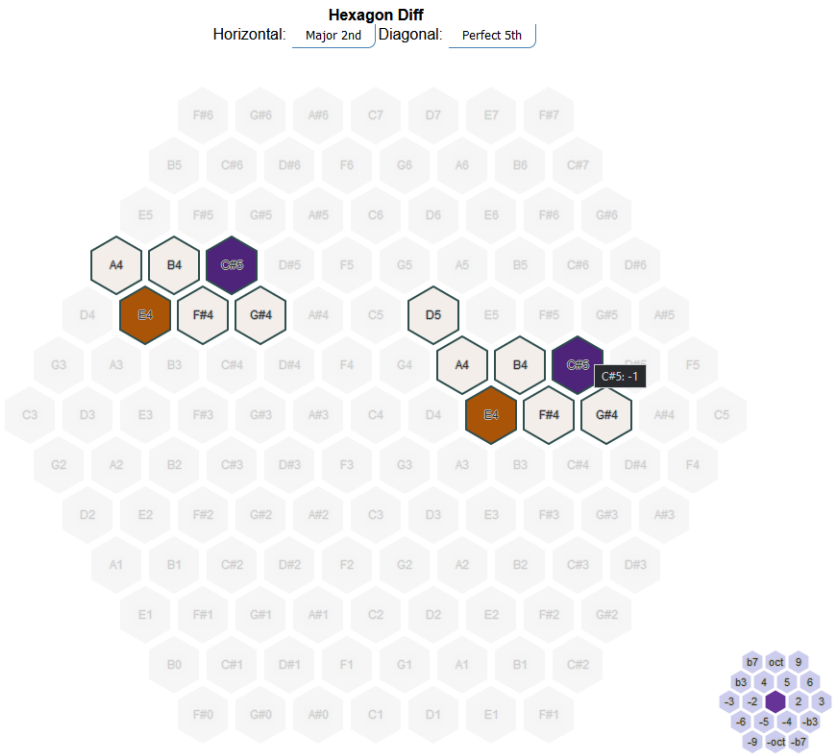

Figure 5.15: Differences between two songs visualized using a hexagonal grid layout. Each hexagon represents a note between C0 and G9. Notes occurring in both songs are colored in neutral white. Differences are encoded via interpolation on a diverging color scale from purple to orange, which respects color blind users. Any notes that do not occur in either of the songs are greyed out. The hexagons are arranged around the fixed center C4 according to the selected interval between horizontally and diagonally adjacent notes. The legend at the bottom right shows the distance between a note in the grid and its neighbors.

5.10 Hexagon Diff

Another method to show differences between two MIDI files in direct comparison is provided with a modification of the hexagonal grid layout introduced in Section 5.6. In this relative layout, each hexagon represents a note between C0 and G9. The center of the visualization is always set to C4, since the middle C is commonly used as a point of reference in music. The other notes are placed around the center according to the selected interval between horizontally and diagonally adjacent notes. This ensures that the distance between intervals is consistent along each dimension of the visualization. Changing the intervals using the dropdown menus at the top automatically updates the layout accordingly. Notes that do not occur in either of the compared songs are greyed out. We apply the same color scale as in Section 5.6 to encode differences in occurrence counts per note. However, their meaning is adapted for this comparative view: the neutral color white indicates notes with an equal amount of occurrences in both of the comparison candidates. Hexagons colored in shades of orange reflect a higher occurrence count of the corresponding note in the first file of the comparison. Purple indicates a bias towards the second file.

In the example shown in Figure 5.15, two versions of the melody of “Happy Birthday“ are compared. The largely neutral coloring suggests that there are little to no differences in the number of notes in each version. In fact, the first version only has one additional instance of E4 than the other, which is reflected by the corresponding hexagon being colored in purple. The second version has an additional C#5, which is represented by the corresponding hexagon being orange. Larger variations in the number, especially when multiple different notes are added or removed, are encoded via different shades of orange and purple respectively.

6 Implementation

The presented visualization system is implemented as a browser application using *JavaScript* to bypass compatibility issues between operating systems. Supported browsers include Google Chrome and Mozilla Firefox. Others are not guaranteed to run the application without issues. The target screen resolution is 1920×1080 pixels, which corresponds to a Full HD screen. Higher resolutions benefit from a larger area available to visualizations, while lower ones are likely to experience issues due to misalignment of page contents.

The remainder of this chapter provides an overview of the technologies in use and the general project structure. It concludes with providing details on the conversion of MIDI data into the systems data format in Section 6.3.

6.1 Toolkit

JavaScript is one of the most popular programming languages for the development of browser applications. After some experiences in other projects using TypeScript, JavaScript seemed like a great choice, while offering the side goal of familiarizing more with the programming language. The following selection of tools and frameworks was used to support the implementation of the visualization system as a browser application:

React React[1] provides a popular framework for the development of interfaces using JavaScript. Views can be divided into simple components. Using states, changes in data are detected and corresponding views are only updated when needed. React version 16.13.1 is used in this thesis. It helps to keep the individual visualizations loosely coupled, such that changes in a single view do not necessarily affect the entire system. The state awareness of React also allows for efficient DOM and view updates.

Node.js Node.js[2] provides a runtime environment to host network applications based on JavaScript. The implementation of the presented visualization system is run using version 14.4.0 of Node.js.

[1] https://reactjs.org/.
[2] https://www.nodejs.org/.

D3 The implemented visualizations are powered by the extensive feature set of D3[3]. This popular library provides capabilities to draw shapes and graphs, as well as for accessing and manipulating the DOM. Integrating D3 with React did posed quite a challenge, since their features partially interfere with each other. Generally, D3 is mostly used to calculate properties of DOM elements, such as size, position, and color, while updating views is handled by React.

6.2 Project Structure

The presented visualization system is implemented as a browser application to avoid compatibility issues between different operating systems. The JavaScript project consists of two main parts: a backend server providing the data for visualizations, and the frontend, where the data is processed and visualized. Both are run as separate Node.js applications and communicate with each other by exchanging requests using JavaScript's Fetch API.

The frontend fetches MIDI files stored in the ground truth data folder in the backend. These files are initially processed in the backend, where all information from the MIDI file is converted into a JSON representation. In the frontend, the MIDI tracks and events are further processed and converted into a JavaScript object containing the MIDI notes alongside some metadata. More details on the conversion between the backend and frontend objects are provided in Section 6.3. The various visualizations are implemented as individual React components that are fed with the parsed data. Wrappers for comparison and individual views of files combine these components into a sophisticated visualization system.

6.3 MIDI Data Conversion

In the backend server, the tracks, events and metadata of MIDI files are converted into a JSON representation. Listing 6.1 shows the contents of a MIDI file as they are returned by the parser. For the purpose of explaining the most relevant parts to the reader, the MIDI content in the example has been simplified to its basic components. The top level of the JSON hierarchy contains general metadata, namely the MIDI format type (see Chapter 3), the number of tracks in the file, and the "timeDivision" value. The track portion of the JSON object holds an array of MIDI events per track. A MIDI event is attached to one of 16 channels and is of a certain type that determines the rest of its contents. The most relevant events for the visualizations are the "Note-On" (type 9) and "Note-Off" (type 8) events. An complete list of event types is available in the MIDI specification [MIDI96], and wiki section[4] in the GitHub repository containing the MIDI parser used for the backend server contains a more compact overview. The "deltaTime" property of a note event specifies the point in time, it occurs on relative to the previous event. The first value in the "data" portion defines the note pitch, while the second value stands for the velocity, i.e. how fast the corresponding piano key is pressed. A single note is represented through the combination of a "Note-On" event with the corresponding pitch and the next matching "Note-Off" event occurring afterwards.

[3] https://d3js.org/.
[4] https://github.com/colxi/midi-parser-js/wiki/MIDI-File-Format-Specifications.

```
{
    formatType: 1,
    timeDivision: 960,              // used to decode the delta times into real time
    tracks: 2,
    track: [                        // an array holding all tracks
        {
            event: [...]            // an array holding all events of this track
        },
        {
            event: [
                {deltaTime: 0, type: 9, channel: 0, data: [45, 95]},    // a Note-On event
                {deltaTime: 960, type: 8, channel: 0, data: [45, 95]},  // a Note-Off event
                {deltaTime: 0, type: 9, channel: 0, data: [48, 95]},
                {deltaTime: 960, type: 8, channel: 0, data: [48, 95]},
                ...                 // more events
            ]
        }
    ]
}
```

Listing 6.1: Raw structure of a simplified MIDI file as it is provided by the backend.

Using the “timeDivision“ value and the total length of the track, the time difference of such combinations is converted into real time by the frontend. The resulting note object holds its start and end times as a value in seconds relative to the start of the track, and the pitch, velocity, and channel information. An array of such note objects is saved per part, i.e. per track or channel depending on the file format type, alongside some metadata derived from other MIDI events. Each part is assigned a name and all the parts together with some general metadata form the “midiData“ of the MIDI file. The JavaScript representation of a file is completed by combining the “midiData“ with the filename into a JavaScript object.

```
{
    name: "A Simple Sample.mid",
    midiData: {
        beatType: 4,
        beats: 4,
        bpm: 120,
        instruments: [...],         // only filled, if the instruments are known
        totalTime: 12,
        partNames: ["Track 1"],
        parts: [                    // an array holding all parts/tracks
            {
                beatType: 4,
                beats: 4,
                bpm: 120,
                measureLinePositions: [2, 4, 6, 8, 10, 12],
                noteObjs: [         // note objects derived from Note-On/-Off events
                    {pitch: 45, start: 0, velocity: 95, channel: 0, end: 0.5},
                    {pitch: 48, start: 0.5, velocity: 95, channel: 0, end: 1},
                    ...
                ],
                tempoChanges: [     // used to determine bpm and beat
                    {tick: 0, tempo: 120}
                ],
                totalTime: 12,
            }
        ]
    }
}
```

Listing 6.2: Structure of a parsed MIDI file encoded as a JavaScript object that is further processed in the frontend.

7 Evaluation

This chapter starts with an introduction to the applied evaluation methodology and an explanation for why it was chosen. We perform an analysis of the presented visualization system using real-world data from freely accessible MIDI databases in the remainder of this chapter. The general assumptions about users, data and tasks discussed in Chapter 4 also apply to the evaluation on the following pages. Our visualizations are fed with different types of MIDI data including music from various genres. The findings are presented alongside documentation of the involved aspects of the visualization system. A discussion of the findings and identified limitations follows in Chapter 8.

7.1 Methodology

Ideally, the implemented visualization system should have been tested and evaluated by domain experts. Due to the circumstances revolving around the COVID-19 pandemic of 2020, the possibilities for personal contacts, meetings, and access to laboratories in the university buildings were strongly limited. Conducting a case study involving extensive feedback from real users was therefore not a feasible approach to gather feedback. The evaluation is instead performed without external users in the form of a usage scenario, as it is defined by Sedlmair et al. [SMM12]. A usage scenario is a common validation tool for design studies, but the findings gathered do not provide the same quality of feedback that a case study with real users would yield. While real-world data is in use, the evaluation in a usage scenario is generally based only on a small selection of constructed tasks inspired by real-world problems and performed by the developers of the evaluated system. The results are therefore inherently biased and cover a narrower scope of the domain.

The assumptions about users, tasks, and data within the scope of this thesis, which were introduced in Chapter 4, do not only apply to the implementation, but also provide the baseline for the evaluation of the visualization system. An analysis of the interaction with the system in different stages of the specified user tasks is performed using real-world data.

7.2 Viewing an Individual MIDI File

A variety of visualizations for the contents of an individual MIDI file is available in the visualization system. Through a combination of these visualizations, insights about the structure of a file and trends in its content can be gathered. The example presented in Figure 7.1 presents an example for how the visualizations work together. The general metadata and a small version of the pitch-time chart represent the gist for Slipknot's "Duality" and provide context for the other visualizations below. A closer look at the structure of the song is provided by the channel heatmap. It shows an overview of the distribution of notes appearing across all channels. Channels that are populated for

Figure 7.1: Various visualization approaches are combined in the detailed view for a single MIDI file. Slipknot's "Duality" is selected here and highlighted in the file card list. Metadata and the small pitch-time chart provide context for the visualizations below. Interactive tabs allow to switch between different visual representations.

most of the total duration hold notes related to the rhythm of the song, such as rhythm and bass guitar as well as the drums. Sparsely populated channels hold special events, like the one-time appearance of a piano during the intro in channel eleven.

Visualizations for note distributions and sequences can be accessed by navigating to the corresponding tab below. For example, the stacked bar chart reflects the exact distribution of notes and their occurrence for each channel. The dominance of the rhythm elements is visible here as well. Nevertheless, the few high pitches placed throughout the song in channel three are clearly audible when listening to the song and play a crucial role for its characteristics. Their importance to the listening experience does not come to show in the visualizations. The focus here lies more on highlighting those note combinations that make up most of the song. An amateur musician, as defined in Section 4.1.3, can use this information to find the requirements for being able to learn and play their favorite song.

The MatrixWave representation is intended to provide a novel and intuitive visualization of sequences within a MIDI file. The high promising expectations to this approach unfortunately do not hold up well when working with real-world data. It is already difficult to follow the relatively simple sequences shown in Figure 5.13. With larger amounts of possible start and target notes per link, the visualization becomes increasingly cluttered. Such an example is provided in Figure 7.2. A possible approach to improve the results of the MatrixWave implementation is discussed in Chapter 8.

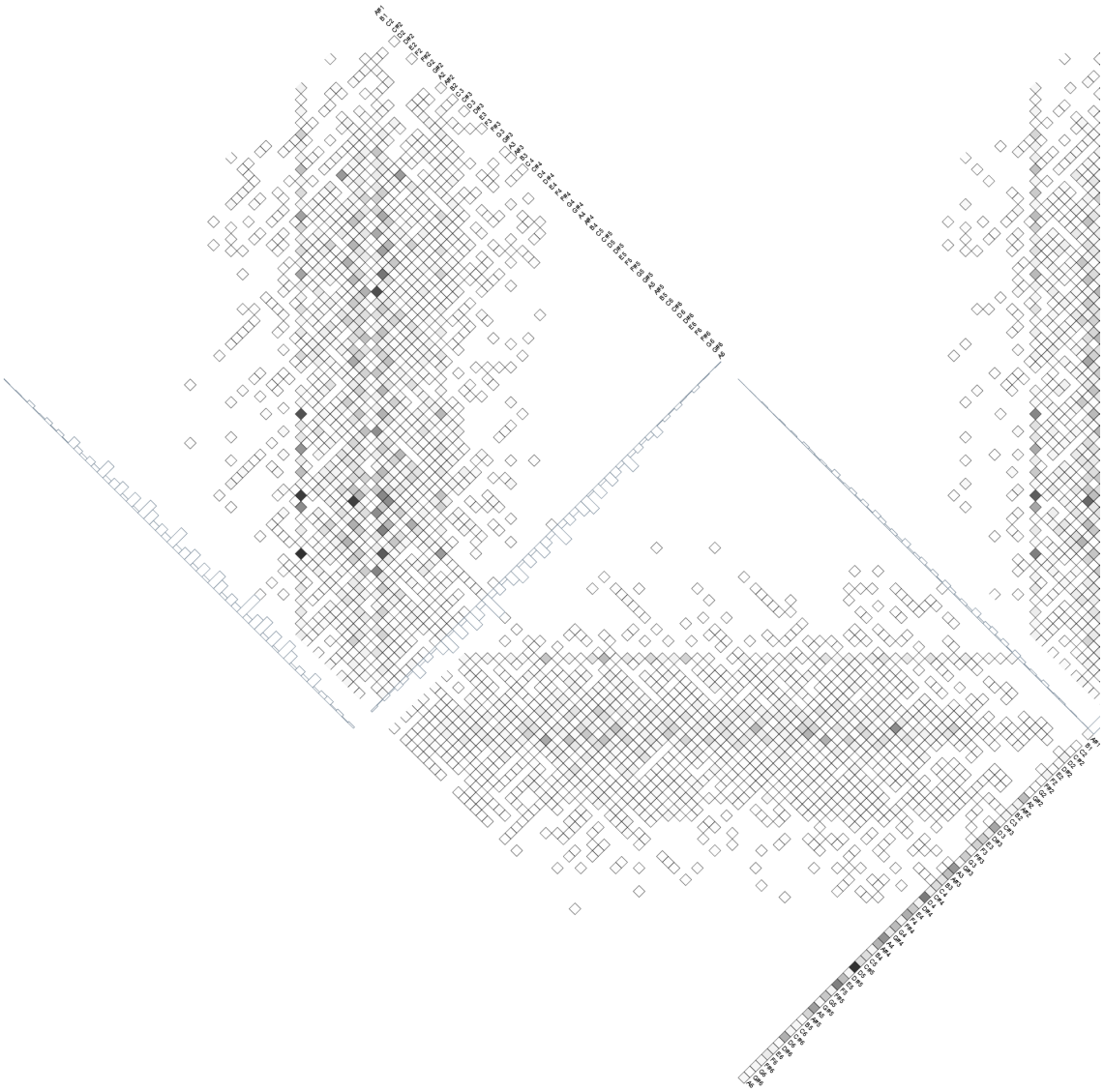

Figure 7.2: This snippet only shows the first two and part of the third step of the MatrixWave chart for Beethoven's "9th Symphony". With 60 different possible notes at the start of a sequence and a similar amount of possible targets, this visualization becomes very cluttered and hard to read.

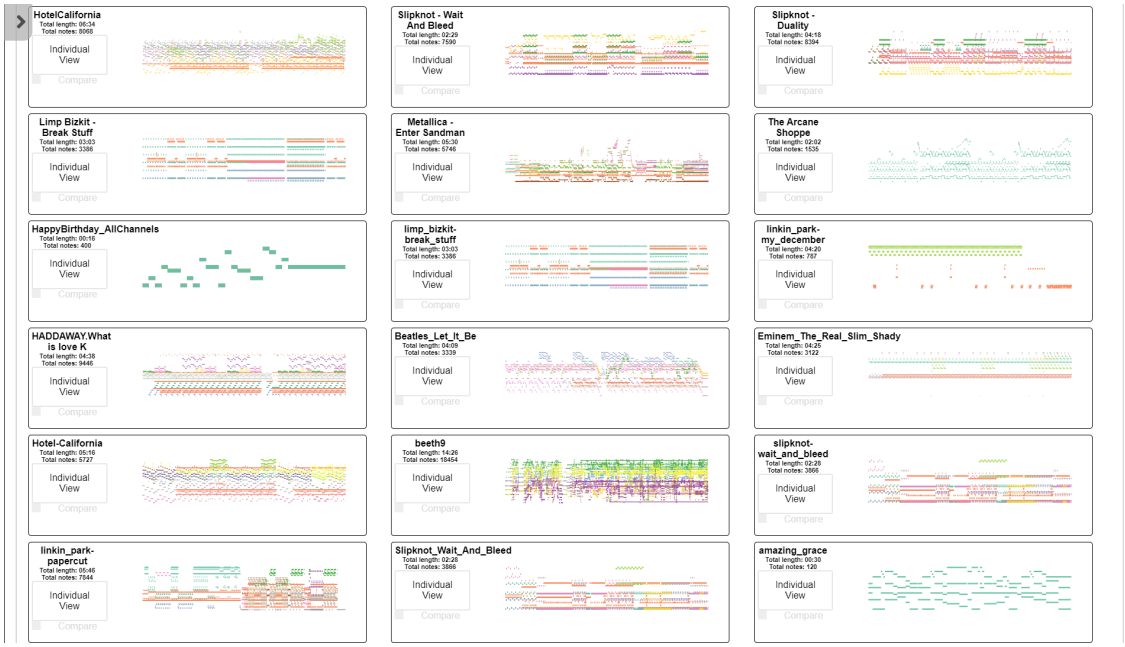

Figure 7.3: The file card list allows comparison of multiple MIDI files at once. Up to 18 files can be viewed at once on a 1920 × 1080 pixel screen. Including even more files in the comparison is supported through vertical scrolling.

7.3 Viewing Multiple MIDI Files

This section provides examples for how the visualization system supports the display of multiple MIDI files at the same time. The user scenarios defined in Section 4.1 are used as a reference for the achievable tasks and goals. Key aspects of the visualizations and possible observations are pointed out for each of the examples, using real-world data.

7.3.1 Comparison of Multiple MIDI files

The visualization system supports comparison of multiple MIDI files simultaneously via the list of file cards. Up to 18 files arranged in rows of three fit onto a screen with a resolution of 1920×1080 pixels at once. Figure 7.3 provides an example showing a full screen list of MIDI files from different genres and with varying complexity. Vertical scrolling allows to compare even more files in the list. Higher screen resolutions benefit from additional rows being visible without the need to scroll.

While the list does not allow for a detailed comparison, it provides an enhanced overview of a collection of MIDI files. The downsized pitch-time chart on each card provides an impression of the gist for the corresponding file. Small snippets or files with only a single channel can easily be discerned from more complex list entries. The total length and note count represent valuable metadata about each file in the list and provide additional means to verify the complexity suggested by the gist. An exact comparison is not viable, since individual notes are hardly visible in the small charts and identifying relations in comparative visualizations using juxtaposition is generally difficult [GAW+11]. The more detailed views for pairwise comparison can be accessed by marking

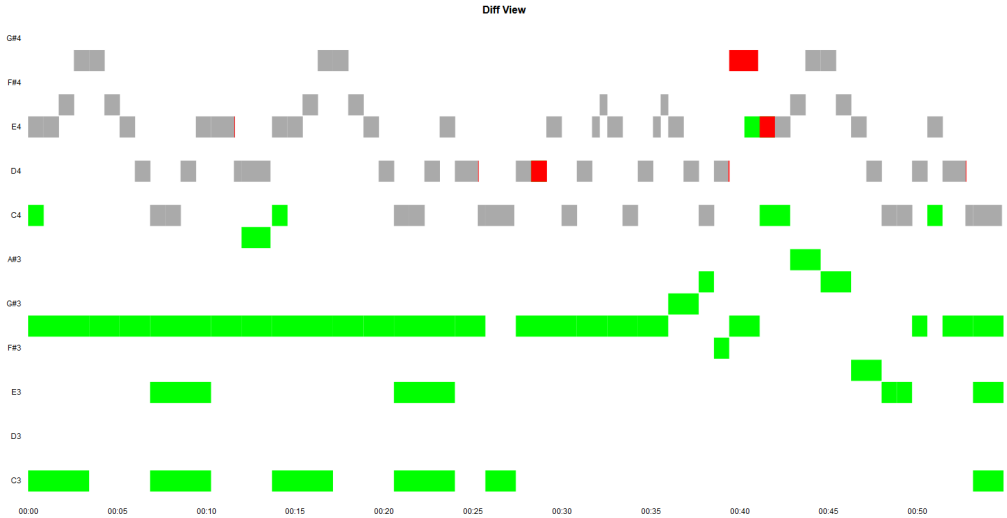

Figure 7.4: This pitch-time diff view compares two versions of the piano melody for Beethoven's "Ode to Joy". The main part of the melody (grey) remains the same except for a few minor changes. A large set of low notes is added (bottom, green).

the respective "Compare" checkboxes of the files in question. With the overview provided by the list of file cards, users can identify musical pieces of a certain level of complexity within a collection of MIDI files. As pointed out in Section 4.1.3, this can help someone learning an instrument to pick a song to practice.

7.3.2 Direct Comparison of Two Versions

The pitch-time diff view introduced in Section 5.9 allows to spot differences between two MIDI files. Figure 7.4 shows, how this can help a user identify changes between two versions of the same piece of music. In this case, a one-handed piano version of Beethoven's "Ode to Joy" is compared with a two-handed one. Except for a few slight variations, the grey colored main melody remains unchanged. Most of the added green notes are an octave lower than the previously existing melody, suggesting that they are played with the second hand. The additional set of notes is also clearly visible in the Hexagon Diff view depicted in Figure 7.5, which compares the same two versions. Many notes are colored in shades of orange there, since they only appear in the two-handed version. Note how there is only one dark purple hexagon. This reflects the additional G4 appearing in the one-handed version, which also stands out as a red rectangle in Figure 7.4.

As this example shows, the system supports the user in identifying changes between different versions of a song. It provides a representation of the corresponding use case of reviewing different stages in the process of writing a song, as defined in Section 4.1.2. After having established the melody, the two-handed version represents efforts towards adding additional segments making the song more interesting and unique. Through superposition of the two versions, minor adjustments to the melody can be identified in the visualization as well. Additionally, an amateur musician might be able to use the information gathered from the comparison to decide, which version of the melody to learn next based on their current skill level.

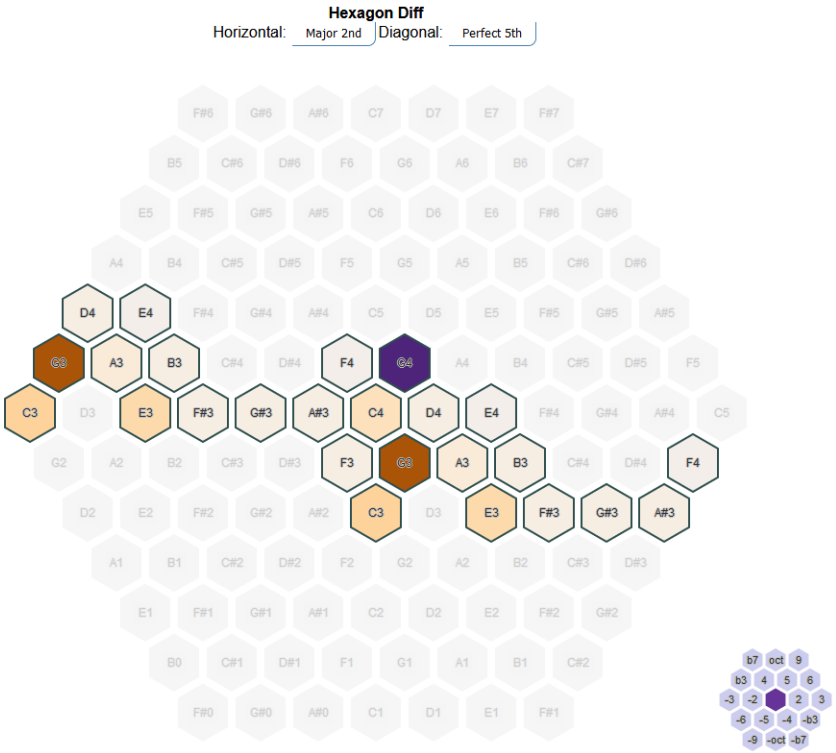

Figure 7.5: The Hexagon diff view for two versions of Beethoven's "Ode to Joy" reveals an additional set of notes (colored in orange) in the two-handed version. The purple G4 hexagon is the only difference biased towards the one-handed version and is therefore very dark.

8 Limitations and Discussion

The evaluation of our visualization system demonstrates that it can support the user groups described in Section 4.1. While comparative visualization of multiple MIDI files within the scope of this thesis is supported, the evaluation also revealed instances where the designs fail to convey information as intended. This section brings attention to the identified limitations of the visualization system. An explanation is provided alongside an appropriate showcase. We discuss the findings and suggest possible approaches towards overcoming the limitations.

8.1 Conceptual Limitations

This section points out limitations induced by the conceptual factors involved in the system design process. The chosen level of granularity, at which MIDI files are processed, plays an important role here. It impacts possibilities for visualization design as well as the capabilities of segmentation techniques.

8.1.1 Main Focus on Individual MIDI Notes

The visualization system presented in this thesis is mainly built around processing MIDI files as a series of individual note events occurring over time. Considering such small entities of very fine granularity and high quantity greatly limits the design space for visualizations. Summarizing chunks of note events for further processing can provide additional possibilities for meaningful visual representations. These chunks could include chords, beats, or entire verses. The channel heatmap from Section 5.2 provides an example for how larger entities can yield additional insights. Figure 5.12 also shows the potential benefits for comparative visualization of a multitude of MIDI files using Arc Diagrams. Some of the other limitations described in this chapter might be mitigated by further pursuing the idea of larger musical constructs as well.

8.1.2 Insufficient Segmentation

Segmentation of music still poses a rather ill-explored research area. In the process of this thesis, two approaches towards segmentation have been explored: applying string matching algorithms to music [CCI+02] for the detection of similarities, as well as treating segmentation as a set partitioning problem [TF17] aiming for optimal coverage of a piece by vectors with motifs. Segmentation should allow the detection of common patterns within and between MIDI files, which could be used to build more powerful visualizations. Both of the explored solutions encountered issues related to the processing of individual note events. The string matching algorithms struggled with correct interpretation of simultaneously occurring events, while the efficiency of set partitioning was strongly

impacted by the large number of possible combinations of notes. Introducing an intermediate step, where chords are extracted from the MIDI file first, would likely improve the quality and efficiency of the results.

With better performing segmentation algorithms in place, we also suggest to revisit and improve on the efforts towards a visualization of MIDI of single or multiple files using Arc Diagrams. This also applies to the MatrixWave implementation, which is discussed separately in Section 8.2.2. These visualizations could benefit greatly from better segmentation techniques resulting in better readability through reduced visual clutter.

8.2 Limitations to Visualizations

The evaluation of the implemented system shows the capabilities of the applied visual representations in the context desired by the defined user tasks. It also revealed instances, where the designs do not perform as intended. We point out and discuss these limitations in the following.

8.2.1 Time Shift in Pitch-Time Diff

While the pitch-time difference view performs well for MIDI files that are in sync, its capabilities fall short when the files are shifted in time. This can happen, when one file comes with a longer intro or interlude than the other. The negative effects of such a time shift are clearly visible in Figure 8.1, where two versions of “Hotel California“ by The Eagles are compared with each other. According to the color coding in red and green, the two versions appear to be completely distinct. In reality, there is a strong similarity in their general patterns. For reference, a comparison using the hexagon view (see Figure 8.2) manages to represent the large overlap in terms of MIDI notes correctly. The change detection for the pitch-time sequence is performed starting from a common zero position, the start of the MIDI file. If a time shift is present, changes and similarities are not detected correctly. A manual or automatic alignment mechanism would be able to solve that issue. Developing a robust solution for this problem surpasses the scope of this thesis and is left to future work.

8.2.2 High Complexity of the MatrixWave Visualization

A adaptation of the MatrixWave representation was implemented and explored as a novel approach to music visualization. While the idea of visualizing sequences of note events as a wave-like stream of transition matrices, the actual result of the implementation turned out to be not particularly intuitive. The visualization is only suitable for files with a small number of notes and hard to understand. Figure 7.2, showing a snippet from the MatrixWave chart for Beethoven’s “9th Symphony“, provides an example for the high amount of visual clutter that can be induced by a high number of notes. The original MatrixWave [ZLD+15] uses a second concentric square per link to represent its count instead of applying a greyscale color directly. Using concentric squares would introduce even more lines to the already overly cluttered layout, which would further reduce its readability. The issue rather lies in the sheer amount of notes within a sequence. Therefore, exploring the effects of visualizing sequences of larger musical constructs, such as chords, might yield better results.

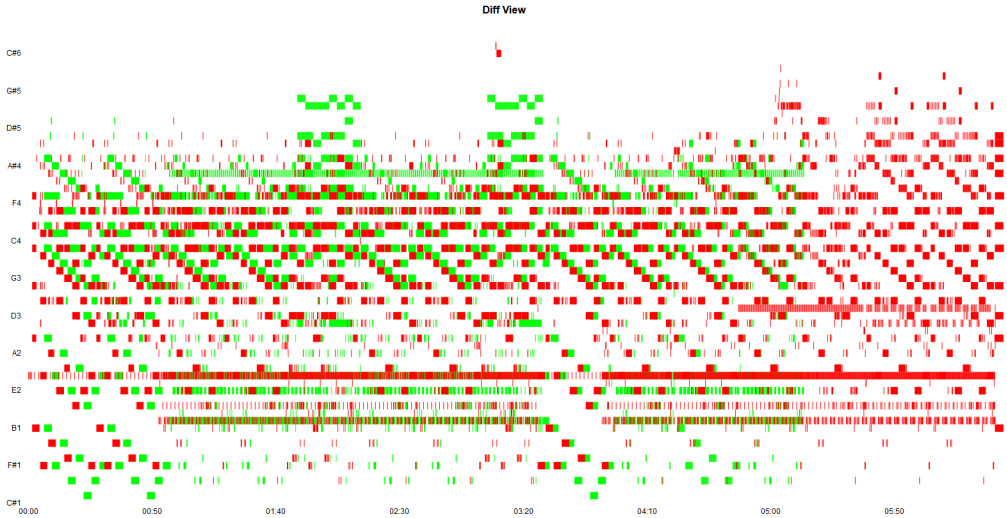

Figure 8.1: Comparing these two versions of "Hotel California" shows a major limitation of the pitch-time diff view: it does not cope well with comparison candidates, that are shifted in time.

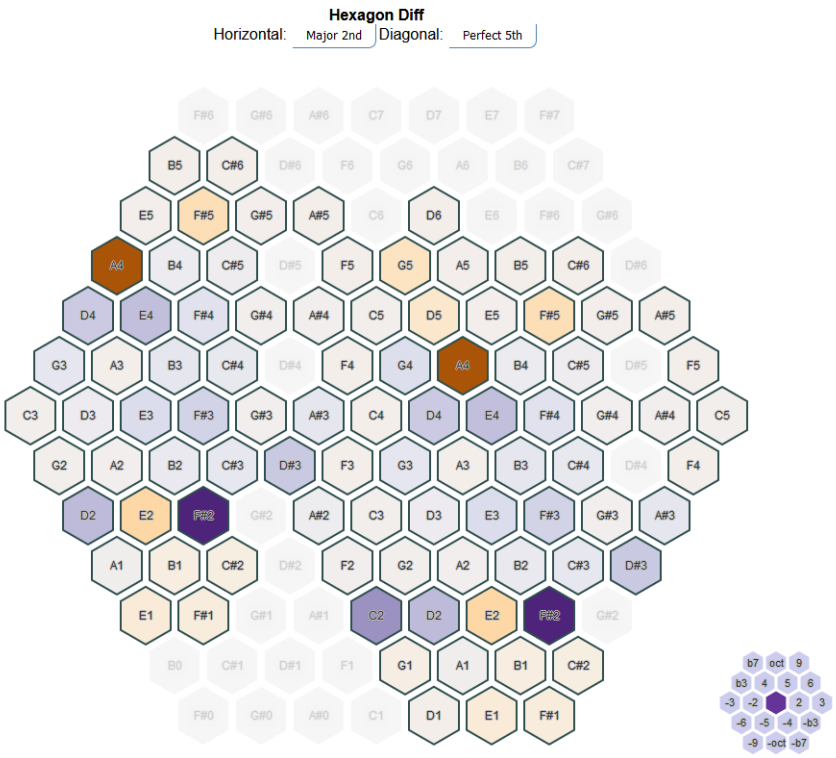

Figure 8.2: The comparison of these two versions of "Hotel California" in a hexagonal grid correctly shows only minor variances between the two.

8.2.3 Inefficient Use of Space in the Hexagon View

For some types of data, the implementation of the Hexagon View does not use the screen space efficiently. The visualization is suitable for files that consist of many different notes, as Figure 5.10 and Figure 8.2 show. However, showing files without much variation in the contained notes results in greyed out hexagons occupying a large portion of the screen space. This effect can be observed in the example shown in Figure 5.15.

The optimization of the spatial distribution posed a challenge to the design of the hexagonal grid layout. A trade-off was made between usability, understandability and the use of meaningful shapes. The initial layout arranges the hexagons into the shape of a large hexagon to fit the general design theme. Users can adjust the layout according to their preferences. All available notes that fit into the generated shape are always included, even if they do not appear in the file on display. This ensures a consistent layout for each setting to reduce the cognitive load of unexpected results for users when working with the visualization regularly. While we consider this manually adjustable layout an appropriate solution, it still offers room for further optimization of the user experience.

8.2.4 Detailed Comparison Only for File Pairs

The adjusted encoings for pairwise comparative visualizations also do not support including more files. Since they are based on techniques common to other types of difference tools, they are inherently only applicable to pairwise comparisons. If an adaptation for more comparison candidates is found, it would likely not hold up well due to the fine granularity of the visualizations in use.

Considering larger musical constructs, as pointed out in Section 8.1.1, could provide a solution to this issue. The presented prototype for a comparative visualization of a multitude of MIDI files (see Figure 5.12) shows how this consideration opens up new design possibilities.

8.2.5 Full-length Comparison Only

The available comparative visualizations are applied to the entire length of the compared candidates. Gathering detailed insights on certain overlapping parts is difficult, especially when comparing very long MIDI files. We therefore suggest to extend the visualizations with tools to select specific time intervals for each member of the comparison. Only those notes that are contained in the selected interval would be included in the visualizations. The implemented visualizations would benefit greatly from this change, making them applicable in a larger variety of scenarios.

8.2.6 Limited Selection of Comparative Visualizations

The implemented visualization system only provides a limited selection of tools for the comparison of MIDI files. As discussed above, the idea of a comparison using Arc Diagrams was dropped due to limited segmentation capabilities. Another possible visualization for the comparison of two sequences is presented in the original paper introducing the MatrixWave [ZLD+15] visualization. The authors present a variation that can be used to compare two sequences with each other by applying a diverging color scale similar to the one used in the Hexagon View. As shown in Section 8.2.2, the

implemented MatrixWave representation already struggles to convey the information of a single MIDI file to the user in a meaningful way. Therefore, we found that increasing the complexity by adding even more encodings did not seem reasonable.

9 Conclusion and Future Work

We presented a system that supports visualization and visual comparison of MIDI files. A set of tasks that are typically performed by different user groups served as the definition of the scope for this project. We designed several approaches to the visualization of MIDI files using techniques and principles established by other researchers. The designs were implemented in a web-based browser application and with respect to the defined scope. Contents of a MIDI file can be visualized in a pitch-time chart. An experimental MatrixWave representation provides an alternative visualization for note sequences within a file. The channel heatmap and a stacked bar chart provide overviews of note distributions. We implemented a visualization showing the distribution in a hexagonal grid layout as a novel solution. For the comparison of two MIDI files, we provide a difference view based on a pitch-time chart and another using the hexagonal grid. More files can be compared with less detail using the list of file overview cards.

The system was evaluated in the form of a usage scenario without involving external users. we analyzed the implemented designs with real-world MIDI data and discussed the findings. Our system supports the specified user tasks and allows to visualize up to 18 MIDI files on a 1920×1080 pixel screen at once. We discussed the strengths and weaknesses of the visualization designs revealed by the evaluation and suggested possible improvements. Treating MIDI files as sequences of individual notes in the conception of the visualization system limited the design space for visual representations. It also affected the performance of techniques for the segmentation of music negatively. The evaluation also revealed problems related to temporal and spatial alignment of contents in a few visualizations.

In the future, the effects of using musical constructs of higher granularity instead of individual MIDI notes on the visualization system should be analyzed. Corresponding conceptual changes and adaptations to the visualization designs could mitigate some of the identified limitations. In particular, we suggest to further investigate the following ideas:

More sophisticated segmentation techniques using chords or chord sequences instead of individual notes could be developed. We expect the MatrixWave representation to benefit greatly from the consideration of larger musical constructs in the visualized sequences. The use of Arc Diagrams would become a more reasonable option for visualization designs as well.

Temporal alignment of files in comparative visualizations would improve the performance of corresponding designs. The alignment could be handled via user interaction to manually align the comparison candidates. An automatic solution could be based on aligning corresponding file segments tolerating more deviation possibilities. We suggest the use of a hybrid approach of combining such an algorithm with user interaction to achieve optimal results.

A similar approach could be applied to enable the selection of specific time intervals for each MIDI file involved in a comparison. We expect the possibility for manual adjustments combined with automatic highlighting of points of interests in the form of identified segments to greatly increase the versatility of comparative visualizations.

With suitable segmentation techniques in use, we also suggest to revisit the prototypes based on Arc Diagrams. For an individual MIDI file, further insights on its overall structure could be gathered. Our design for a comparative visualization using Arc Diagrams to highlight similarities between heatmap representations of MIDI files was only dropped, because it offered no real benefit without proper segmentation. With this issue fixed, we think it provides a suitable solution for visual comparison of more than 20 MIDI files.

The visualization of entire music collections could reveal more interesting relations between various types of music. Toiviainen and Eerola [TE06] already showed how characteristics about the tonal content for folk music of different geographical origin can be identified through visualization. We think that with a similar approach, significant features could be extracted for each genre in a collection of music, such as the predominant keys, chord sequences, and instruments in use.

Bibliography

[AMST11] W. Aigner, S. Miksch, H. Schumann, C. Tominski. *Visualization of Time-Oriented Data*. en. Human–Computer Interaction Series (HCIS). London: Springer, 2011. ISBN: 9780857290786. DOI: 10.1007/978-0-85729-079-3 (cit. on p. 15).

[CCI+02] E. Cambouropoulos, M. Crochemore, C. Iliopoulos, L. Mouchard, Y. Pinzon. "Algorithms For Computing Approximate Repetitions In Musical Sequences". In: *Int. Jour. Computer Mathematics (IJCM)*. 2002, pp. 1135–1148. DOI: 10.1080/00207160213939 (cit. on pp. 16, 49).

[Cop13] D. Cope. "Song Production: The Marriage Between Composition and Audio Production". en. In: *From Demo to Delivery*. Ed. by R. Hepworth-Sawyer. New York: Routledge, 2013. Chap. 1, pp. 3–20. ISBN: 9780080928432. DOI: 10.4324/9780080928432 (cit. on p. 21).

[Fra13] S. L. Franconeri. "The Nature and Status of Visual Resources". In: *Oxford Handb. Cognitive Psychology*. Oxford University Press, 2013, pp. 147–162. DOI: 10.1093/oxfordhb/9780195376746.013.0010 (cit. on p. 17).

[GAW+11] M. Gleicher, D. Albers, R. Walker, I. Jusufi, C. D. Hansen, J. C. Roberts. "Visual comparison for information visualization". In: *ACM Information Visualization (IVIS)*. 2011, pp. 289–309. DOI: 10.1177/1473871611416549 (cit. on pp. 16–18, 46).

[GB05] E. Gómez, J. Bonada. "Tonality Visualization of Polyphonic audio". In: *Int. Computer Music Conf. (ICMC)*. 2005. URL: http://hdl.handle.net/2027/spo.bbp2372.2005.174 (cit. on p. 15).

[HHN00] S. Havre, B. Hetzler, L. Nowell. "ThemeRiver: visualizing theme changes over time". In: *IEEE Symp. Information Visualization (InfoVis)*. 2000, pp. 115–123. DOI: 10.1109/INFVIS.2000.885098 (cit. on p. 15).

[HKIH07] H. Hagh-Shenas, S. Kim, V. Interrante, C. Healey. "Weaving Versus Blending: a quantitative assessment of the information carrying capacities of two alternative methods for conveying multivariate data with color." In: *IEEE Trans. Visualization and Computer Graphics (TVCG)*. 2007, pp. 1270–1277. DOI: 10.1109/TVCG.2007.70623 (cit. on p. 17).

[HL05] G. Haus, M. Longari. "A Multi-Layered, Time-Based Music Description Approach Based on XML". In: *Computer Music Jour. (COMJ)*. 2005, pp. 70–85. DOI: 10.1162/COMJ.2005.29.1.70. URL: https://muse.jhu.edu/article/180167 (cit. on p. 15).

[HMF02] R. Hiraga, R. Mizaki, I. Fujishiro. "Performance Visualization: A New Challenge to Music through Visualization". In: *ACM Multimedia*. 2002, pp. 239–242. DOI: 10.1145/641007.641054 (cit. on p. 15).

[Jaw16] U. Jawad. *Pentagrams and 666 appear in DOOM's soundtrack in a spectrogram*. en. May 2016. URL: https://www.neowin.net/news/pentagrams-and-666-appear-in-dooms-soundtrack-in-a-spectrogram (visited on 05/05/2020) (cit. on p. 11).

[JME10] W. Javed, B. McDonnel, N. Elmqvist. "Graphical Perception of Multiple Time Series". In: *IEEE Trans. Visualization and Computer Graphics (TVCG)*. 2010, pp. 927–934. DOI: 10.1109/TVCG.2010.162 (cit. on p. 15).

[JN16] D. Jeong, J. Nam. "Visualizing Music in its Entirety using Acoustic Features: Music Flowgram". In: *Techn. for Music Notation and Representation (TENOR)*. 2016, pp. 25–32. ISBN: 9780993146114. URL: https://mac.kaist.ac.kr/pubs/JeongNam-tenor2016.pdf (cit. on p. 16).

[Kal06] A. Kalos. "Modeling MIDI Music as Multivariate Time Series". In: *IEEE Trans. Evolutionary Computation (TEVC)*. 2006, pp. 2058–2064. DOI: 10.1109/CEC.2006.1688560 (cit. on p. 15).

[KHS12] T. Kohonen, T. S. Huang, M. R. Schroeder. *Self-Organizing Maps*. English. Berlin, Heidelberg: Springer, 2012. ISBN: 9783540679219 (cit. on p. 15).

[MFH04] R. Miyazaki, I. Fujishiro, R. Hiraga. "comp-i: A System for Visual Exploration and Editing of MIDI Datasets". In: *Int. Computer Music Conf. (ICMC)*. 2004. URL: http://hdl.handle.net/2027/spo.bbp2372.2004.117 (cit. on p. 15).

[MIDI20] The MIDI Association. *Details about MIDI 2.0™, MIDI-CI, Profiles and Property Exchange*. en. June 2020. URL: https://www.midi.org/midi-articles/details-about-midi-2-0-midi-ci-profiles-and-property-exchange (visited on 10/03/2020) (cit. on pp. 11, 19, 20).

[MIDI96] The MIDI Association. *The Complete MIDI 1.0 Detailed Specification*. en. 1996. URL: https://www.midi.org/specifications-old/item/the-midi-1-0-specification (cit. on pp. 11, 19, 40).

[Mil07] J. R. Miller. "Attribute Blocks: Visualizing Multiple Continuously Defined Attributes". In: *IEEE Computer Graphics and Applications (CG&A)*. 2007, pp. 57–69. DOI: 10.1109/MCG.2007.54 (cit. on p. 17).

[Mun09] T. Munzner. "A Nested Model for Visualization Design and Validation". In: *IEEE Trans. Visualization and Computer Graphics (TVCG)*. 2009, pp. 921–928. DOI: 10.1109/TVCG.2009.111 (cit. on p. 21).

[PWB+09] K. Potter, A. Wilson, P.-T. Bremer, D. Williams, C. Doutriaux, V. Pascucci, C. R. Johnson. "Ensemble-Vis: A Framework for the Statistical Visualization of Ensemble Data". In: *IEEE Int. Conf. Data Mining Workshops (ICDMW)*. 2009, pp. 233–240. DOI: 10.1109/ICDMW.2009.55 (cit. on pp. 15, 30).

[RIL09] D. Rizo, J. Iñesta, K. Lemström. "Ensemble of state-of-the-art methods for polyphonic music comparison". In: *Proc. Workshop Exploring Musical Information Spaces (WEMIS)*. 2009, pp. 46–51. ISBN: 9788469260821. URL: https://www.dlsi.ua.es/gent/drizo/wemis09/proceedingswemis2009.pdf (cit. on p. 16).

[SMM12] M. Sedlmair, M. Meyer, T. Munzner. "Design Study Methodology: Reflections from the Trenches and the Stacks". In: *IEEE Trans. Visualization and Computer Graphics (TVCG)*. 2012, pp. 2431–2440. DOI: 10.1109/TVCG.2012.213 (cit. on p. 43).

[SZPH16] S. Singh, S. Zhang, W. Pruett, R. Hester. "Ensemble Traces: Interactive Visualization of Ensemble Multivariate Time Series Data". In: *IS&T Visualization and Data Analysis (VDA)*. Vol. 2016. 2016, pp. 1–9. DOI: 10.2352/ISSN.2470-1173.2016.1.VDA-505 (cit. on p. 15).

[TE06] P. Toiviainen, T. Eerola. "Visualization in comparative music research". In: *Proc. Computational Statistics (Compstat)*. 2006, pp. 209–219. ISBN: 9783790817096. DOI: 10.1007/978-3-7908-1709-6_16 (cit. on pp. 15, 31, 56).

[TF17] T. Tanaka, K. Fujii. "Melodic Pattern Segmentation of Polyphonic Music as a Set Partitioning Problem". In: *The Musical-Mathematical Mind: Patterns and Transformations*. Ed. by G. Pareyon, S. Pina-Romero, O. A. Agustín-Aquino, E. Lluis-Puebla. Cham: Springer, 2017, pp. 291–298. ISBN: 9783319473376. DOI: 10.1007/978-3-319-47337-6_29 (cit. on pp. 16, 49).

[TK04] S.-L. Tan, M. E. Kelly. "Graphic Representations of Short Musical Compositions". In: *SAGE Psychology of Music*. 2004, pp. 191–212. DOI: 10.1177/0305735604041494 (cit. on p. 11).

[Toi05] P. Toiviainen. "Visualization of Tonal Content with Self-Organizing Maps and Self-Similarity Matrices". In: *ACM Computers in Entertainment*. 2005, pp. 1–10. DOI: 10.1145/1095534.1095543 (cit. on pp. 15, 31).

[VN20] K. Vrotsou, A. Nordman. "A Window-based Approach for Mining Long Duration Event-sequences". In: *EuroVis Visual Analytics (EuroVA)*. 2020. ISBN: 9783038681168. DOI: 10.2312/EuroVA.20201087 (cit. on p. 16).

[War10] C. Ware. *Visual Thinking for Design*. en. Burlington: Morgan Kaufmann, 2010. ISBN: 9780080558417 (cit. on pp. 17, 27, 30).

[Wat02] M. Wattenberg. "Arc diagrams: visualizing structure in strings". In: *IEEE Symp. Information Visualization (InfoVis)*. 2002, pp. 110–116. DOI: 10.1109/INFVIS.2002.1173155 (cit. on pp. 16, 33).

[ZLD+15] J. Zhao, Z. Liu, M. Dontcheva, A. Hertzmann, A. Wilson. "MatrixWave: Visual Comparison of Event Sequence Data". In: *ACM Conf. Human Factors in Computing Systems (CHI)*. 2015, pp. 259–268. ISBN: 9781450331456. DOI: 10.1145/2702123.2702419 (cit. on pp. 15, 34, 50, 52).

All links were last followed on November 24, 2020.

Declaration

I hereby declare that the work presented in this thesis is entirely my own and that I did not use any other sources and references than the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations. Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before. The electronic copy is consistent with all submitted copies.

---

place, date, signature