Ludwig-Maximilians-Universität München
Department "Institut für Informatik"
Lehr- und Forschungseinheit Medieninformatik
Prof. Dr. Heinrich Hußmann

Projektarbeit

AudioPhield: A framework for music similarity analysis.

Matthias W. Schicker
schickea@ifi.lmu.de

Bearbeitungszeitraum:	10. 11. 2006 bis 10. 5. 2007
Betreuer:	Dipl. Inf. Otmar Hilliges
Verantw. Hochschullehrer:	Prof. Dr. Butz

Zusammenfassung

Neue Computer- und Netzwerktechnologien haben zu einer weiten Verbreitung großer digitaler Musikbibliotheken in Form von privaten Sammlungen oder Webshops geführt. Aufgrund unzureichender und unkomfortabler Zugriffsmechanismen werden neue Interaktionsmöglichkeiten und Visualisierungen zum Durchsuchen und Organisieren dieser Bibliotheken notwendig.

Zur Lösung dieses Problems haben wir mit der Entwicklung des Softwaresystems "AudioPhield" begonnen. Damit soll eine intuitive Interaktion mit größeren Musiksammlungen möglich sein, indem einzelne Musikstücke als Icons auf einer Starfield-artigen Benutzeroberfläche dargestellt werden. Lieder, die sich ähnlich anhören, sollen dabei möglichst nahe beieinander platziert werden.

Im Rahmen dieser Projektarbeit haben wir den ersten Teil des Systems implementiert: Mit Hilfe von Methoden und Techniken aus dem Forschungsbereich des "Music Information Retrieval" (MIR) sollen zu beliebigen Musikstücken im MP3-Format verschiedenste Attribute wie "Geschwindigkeit", "Basslastigkeit", "Konsonanz", o.ä. nur anhand des tatsächlichen Audiosignals zugeordnet werden. Dazu haben wir ein flexibles und erweiterbares Framework als Umgebung für Algorithmen zur Extraktion dieser Attribute entwickelt und bereits mit einigen derartigen Algorithmen bestückt.

Im Folgenden werden wir die psychoakustischen Grundlagen des MIR, Design und Implementierung des Frameworks und bereits entwickelte Extraktoren vorstellen.

Abstract

Recent computer and network technologies let to widespread distribution of big digital music databases in the form of private collections or web shops. Because of insufficient and uncomfortable access mechanisms the need for new ways of interaction and visualizations arises.

To solve this problem, we started developing the software system "AudioPhield". It is supposed to enable intuitive interaction with larger collections of music by depicting single pieces of music as icons on a starfield-like user interface. Similar sounding songs are to be placed as close to each other as possible.

In the context of this project thesis, we implemented the first part of this system: Utilizing methods and techniques from the field of research "Music Information Retrieval" (MIR) attributes like "speed", "bass-intensity", "harmony", etc. shall be assigned to arbitrary pieces of music just by processing the actual audio signal. Therefore, we developed a flexible and expandable framework as environment for algorithms for extraction of these attributes and provided it with some algorithms of this kind.

In the following, we will introduce the psychoacoustic basics of MIR as well as the design and implementation of the framework. Also, we will present some already developed extractors.

Aufgabenstellung

Tasks:

- Design and implement a framework as basis for algorithms to analyze digitalized music.
- Find and implement algorithms that are able to extract meaningful information out of music represented as digital audio signals. The extracted information is supposed to be useful for the computation of the perceived similarity of different pieces of music.
- Give a presentation of the results in the Oberseminar.
- Write a summary of the work done. Its size should be about 60 pages.

Ich erkläre hiermit, dass ich die vorliegende Arbeit selbstständig angefertigt, alle Zitate als solche kenntlich gemacht sowie alle benutzten Quellen und Hilfsmittel angegeben habe.

München, 21. Mai 2007

. . . . . .

Contents

1 Introduction

Music plays an important role in our daily life. Music accompanies us as background noise in public places like elevators and subway stations, as stimulative soundtrack during sports, as medium to transport emotions in movies (what would "Once Upon a Time in the West" be without the legendary melody by Ennio Morricone?) or even as primary entertainment in concerts. As manifold as the named surroundings for music consumption, are also their characters and forms of appearance: The attempt to sort music into genres spawned few rather non-specific major genres (like "rock", "pop", ...) and an unmanageable number of ambiguous sub-genres. Steady progress of technology makes this problem even worse: Computational aid integrated in the creation process of music offers access to entirely new "instruments" and effects - and so to previously unimaginable new songs. The mentioned ambiguity makes it very difficult to choose from the vast amount of songs, which the music industry offers today, although most users know well, which kinds of music they like and dislike.

Encouraged by a lot of positive feedback from our previous project "AudioRadar"[17], we started developing the software project "AudioPhield"[1] as an approach to solve this problem: AudioPhield is supposed to be able to compare arbitrary digitally stored songs and visualize their similarity as clearly as possible. The following project thesis covers the fist part of this task: A flexible, expandable framework is to be developed, which uses technologies from "music information retrieval" (MIR) to extract as meaningful attributes as possible from songs.

1.1 Music Information Retrieval

The branch of research combined under the term "music information retrieval" (MIR) tries - roughly spoken - to teach computers to listen to music similar to the way humans do. To achieve this, researchers preferably (but not exclusively) use music represented as digital audio signal; i.e., the music is stored as a sequence of binary values corresponding to sound pressure levels in an analog acoustic waveform. Although this kind of data can be easily processed by modern computer systems, it has proven surprisingly difficult to build algorithms that can imitate the listening process of even untrained human listeners, who are able to label a song emotionally (e.g. as aggressive or melancholic) or recognize the playing instruments from short excerpts of music. To achieve the goal of human-like music perception, MIR incorporates many different research fields, in particular computer science, musicology, medicine, neurobiology and sociology[2].

MIR offers - besides academic insights - the foundation for many useful applications. The following paragraphs shall give an outline of the seemingly most important ones:

Musical Genre Recognition is the "classic" exercise of Music Information Retrieval. It's target is to sort a song into an array of meaningful "genres". Although a lot of research is done to range music in the typical categories, as "grunge" or "bluegrass", these genres are not necessarily limited to these: Zanon and Widmer for example introduced a system which is able to distinguish between the playing styles of famous pianists[73].

Automatic Music Transcription means the creation of sheets of music from already sampled digital music or - formally spoken - the transformation of an acoustic signal into a

[1] The name is a combination of "audio" and a blend of "field" and "-phil"(Greek for "loving").

[2] Although the computer science aspects are clearly central in this paper, you can find a short discussion of some (here relevant) basic concepts of musicology in chapter 1.2.3 and some medical aspects in 1.2.1 on page 3

symbolic representation[22]. This may prove useful for people who want to play along their favorite music, or for the recording of spontaneous jams (e.g. Jazz); but AMT-techniques predominantly serve as basis for other techniques like "Query by humming"(see below).

Real-Time Accompaniment tries to supply "a missing musical part or parts, generated in real time, in response to the sound input from a live musician"[52]. Thus, the software can support a musical soloist and generate the illusion for her of being accompanied by possibly a large ensemble of musicians. This could enrich the experience of playing music at home as well as live performances.

Audio Fingerprinting means the reduction of a song to a much smaller (in the sense of storage space) sequence of numbers that identifies the track unambiguously. The special challenge is to achieve this independently from the specifics of an actual recording as the encoding of the song (e.g., it should be robust against changing the codec or bit rate). This could be used to observe music transfers in file-sharing networks (to block copyright protected content or automatically collect royalties) or to deliver additional information to songs in private databases[14][61].

Efficient music database query: "Query by humming" Imagine the following situation: You are listening to your radio while driving your car, switch the channel - and hear a tune you have been looking for for a long time. Unfortunately the radio station fails to tell you the title or band of this song. Up to now, you could only hope to hear the same song with some context again or you could sing the melody to a friend, hoping she might recognize the song and happen to know the title. With the help of MIR technologies, a computer can take the role of this friend: A music database could extract melody lines from an inventory of music files and store them. If one hums the catchy tune now into a microphone, the database can compare the hummed melody with the stored ones, and so deliver a list of possible corresponding titles[3] quickly. Such systems already exist and are called "Query by Humming" (try for example [35]). Query by Humming may also be a very valuable feature for music play and management programs like Apple's "iTunes"[1] or "Nullsoft Winamp"[45]: The user could just hum the song she would like to hear, and the software would play along[11].

Finding similar songs As stated above, the sheer amount of currently available music makes it very hard for customers of music stores to find songs which meet their individual taste. Using MIR technologies to estimate, how similar different songs are, seems a promising way to cope with this problem: Instead of relying on recommendations from friends or listen to extracts of randomly chosen songs, a user just specifies her favorite songs; a software system can now find and offer the songs that were identified as most similar to the given ones. Assuming that the software chose the "right" dimensions to compute the similarity, the user will likely enjoy listening to this automatic selection.

A good estimation of the similarity of songs can also be very valuable for automatic generation of playlists. To provide the acoustic background for, for instance, parties or sports, it is sufficient to specify just one song as starting point; the software identifies then similar songs in the database and enqueues them in the playlist. The web radio "Pandora"[51], as to our knowledge the first system of this kind, shows impressively how well this can work - albeit their underlying data is not automatically but manually extracted

[3]Mostly it will not be possible to specify the artist too, because many tunes are often reused ("covered") by different artists, who may change the surroundings of the melody but keep the abstract tune unmodified.

from the music[4]. The knowledge of the "distance" between songs also opens the door for completely new ways to visualize and interact with music databases, as projects like "AudioRadar"[17] or "PlaySOM"[43] demonstrate and our current project "AudioPhield" is supposed to confirm.

In section 4.2, we will discuss some possible algorithms and methods, that will deliver the data for "AudioPhield", relying on the knowledge of psychoacoustics outlined in chapter 1.2.

1.2 Psychoacoustics and the Perception of Music

This chapter covers some basic understandings about the way humans perceive music. It is divided into three major blocks:

After a short introduction to physical properties of sound and its reception and translation to neurological impulses by the human ear in 1.2.1, we will discuss the perception of loudness and pitch, and the differences between tones from different instruments. In the last part, 1.2.3, we will give an overview of basic musicology and outline the perception of rhythm.

1.2.1 The Ear

"Audition" (the medical term for the sense of hearing) is a process depending primarily on two organs, the ear and the brain. For now, we will concentrate on the ear, since its physiology is very important for the way, humans perceive sound; also its functionality is in comparison well understood - albeit far from exhaustively.

The ear is responsible for picking up sound as sound waves from the environment and translating it into neurological impulses, which the brain then interprets. Most of the sounds humans are capable to sense travel to them through the air as airborne sound[5]:

See Figure 1.1 as illustration of the following descriptions. The sound waves are gathered by the outer ear (the "pinna") and sent down the ear canal to the eardrum. The waves cause the eardrum ("tympanic membrane") to vibrate. Since the eardrum is connected to the system of tiny bones ("malleus", "incus" and "stapes") in the middle ear, these are also set in motion. Thereby, the bones act as a mechanical impedance translator: Normally, because of the impedance differences of the media in the air-filled "tympanum cavity" and the fluid-filled inner ear, 98% of the arriving waves would be reflected; the bones of the middle ear compensate this effect and lower the reflection rate to 40 %. Then, the stapes transfer the received and translated vibration to the "elliptical window", which causes the fluid in the cochlea, the "perilymph", to move. Depending on the frequency of their vibrations, waves traveling through the narrowing cochlea initiate strong oscillations in different parts of the basilar membrane. This membrane is covered with about 30,000 "hair cells" (cells with multiple proteine strings, which protrude into the lymph fluid, attached to them), which emit neurological pulses through the auditory nerve to the brain when their "hairs" get bent. Thereby applies: The stronger the stimulus (hence movement of the perilymph around hair cells), the "stronger" (more nerve endings firing at a higher rate) the signal to the brain [27, 32].

[4] "LastFM" (`http://www.last.fm`) is another web radio and offers a similar service, but takes a completely different approach: Instead of paying attention to the actual music, lastFM observes the listening habits of a large community, generates profiles and combines matching profiles to deliver recommendations.

[5] *Most* of the sounds, because sounds of very low frequency ("subsonic noise" beneath about 25 Hz) and/or high energy initiate vibrations in the bones of a listener and are perceived as "impact sound" without passing the auditory canal. Also, when the head is surrounded by other media like water, sound can travel through these media, too.

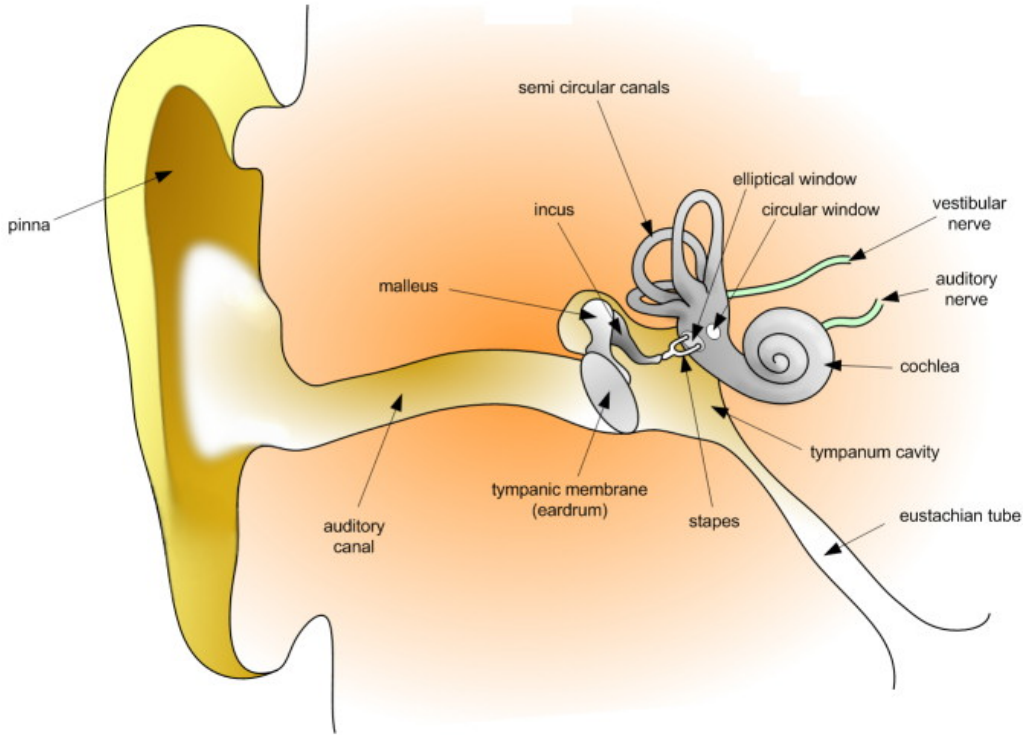


Figure 1.1: Anatomy of the human ear. (The length of the auditory canal is greatly exaggerated in this image). Source: Wikipedia[77].

1.2.2 Tone Perception

As seen in section 1.2.1, the energy of sound waves reaching one's ear determinates the strength of neurological impulses sent to the brain. Analog to this, the frequency of the incoming waves determinates at which spots in the cochlea resonances emerge, and so which nerve ends "fire". Unfortunately, these correlations depend on various factors - and even then, the intensity of the signals sent to the brain can not be directly interpreted as the perceived "loudness". The same is true for the actual frequency of a sound wave: Depending on the sound, there is no simple correlation between which hair cells are stimulated and the "pitch" of a sound; what a person hears seems to have little to do with the actual sound waves in some cases. We will summarize some of the major factors of influence on the perception of loudness and pitch in the following paragraphs.

The intensity of sound is usually measured as "sound pressure level" (SPL). SPL is a logarithmic measure of the magnitude of sound waves at an arbitrary point in space relative to a reference sound and is usually quantified in decibel[6] as dB(SPL). According to the "Weber-Fechner law", the perceived loudness of a sound also correlates roughly logarithmically to its sound pressure. Thus, it is convenient and widely used - albeit quite imprecise - to measure the volume of sounds in dB relative to the hearing threshold of the human ear at the frequency of 1kHz, the once thought quietest sound a human can hear[7] (as we will see later in this chapter, it is necessary to define the frequency of the reference tone). The fact that sound intensities are mostly given in dB(SPL) somewhat disguises

[6]The decibel, as a tenth bel, is a logarithmic scale used to specify the ratios between two powers or intensities. It is defined as $X_{DB} = 10 \log_{10} \left(\frac{X}{X_0} \right)$, with X as value to compare and X_0 as reference value.

[7]The actually quietest sound audible for the average human ear has a frequency of about 2kHz and is about 8 dB(SPL) softer than the reference tone (see figure 1.2).

the ear's enormous intensity-bandwidth of about 120 dB(SPL); the softest audible sound causes thereby the ear-drum to vibrate with an amplitude of less than the wave-length of visible light[8][27], while the loudest sounds, which can be heard without immediate damage to the auditory system, have an amplitude 10^{12} times bigger. In this wide range of possible sound intensities an average person can normally distinguish between two sounds that are at least 1 dB apart from each other; this means that the just noticeable loudness differences increase with increasing volume. Further, the auditory resolution is negatively influenced by the time passed between two sound events and positively by the existence of reference sounds, which keep their volume. If different sounds occur at roughly the same time, masking effects can modify the audibility of individual sounds: Loud sounds can impede the perception of weaker sound events at the same time ("simultaneous masking") or shortly afterward (in some cases even *previous*) according to their frequencies and associated "critical bands". The width of these critical bands, and thus the range of masked sounds, again depends among other things on the frequency range and intensity of the sounds (see [76] for further discussions of this topic)[9]. It is also due to masking that melody lines are easily distinguished from the background in complex musical scenes: Although they are not actually perceived, masked sounds add to the overall stimulation of the inner ear. The perception of loudness depends mainly on this overall stimulation. Thus, melody-lines are perceived louder than they physically are [42, 74, 56].

Perception of Loudness also strongly depends on the frequency of sounds: An average listener may apprehend sound waves with 60 dB(SPL) as nearly inaudible (e.g. at 40 Hz) or fairly loud (e.g. at 2 kHz). To cope with this problem, "equal loudness contours" were developed, which allow the conversion from objectively measurable dB(SPL) into a unit of subjective loudness perception. The first ones to measure contours of this kind were Fletcher and Munson in 1933. They gave headphones to a number of test persons and let them compare sinusoid tones with reference tones at 1 kHz. The intensity of the test tone was then adjusted until it was perceived to be of the same loudness as the reference tone. See figure 1.2 to see their results. A few studies with virtually the same setup, but varying results, were conducted since then, which led to the standardized equal-loudness contour "ISO 226:2003". Based on the Fletcher and Munson contour, the unit "phon" was created: A value given measured in phon should be perceived equally loud regardless of the sound's frequency. Thereby, a difference of 1 phon equals per definition the difference of 1 dB(SPL) for pure tones at 1000 Hz citeurlWikipediaPhon.

So, measuring loudness in phon allows direct comparison of sound events of different frequencies. However, it has to be mentioned that equal-loudness contours reflect the average perception; individual loudness estimations often vary widely from these curves, influenced by genetic disposition, age, gender[10], race, damage to the cochlea, recent exposure to loud noises or even tiredness. In addition, equal-loudness contours just indicate perceived loudness for sinusoid tones - actual noises with broader spectra of frequencies are usually evaluated differently. Furthermore, the phone values (which are basically dB(SPL)) are not proportional to perceived loudness: The perceived volume-difference between 10 and 20 phon feels substantially greater than the difference between 70 and 80 phon. Because of that, the unit "sone" was created; sone is based on phon, but compensates the differences in perceived loudness depending on absolute sound pressure levels. Thus, by

[8]These volumes are rather theoretically, since no usual circumstances are quiet enough to hear sounds near the threshold of hearing: A room with a basic sound level of 30 dBA is often characterized as "very quiet"[42]

[9]Masking effects proved very valuable at the compression of music. Modern, strongly compressing digital music formats like *Fraunhofer Institue*'s famous "MP3" make heavily use of these to achieve their nearly inaudible, but lossy compression.

[10]It has been proved, that women usually have a lower threshold of hearing than men. Also, their decrease of audition-sensitivity with age seems to occur slower.

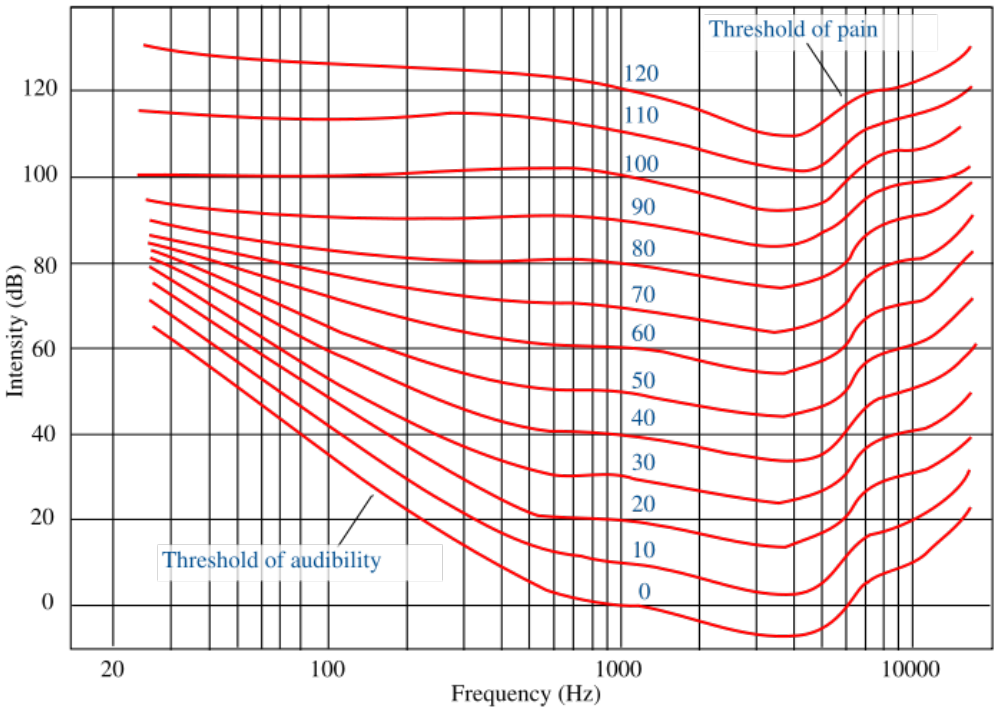


Figure 1.2: Fletcher-Munson curves for 0 to 120 phon. The 0-curve describes the intensity of just audible sounds. Source: Wikipedia[74]

using sone, the perceived volumes of (pure) sounds get comparable, regardless of their objective intensities and frequencies [27, 78, 42].

Since the computation of phon or sone are rather costly, technical equipment and software (e.g., AudioPhield) often utilize weighting filters as another way to compensate the irregular loudness perception of humans. These filters assign gains in dB relative to the original sound volume depending on the frequency. So, opposed to the several phone curves, there is just one curve used for all frequencies and sound intensities. Since the phon-curves are not parallel (as can be seen in Figure 1.2) such weighting curves can only be valid for a narrow band of intensities around the curve. Because of that, there is a family of curves defined in IEC179 with each targeting at a different range of sound intensities. These are labeled from 'A' to 'D', with A (the most commonly used, albeit often improper) covering quiet sounds and D covering loud noises (see figure 1.3) [75].

Furthermore, perception of loudness also depends on the duration of a tone. Sounds with durations shorter than 200ms are perceived softer than tones longer than this duration. In this area, the correlation between duration and perceived loudness is linear: A pure tone between 250 Hz and 4000 Hz needs to have roughly a 1.5 times bigger sound pressure level than a tone with the same frequency and amplitude but the double duration to sound equally loud. This correlation does not apply for tones longer than this threshold duration; their volume is perceived as equal. Single sound events shorter than 50 ms are generally not heard as tones, but rather as clicks regardless of their actual frequency. However, if a sound event continues for long times (> 30 s), the brain gets used to the sound and mutes it out as unimportant background noise. The average duration of a sound event in music or speech is between 50 milliseconds and two seconds [28, 41, 32].

Also, the ear adapts to the average noise level of the current environment similar to the eye adapting to the ambient brightness. Responsible for this adaption are two muscles in the middle ear, which manipulate the tension of the eardrum (see figure 1.1) and adjust

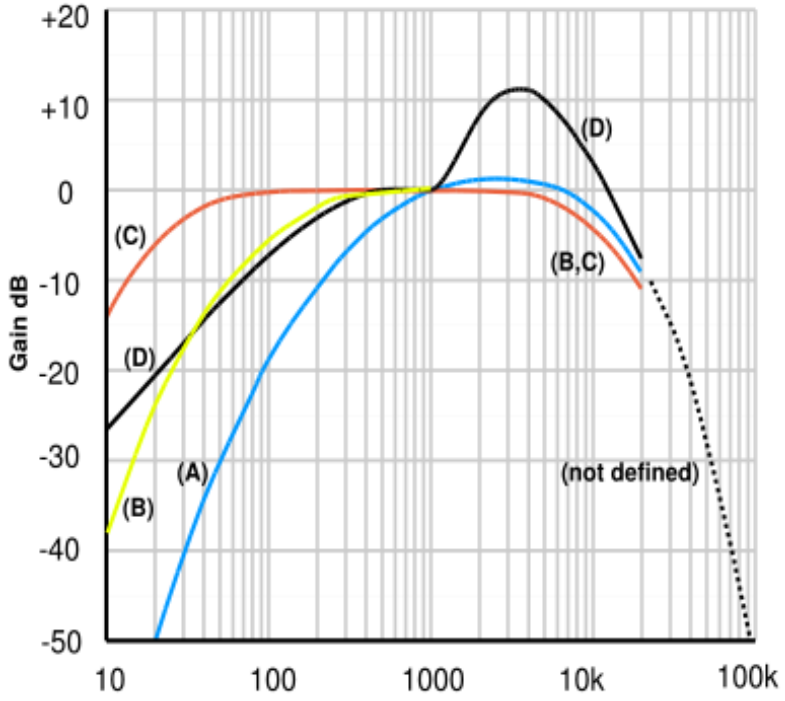


Figure 1.3: Weighting curves: A-weighting (blue), B (yellow), C (red) and D (black). Source: Wikipedia[75]

the mobility of the tiny ear bones. Because of this mechanism, it is impossible for humans to measure the absolute loudness of sounds; loudness can only be quantified relatively to other sounds [27].

For not pure tones (i.e., sounds which are built out of more than a single sinusoid wave) the perception of loudness is also dependent on the individual frequency profile of a sound: Narrow and harmonic spectra seem usually softer than broad or "noisy" spectra, even if their sound pressure levels and pitches are about the same. In the field of complex sounds, the perception of loudness (and pitch as we will see) is highly complex and subjective.

Another very important feature of sound apart from loudness is the perceived "pitch" of a sound. We will introduce the concept of pitch in the following paragraphs beginning with the perception and discrimination of the pitch of simple sounds and will then discuss the pitch of complex sounds, as notes played by music instruments.

With the term "pitch" we mean the tonal height of a sound event as perceived by the average listener. As described in section 1.2.1, the frequency of sound waves entering the auditory system determinates at which point oscillations in the cochlea occur and thus which hair cells are stimulated. So, for the simple case of pure (i.e. sinusoid) sounds, the frequency of a sound correlates directly with the perceived pitch (albeit influenced by the intensity of the sound). This correlation, however, is not proportional but logarithmic: The difference in frequency necessary for two tones to sound equally distant to each other is substantially higher in high frequency regions than in low regions. The inverse is true for the ability to distinguish between tones of adjacent frequencies: While in the low end of the spectrum of audible frequencies (around 20 Hz) differences of 5 Hz are enough to separate tones, in the high end of the spectrum (around 16 kHz for an adult) even sounds that

are several hundred Hz apart may be perceived as equally high. Altogether, the human ear is able to distinguish between 620 tone heights, when they are played separately - the resolution for tones played at the same time is much higher. Stevens, Volkman and Newman proposed therefore a scale, that would provide a linear measurement of the perceived pitch and thus hide the complexity of the logarithmically changing sensitivity of the human ear. They called this scale "mel" for "melody" [79, 32, 64, 28].

Real life sounds, unlike the previously watched pure tones, consist normally of several waves of different frequencies and intensities, and so may have several pitches - or none, as in the case of broadband "white noise"[11]. In the case of a single note played by music instruments like a piano, in contrast, the average listener hears just *one* pitch, although sound waves of at least ten measurable distinct frequencies stimulate her eardrum. To explain this, there are currently two main types of pitch models: Firstly, *place* models of pitch explain the perception of few pitches in complex sounds as the result of pattern-recognition analysis of a sound spectrum (so, considering the cochlea as a spectral analyzer). Place models are able to explain a wide range of acoustic phenomena, but expect a spectral resolution of the ear that is not actual there; so, the model works fine but does not reproduce the way the human hearing system works. Alternatively, there are *temporal* models of pitch. In temporal models, the cochlea acts again as spectrum analyzer, but instead of building pitch directly out of combinations of found frequencies, the (regarding frequency-separation as quite rough estimated) output of the cochlea is inspected in the time domain. Thereby it is assumed that the entire time domain signal is coded into the firing rate of the nerve fibers. The resulting pitch is computed out of periodic fluctuations in the separate subband signals using a variety of algorithms. Although it was shown, that the human auditory system actually uses a complex combination of both models, we will use generally the place model of pitch in the following, since it proved better comprehensible and computable [56, 25].

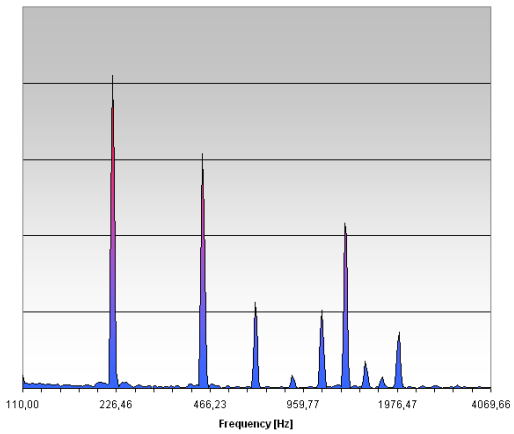
The difference from perceptively single-pitched complex sounds to the previously mentioned noise is that here the separate frequencies of the sound's spectrum are in a well defined order: They are all exact integer multiples of the lowest frequency in the spectrum, the "fundamental frequency"; the tones at higher frequencies are called "overtones"[12] (see figure 1.4). This fundamental frequency is the pitch most listeners assign to the complete spectrum of frequencies. The vast majority of fundamentals in speech and music falls in the range from 60 Hz to 1,000 Hz, higher ones are rare, but occur in the case of classical music up to 4,500 Hz [41, 56].

Thereby, different instruments have their own unique patterns of overtone-strengths compared to the fundamental, which stay about the same regardless of the played note or intensity (see Figure 1.4). In psychoacoustics, these unique patterns, which are easily recognized and distinguished even by unschooled listeners, are often referred to as "timbre". Depending on the actual instrument, the overtones may not meet the exact multiplications of the fundamental frequency, what causes the instrument to sound "rougher" or "noisier"[13]. Also, partials contained in the typical spectrum of an instrument are not necessarily "harmonics", i.e., integer multiples of the fundamental frequency; these so called "inharmonic" overtones usually follow other well defined rules of multiplications from the fundamental (e.g. 3:2) (see section 1.2.3 on page 17 for further discussion of the combination of com-

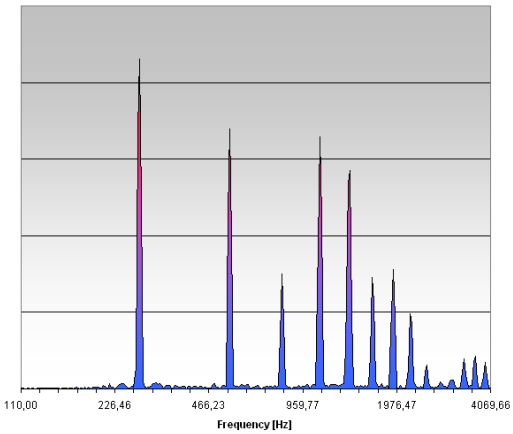
[11] White noise is a completely random signal with equal energy in every possible frequency of a given bandwidth.

[12] A similar term is "partial": It describes any distinct part frequency of a tone's spectrum - regardless if harmonic or not.

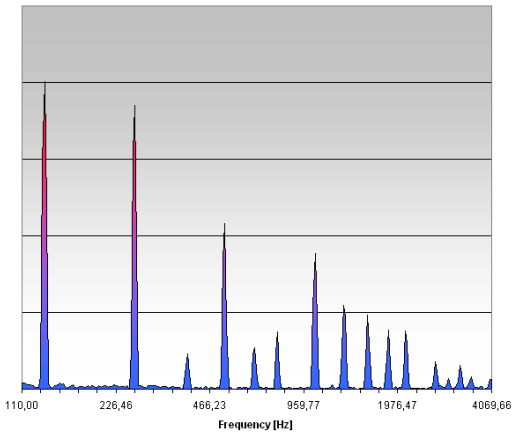
[13] Livshin and Rodet state in [31], that although the harmonic parts of a spectrum suffice normally to identify an instrument, the "noise" part is not be neglected, since it transports subtle information about the instrument and the musician's playing style.



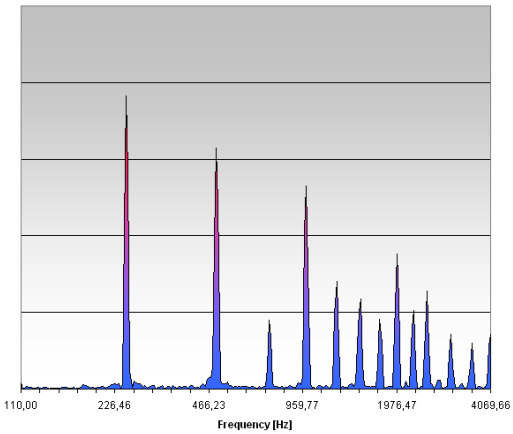
(a) E-Bass (undistorted): A3



(b) Acoustic guitar: C#4



(c) Piano: C3



(d) Piano: B3

Figure 1.4: Typical frequency spectra of three instruments at roughly the same octave. Note the logarithmic frequency axis in all diagrams.

plex frequency patterns to few pitches). Whereas some styles of music prefer the sound of instruments as clear (or "sharp" as on page 16 defined) as possible, other styles appreciate a certain noisiness in their spectrum. This is true, for instance, for electric guitars, whose sounds get deliberately distorted and smeared over a wide bandwidth to achieve a certain "dirtiness" convenient for Hard Rock or Heavy Metal. [64, 80].

In addition to the just discussed "spectral pitch", where the perceived pitch is an actual part of the considered spectrum, there also exists a "virtual pitch"[14]: The perceived tonal height does not exist in the corresponding frequency spectrum but is hinted as the greatest common divisor of the frequencies[15](see Figure 1.5). As Yost demonstrated, the human brain uses not the complete audible spectrum but a comparatively narrow frequency region from roughly 400 to 2000 Hz for the "calculation" of virtual pitches. Frequencies outside of this "dominant spectral region" seem to contribute little to the perceived virtual pitch. Due to the masking effects discussed earlier, this subconscious "completion" of a spectrum is necessary for the hearing of polyphonic music, since it happens often that overtones of

[14]The phenomenon is called "missing fundamental", "residue pitch", or "phantom fundamental" as well.

[15]Manufacturers of hi-fi systems use virtual pitch to compensate for the fact, that small speakers are rather unable to reproduce low frequencies. So, they create the illusion of bass by subtle changes in the spectrum of music during the playback.

one instrument mask the fundamental of another. For the computation of virtual pitches from a given sound spectrum, place models of pitch (see page 8) proved quite capable, albeit not in all possible circumstances, and outperformed current time models. If a "fundamental frequency" exists, but does not fit exactly a series of overtones, two different types of listeners can be observed, as recent studies by the University Hospital Heidelberg revealed. The first type, called the "f_0 listeners" primarily takes the fundamental frequency to decide the the overall pitch of the spectrum; in contrast to that, the second type (the "f_{Sp} listeners") is more influenced by the overtones when hearing the overall pitch[16]. These results also suggest that there may be individual differences in the perception intensity of virtual pitches [58, 59, 82].

The concept of virtual pitch is especially necessary for the understanding of pitch in speech and singing: A typical male speaking voice has a fundamental which ranges from 70 Hz to 150 Hz. These frequencies, however, contribute virtually nothing to intelligibility and the perceived pitch of the voice, as anyone who ever used an analog telephone can attest - a classic phone can only transmit frequencies from about 300 to about 3400 Hz. While the understanding of virtual pitches is surely necessary for the understanding of pitch in speech and singing, sufficient it is not. This is hardly surprising, since humans use speech as one of their primary communication techniques. Therefore, the human brain has special regions dedicated only to the coding and decoding of spoken content[17]. These regions are different from the areas of the brain used to process non-speech sounds and apparently work different, too. How exactly speech is decoded in the brain and how the speech center influences the perception of pitch is still an object of research. The human singing voice is rich in harmonic content, but the harmonic pattern is less regular compared to sounds of typical solo instruments: Whereas the the harmonic patterns (the ratio of separate overtones to the fundamental frequency and their relative strengths) of musical instruments vary comparably little, the overtone structure of singing is highly flexible. So, the overtone spectra of one singer can be very different, depending on the currently articulated vowel (see figure 1.6). The bandwidth of possible sounds gets even broader when considering the whole human speech organ: Singing can reach from highly harmonic opera singing to complex and partly noise-like "beatboxing".

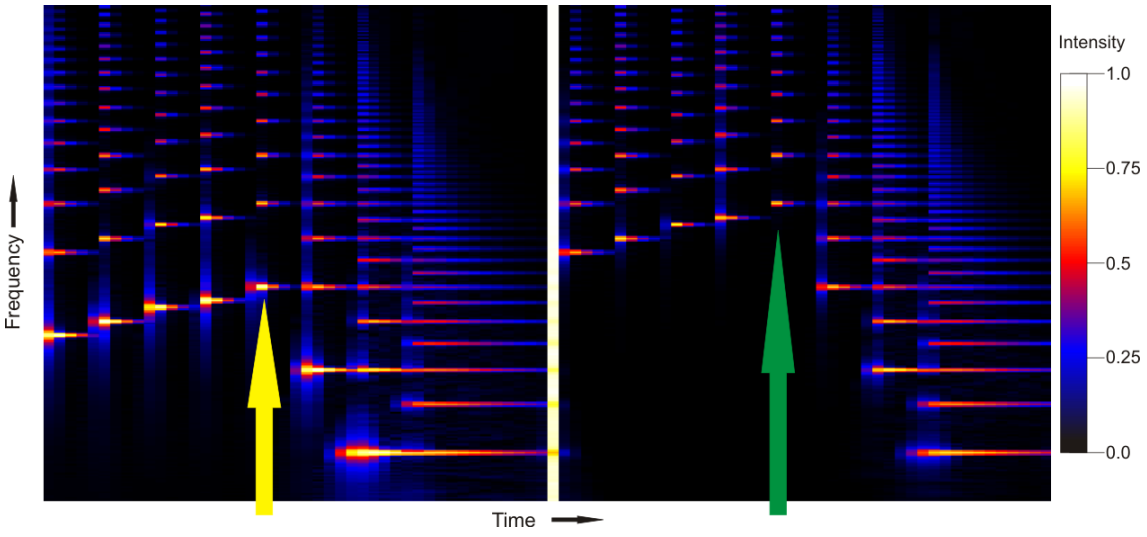
1.2.3 Music Perception

In this section, we will outline the basics of how humans perceive music by discussing musicological concepts as octaves and their partitions, harmony and simple rhythms.

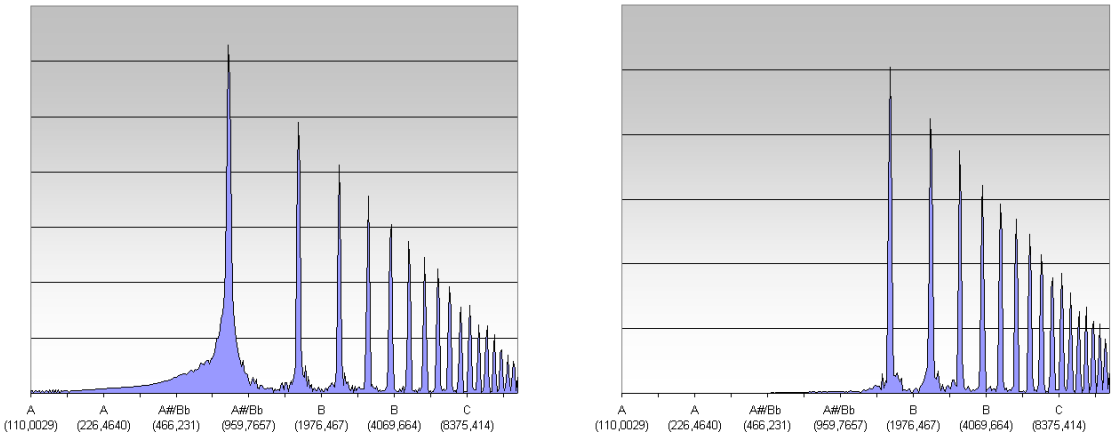
As shown in the previous section, humans differentiate between pitches not linearly but logarithmically. Therefore, the concept of "octaves" is widely spread: One octave describes the range from one frequency to the double of that frequency (e.g. from 100 Hz to 200 Hz). So, with the logarithmic pitch separation of the human ear, one octave covers the same width of perceived pitches regardless of the used start frequency. Tones that are separated by exactly one or more octaves sound very similar; this can be explained by the on page 8 introduced overtones. As seen, harmonic overtones have frequencies which are multiples of the fundamental frequency - in other words, the first harmonic overtone is exactly one octave apart from the fundamental, the third harmonic two octaves, and so on. Thus, notes separated by octaves have every other harmonic in common and show the best consonance besides the "prime" (the repetition of the same note), as can be seen in Figure 1.7. Possibly

[16] The same study also revealed a correlation between the hearing type and preferred instruments and types of music: Thus f_0 listeners enjoyed particularly instruments producing short, sharp tones like drums, guitar or transverse flute and rhythm accentuated music, while f_{SP} listeners preferred instruments, whose tones are longer and darker like cello or organ, and music with pronounced melody lines.

[17] As a result of this, human singing is *always* perceived as in the foreground of an acoustic scene, even if it is far weaker than the strongest sound source.

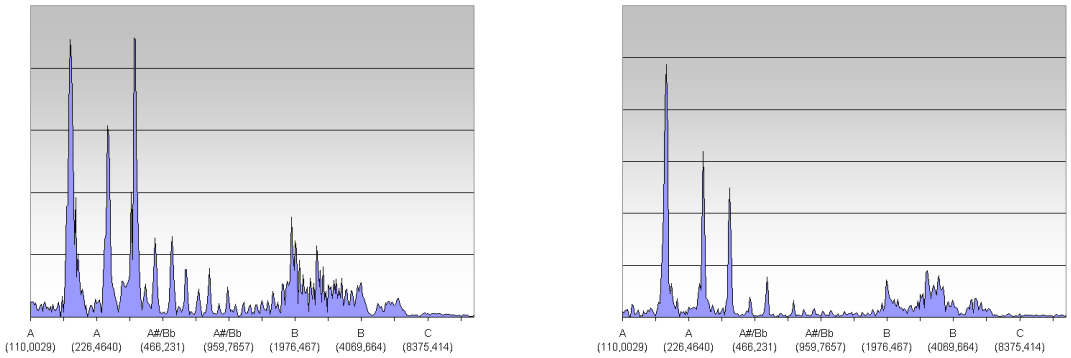


(a)



(b) (c)

Figure 1.5: Illustration of "virtual pitch". (a) shows a sequence of tones first played regularly, then without the fundamental frequency, analyzed using the CQT method (see 4.2.3). The diagrams below show the spectra of two tones from (a which are perceived as virtually identical: (b) full spectrum, (c) virtual pitch spectrum. Note that the frequency axis in all diagrams is logarithmic.



(a) Sung vowel: “e”

(b) Sung vowel: “i”

(c) Sung vowel: “o”

(d) Sung vowel: “u”

Figure 1.6: CQT spectra (see 4.2.3) of a singer singing four times the same tone with the same loudness, but articulating four different vowels. Note the logarithmic frequency scale.

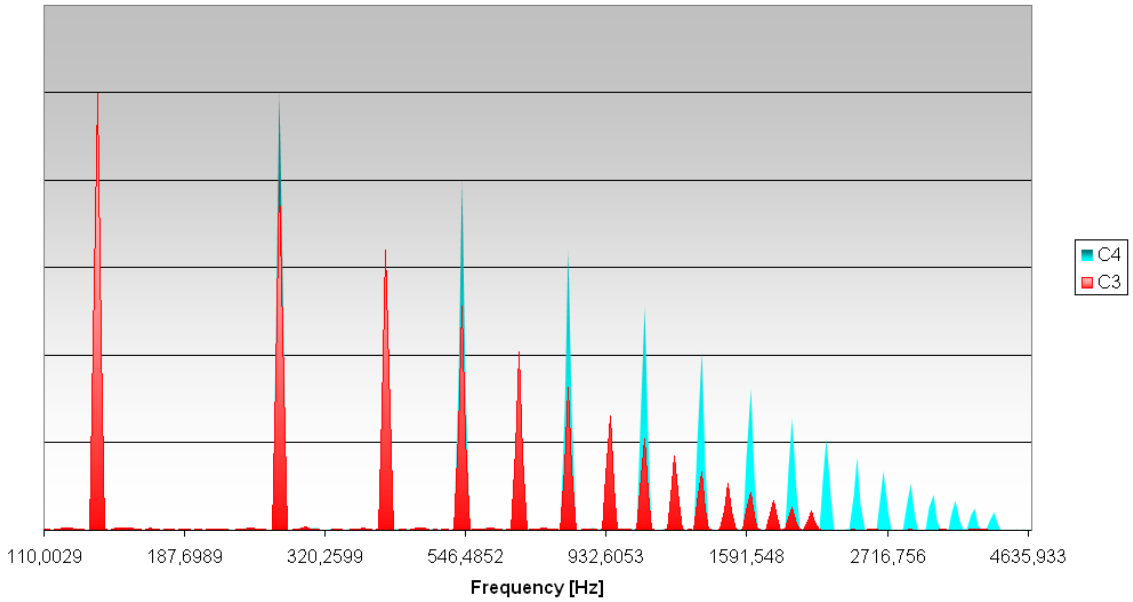


Figure 1.7: Two idealized harmonic tones (C3 and C4) exactly one octave apart from each other depicted in the frequency domain. Note the logarithmic frequency scale.

because of that, the octave is used in all music cultures worldwide as the basic interval and as basis for musical scales.

Virtually all known music has some kind of "musical scale" as fundament, which defines the pitch-relation from one note to another. These scales usually divide one octave evenly[18]: In Western countries, where the "12 equally tempered system" dominates, one octave contains 12 tones[19]; thereby, one note has a frequency of $2^{n/12}$ times the base frequency (with any whole number n). In other musical cultures, different systems are usual: In the middle east, for example, the 53 equally tempered system (with factors of $2^{n/53}$) dominates, in south India one octave has 72 tones, to mention just a few. To prevent the following discussion from becoming too complicated, we will restrict us from now on to modern Western scales. The scale containing all 12 notes on one octave is called the "chromatic scale", the intervals between two neighbors in this scale are named "half tone steps" or "semitones"; consequently, the combination of two half tone steps is considered a "whole tone step". Each of the semitones is given a name to identify them precisely, beginning from the base frequency: A, A#, B, C, C#, D, D#, E, F, F#, G, G#. These names repeat themself in every octave; therefore one usually adds the octave number to name a specific note[20]: D0 means the fifth semitone in the lowest octave (18.35 Hz), A4 the first semitone in the fourth octave (440.0 Hz), etc.. It has to be noted that the scales of most actual instruments do not follow the above scheme precisely, but are stretched. This means that low tones are tuned slightly flat (lower than the canonical frequency), high tones slightly sharp (higher); this phenomenon is not due to flawed instruments or incapable tuners, but to the observation that scales tuned in this manner are perceived as more natural and pleasant.

[18] One exception is the "natural scale system", which just defines tones allowed from any given base tone as $2^n \times 3^m$ respectively $2^n \times 3^m \times 5^p$ times the basic frequency, with n, m and p as natural numbers. This scale does not include the concept of octave and is widely used in folk music.

[19] Actually, since the octave is subdivided 12 times, the octave contains 13 notes. However, it is common practice to not count the highest tone, because it is regarded as belonging to the next octave.

[20] This naming convention is also called "scientific pitch notation"

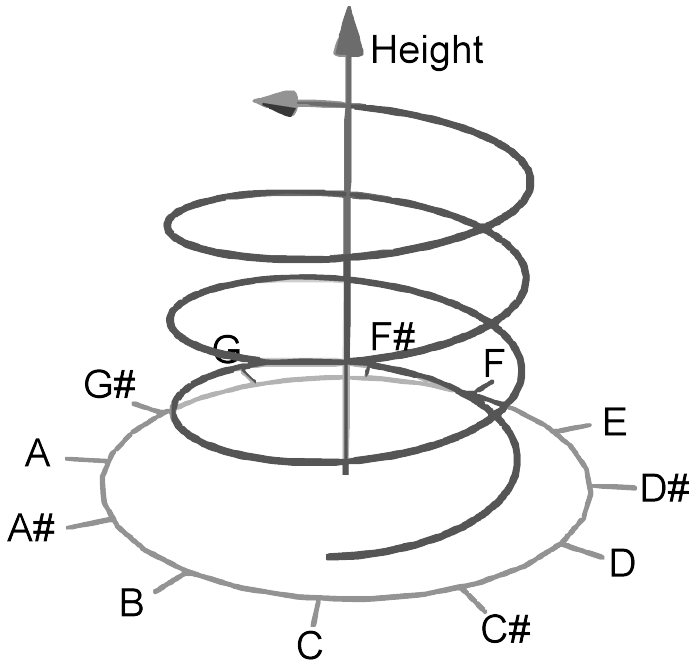


Figure 1.8: Shepard's "pitch helix" can be used to estimate the affinity between two notes. In this model, pitch is expressed on two dimensions: the pitch-class direction around the helix relates pitches according to their distance within one octave and the height direction relates notes of different octaves.

As stated before, notes that are exactly one octave apart from each other sound very similar (a phenomenon usually referred to as "octave affinity"); also, notes separated by only one half-tone step are considered similar, too. It raises the question, which affinity is stronger. Numerous models have been developed to answer it - with varying applicability and precision. Among them, there is a number of approaches called "pitch spaces", which try to assign to any two notes an affinity-value. Because there is a vast number of proposed pitch spaces, we want to introduce only one rather descriptive here, Shepard's "pitch helix" (see Figure 1.8): Euclidean distances between two arbitrary points on the helix representing actual notes give an estimation of their affinity. The position on the basic circle of the helix is also called "chroma" referring to the chroma scale. It has to be criticized that the special affinity with the "fifth", a tone with a frequency 1.5 times higher than the base tone (see below), is not clear in this model. The actual similarity, however, depends heavily on the timbre of the played instrument [56, 5].

The chromatic scale is rarely used as a whole in actual music; it should rather be seen as the superset for all actually used scales. Over the last centuries a lot of different scales were in use, like the "tetratonic", "pentatonic", or the "enigmatic" scale, but today's Western music culture is dominated by the "diatonic" scales. These scales, which are also called the "heptatonia prima"[21], always contain seven notes with five whole-tone steps and two half-tone steps. The semitones must thereby be maximally separated, i.e., there are always two or three whole-tones between them. This pattern is repeated after a full octave; so, from all possible semitones of the chromatic scale a subset of notes, which is considered harmonic and ranges over all audible frequency bands, is selected. Intervals between notes in the diatonic scale have thereby their own names, that are widely used by musicians, as

[21]"Heptatonia prima" may be roughly translated as "first seven step scale". The musical term "octave" (from Latin "octavus": the eighth) originates from these seven steps: It is the interval containing eight steps.

follows (amount of semitone intervals in brackets): unison/prime(0), second (1, 2), third (3, 4), fourth (5), fifth (7), sixth (8, 9), seventh (10, 11), octave (12). It is common, to add to ambiguous intervals (like seconds) the words "minor" and "major" ("second minor", for example, means that the second is just one semitone wide). Since the pattern of minor and major seconds in the diatonic scale is repeated over the whole chromatic scale, there are only twelve different sets of distinct diatonic scales [57].

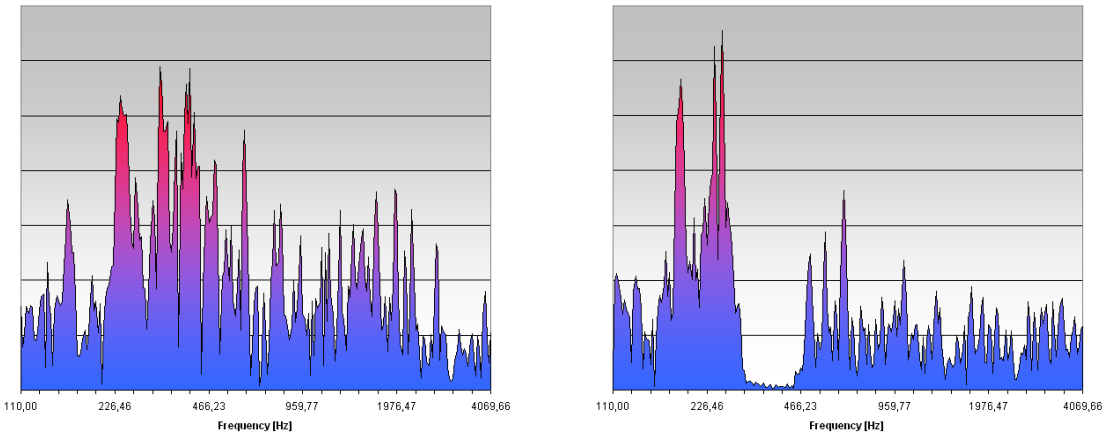
However, in most situations one special note will be chosen as the "tonic". This note (or pitch class, since it is not assigned to any octave) is supposed to be the central note of the scale. Together with the tonic a "mode" is selected. The mode can be either "minor"[22] or "major" and defines which diatonic pattern is used, that is to say, at which positions in the scale the minor and major seconds are located[23]. Since the tonic is always the first note of the scale, the selection of notes belonging to this scale is precisely defined. With this knowledge, it is now possible to understand the semitone names in the chromatic scale: The notes of C-major are considered as the basic and dominant notes, so these seven notes got identifiers from A to G. Semitones not included in this selection were then defined as pitch modifications from these basic notes; thus, the second semitone in the chromatic scale is named "one half-tone above the A", shortened to "A#" (spoken:"A sharp"), or "one half-tone below the B", shortened to "Bb" (spoken: "B flat") . These selections, however, do not necessarily contain different notes: C-major and a-minor for example contain the same notes. Here, it is important to distinguish between scale and "key". While the scale defines which notes the harmonic set of notes includes, the key defines the center of gravity, established by particular "chord progressions". Chord progressions specify an arbitrary sequence of chords (a few notes of the scale played simultaneously, e.g. three keys on the piano struck at the same time). Although they usually are played just by accompanying instruments, the sequence of chords is crucial for the general mood and tense of a song. A precise discussion of possible progressions and their emotional effects would go beyond the scope of this paper, but we recommend [26] and [38] as a profound introduction to this topic.

Major respectively minor keys are among each other quite similar or even exchangeable. It is possible to move an entire piece of music up or down in pitch to another key (a technique called "transposition"); as long as this key has the same mode as the original key, the difference will only be noticed by professional musicians or absolute listeners[24]. But, if a transposition changes the mode of the key for example from major to minor, every listener notices immediately a difference in the basic mood of the piece of music. It is important to stress that the mentioned concepts just define ratios between tones, not actual frequencies. To enable multiple instruments to play harmonically together, they need to be tuned to the same chromatic scale; since the ratios between semitones are precisely defined, it is sufficient to just specify the frequency of one note. To achieve this, musicians usually use "tuning forks", special metal forks which are crafted to resonate at a specific frequency with as few overtones as possible (thus, emitting a pure, sinusoid sound). It is common practice today to use a tuning fork which vibrates at 440 Hz as reference tone for the A4, the A above the middle C. Instruments tuned this way are called tuned to "concert pitch". However, often the tuning of actual instruments still differs by up to 15 Hz from this standard depending on the preferences of the playing musicians. Since

[22] In fact there are some slightly different "minor" scales. We neglect these differences here for the sake of readability. A good introduction can be found in [26].

[23] A major scale has these intervals in semitones: 2,2,1,2,2,2,1.

[24] Absolute listeners are people with the rare ability to identify single notes or even chords without any musical context or reference, as for example an arbitrary tone played on the piano. This ability is also called "absolute pitch" or "perfect pitch". Absolute pitch possessors are rare: According to current estimations, there is only one absolute listener among 1,000 ordinary (or "relative") listeners in the United States[40].



(a) (b)

Figure 1.9: Illustration of the noise-like sound character of percussion instruments: (a) shows the spectrum of an accelerating motorcycle (a sound some people tend to like); (b) shows the spectrum of a common tom-tom drum.

most listeners are only able to distinguish pitches relative to the musical context, these variations are generally considered to be inaudible [57].

The perception of actual music depends highly on the consonance of the played tones. "Consonance" means, as how pleasant a combination of tones is considered. Thereby, two different - albeit correlated - components of musical consonance have to be considered: Sensory consonance, a measure of the general pleasantness of any sound, on the one hand and harmony, the classical teachings of chords, keys and cadences on the other hand.

As seen on page 8, there is no clear distinction between musical instruments and generators of noise; even the timbre of solo instruments consists to a considerable degree of noise components (especially onset-sounds), while the timbre of percussion instruments differs little from noise (see Figure 1.9). However, people are capable of sorting any set of sounds on a scale from "annoying" to "pleasant". This ability can be explained by the concept of "sensory consonance", which depends on the three fundamental musical attributes:

1. "Roughness" means rapid (less than 30 ms) changes in a given signal. This may be caused by general noise or by complex sounds, whose fundamentals or overtones interfere in a way that causes a modulation ("beat"[25]) frequency between 30 and 300 Hz, played together. The intensity of perceived roughness depends on the shape of individual impulses and their frequencies. The concept of beat is capable of explaining, why some chords are perceived as consonant and pleasant while others sound tense and unpleasant: the stronger emerging beatings are the more dissonant the chord sounds.

2. "Sharpness" generally means, how well defined, that is to say, how narrow a frequency band is: A tone that contains exactly one frequency (and according overtones) is regarded much sharper than, for example, narrow-band white noise.

3. "Tonalness" describes the intensity ratio between harmonic overtones and noise-like parts of a sound.

The relative influence of these aspects of sensory consonance on the perceived pleasantness

[25]Not to be confused with rhythmic beat discussed later.

varies from listener to listener, but roughness is generally regarded as most important [56, 65].

Harmony is for the perception of musical consonance even more important. It is concerned with the typical relationships that exist between the sounds and tones of music. A basic component of harmony is the already introduced concept of "affinity of tones" (see page 13), which assigns to any frequency ratio a distinct similarity. Thereby, there are two special ratios, the octave (2:1) and the fifth (3:2), which are considered as most natural and pleasant. The other integral part of harmony theory is "root-relationship", which postulates a mutual relationship between chords and bass notes ("roots"). This can be seen as a special case of the above discussed virtual pitch, albeit root-relationship is not restricted to sounds occurring simultaneously, but can also be applied to a sequence of tones. By reducing chords to their roots and analyzing the tonal affinity of those, one may obtain a good estimation of how consonant a piece of music sounds by just looking at the sheet of music. However, the conventional harmony theory ignores timbre completely and works just with pure tones; timbre, though, can strongly affect the predicted consonance. Because of that, newer models of musical consonance take the full spectrum of overtones into account and also incorporate virtual pitches.

The just discussed methods to estimate musical consonance can be used for "vertical" (tones played at the same time) and "horizontal" (tones played sequentially as a "melody"[26]) considerations. The effects on the perceived effects can thereby be quite different: For example, while fifths and octaves played in a chord add virtually nothing to the perceived fundamentals and just melt with them, played sequentially they are considered most consonant. Also, dissonances are not automatically to avoid. In the contrary, much of the emotion and tense of a piece of music is created by skillful arrangement of passages building up tense and passages resolving this tense. Because of strong individual preferences, there is no "right" amount of dissonance in a piece of music - which can be seen from the vast amount of different styles of music with strongly varying usage of dissonance in the form of roughness, dissonant chords or melodic tense.

Songs are not only characterized by frequencies and dissonances, but are shaped strongly by the used beat and rhythm. The term "beat" describes a fundamental, regular pulse which gives a piece of music a defined basic time unit. Listeners of music tend to involuntary tap with the beat. Even if there normally exists a dominant beat, i.e., a primarily occurring time interval between two succinct sound events, there is not just one beat: If a metronome accompanying any song is set to twice its original frequency, it still ticks synchronously to the music. The beat does *not* define the shortest time interval occurring in a song (that would be the "tatum"), but usually the most frequently used time interval. In Western popular music a percussion instrument usually is played according to the beat. Tightly coupled with the beat is the "bar" in the musical notion, which defines a segment with the length of a specific number of beats. Bars are the basic time framework for actual music. Thus, melody lines or sequences of percussion sounds are usually one or more bars long. Bars are named after the amount of beats in one bar and the beat's speed: A "4/4 bar" (the by far most frequently used bar) contains 4 fairly slow, a "3/8 bar" 3 rather fast notes. The first[27] and third beat in a 4/4 bar are considered the strongest, the second and fourth beat as weak[28]. It is common practice, however, to emphasize different notes in one bar; playing "offbeat" for example is widespread and means that the weak beats two and four in a 4/4 bar are played accentuated. As how

[26] Just considering pitch changes is insufficient for the definition of melody. Melodies are also characterized by the the length of individual tones and breaks (time intervals, when the playing instrument is mute). See next paragraph.

[27] The first beat in a bar is also called the "downbeat".

[28] These are also called "backbeats".

strong the beat is perceived depends on the contrast of intensity of the beat sounds compared to the sound intensity in the same critical bands (in most kinds of music the lower bands) shortly before the sound. This is due to the fact that the signals, the hair cells send out, are stronger, when the cells were not stimulated before. The combination of these accentuations with specific lengths of tones and breaks build the "rhythm" of a piece of music. In contrast to beats and accentuations, which are common for specific styles of music, rhythms are often shared by just a few songs. The combination of rhythm and a sequence of pitch-changes (not absolute tones) is considered a "melody"[29]. Melodies are highly memorable and identify a piece of music unambiguously. The melody is mostly sung or played by a solo instrument and transports much of the emotion associated with a song.

In the following chapters, we will firstly give an overview of related work in the area of MIR and frameworks similar to AudioPhield. This will be followed by a precise description of our goals in this paper. In chapter 4, we present the general architecture of the implemented framework and describe the central algorithms of the project. Finally we will close this paper with a discussion of our achievements and results and outline our future work.

[29]This definition proved useful for observations regarding most kinds of Western music. It is, however, insufficient to cover all kinds of known melodies. In some kinds of music, "melody" (in the sense of most memorable and incisive element in a piece of music) emerges merely by varying timbres or intensities [13].

2 Related Work

As you may have seen in the introduction on psychoacoustics in the first chapter, it is a quite complex task to understand how humans listen to music. So, it is not very surprising, that MIR, the branch of research, which tries to emulate human music perception with software systems[30], is highly dynamic and difficult to categorize. This is due to the fact, that MIR as cross-discipline is provided with a constant input of new insights from other fields of research (e.g. psychoacoustics or computer science); also the rapidly increasing computational speed and storage capacities enable until recently unimaginable approaches.

Anyhow, in the following we will attempt to outline MIR in its to our work most related aspects. Finally, a short description of already developed frameworks for integration of several MIR techniques to relate pieces of music – similar to AudioPhield – closes this chapter.

2.1 MIR

Music Information Retrieval, as a scientific discipline, emerged in the late 1990s, albeit much of the basic concepts used by MIR are much older. One of the very first studies in the area of MIR (in its modern sense) was published in 1966 by Kassler [21]. In this article, it is to our knowledge also the first time the term "Music Information Retrieval" shows up in scientific literature. Kassler, as many of the researchers of his time, saw MIR primarily as an programming language problem. Thus, several programming languages, which operated on symbolic representations of music similar to classic sheets of music, were developed until the 1980s. These languages, as MIRA (Music Information Retrieval and Analysis)[63] or SML (Structured Music Language)[50], were usually created for a special purpose and found little acceptance. After the limitations of this first approach became obvious, researchers started to build their MIR-systems as integrated sets of separate tools. A good example for these is the still used "Humdrum Toolkit" by David Huron [18], which includes a collection of about 50 programs dedicated to special tasks; UNIX-like command line inputs allow the combination of these tools with each other in arbitrary order and enable the experienced user to assemble quite impressive results. The "Humdrum Toolkit", as all similar systems, still depends on symbolic representations of music; these were usually either too simplistic (and so only of limited use) or too complex; although the later ones were quite powerful (albeit mostly limited to monophonic music), the data still needed to be entered manually - a cumbersome task [9].

Parallel to this development, some researchers already tackled the problem of extracting meaningful information out of already sampled pieces of music. Indeed, already in the late 1960s important work regarding pitch detection was done by Noll [44] and others. The approaches published at this time, which were created at the beginning to enable speech recognition, experienced several refinements, like Terhardt's model of virtual pitch [66] or the inclusion of neuronal networks [54] in 1989. In contrast to the early days of pitch detection, newer concepts and algorithms were no more limited to monophonic music but addressed complex tonal signals. This spawned a new branch of research named "Auditory Scene Analysis" by the most important scientist and founder of this field of research, A. Bregman[3]. The research of pitch followed thereby always developments and observations made by researchers of psychoacoustics, a discipline, whose roots go back to the fundamental publications of Helmholtz [71] in 1885. However, the problem of pitch

[30]This definition for Music Information Retrieval is quite restrictive. There are numerous publications, which are not covered by this definition, but are accepted to belong to MIR, as the visualization of music or the fingerprinting of songs. A more general definition would be: MIR envelopes all techniques dedicated to the retrieval and usage of information extracted out of music in any form.

detection can still not be considered as solved and stays one of the major tasks in the field of MIR [13].

Another early approached, typical problem of MIR is "beat-tracking", that is to say, the automatic finding of a song's beat (see page 17 for a short introduction to the concept "beat"). Already the 1985 presented algorithm of Povel and Essens[49] could reasonable robustly identify beats in music represented as inter-onset intervals. After several years of minor improvements, the 1994 described beat-tracking model by E. Large[30] outperformed previous approaches clearly and delivered results fairly close to human perception, but was still dependent on symbolic representation of music. One of the first reasonable accurate algorithms operating on acoustic data was developed by Goto and published in 1995 [16]. Still, beat tracking, like pitch tracking, is far from solved and offers a lot of potential for improvements - especially in combination with attempts to understand the rhythm of a song [20].

With the availability of great computational power in the form of ordinary personal computers and widespread understanding of the usefulness of possible MIR applications (as introduced on page 1), the discipline Music Information Retrieval emerged in the late 1990s. From this time on, techniques and observations from the above mentioned, historically separated fields of research were combined with each other.

After this short introduction to the historical development of Music Information Retrieval (at least its facets concerned with signal processing), we will now outline its current centers of gravity, again restricted to topics related to the current stage of our project AudioPhield[31].

Seppänen and others [60] presented in 2006 an algorithmic system for beat and tatum tracking from acoustic signals of popular music. The system starts by resampling the input signal to a lower bitrate to reduce computational cost. Then, an accent filter bank splits the data into four subbands, whose width follows the logarithmically spacing of Stevens' "mel" scale [79], and computes for each subband an accent signal, i.e., a signal related to the estimated perceived accentuation. Thus, the data-rate is further reduced and the signal is prepared for the next step, the estimation of accentuation periodicity in each subband by computing the autocorrelation. Afterward, a statistical algorithm combines the separated periodicity estimations to one, before in the last stage an estimation for beat and tatum is generated utilizing musicological knowledge. While this system follows substantially the approaches from Scheirer [55] and Klapuri[24] and performs comparably well, it is by far more efficient then previous systems; in fact, the system operates in real-time on a smartphone platform. Whitely and others recently published an approach to not just extract the beat of a piece of music, but more complex rhythmic information[72] by utilizing Bayes' theorem. Thus, tempo and rhythmic patterns are considered a latent state interference problem; a sequence of observed data (extracted from a piece of music) in combination with an observation model that relates data and hidden state (or states) hint then reasonably probable estimations for the hidden state (i.e., a rhythmic pattern). Unusual is the system's capability to operate on symbolical MIDI data as well as on audio signals. Kurth uses state-of-the-art and well proven techniques in his recent work on beat-tracking[29]; he strives, however, for a goal similar to Whitely: extraction of complex rhythmic information out of music represented as audio signal. Therefore, the system – following the canonical procedure – consecutively downsamples the data, applies first a short time Fourier transform and then a comb filter bank on it, to build up "beat spectrograms", vectors containing information about resonance intensity for different periods

[31] This is a rather strong limitation, since a lot of the current research, which is concerned with MIDI-music, efficient and intuitive music-database queries, automatic playlist generation, etc., is omitted.

for short time (few seconds) windows. Afterward (and this is the novelty) residual tempo classes similar to chroma classes (see chapter 1.2.3, page 13) are generated out of the beat spectograms. Finally, the system combines these by adding tempi of the same "chroma" to create the "cyclic beat spectrum" for each time window - from which it is easy to read out rhythmic patterns or tempo information. The whole system is remarkably robust against time distortion and shows potential to outperform competing approaches.

Geoffroy Peeters from the Ircam Sound Analysis/Synthesis Team presented in [48] an improved system for the automatic estimation on keynote and mode of a piece of music. He considers this estimation as a process with three stages: 1) extraction of pitch information from the audio signal, 2) mapping this information to the chroma domain, and 3), deducing a global key from the succession of chroma vectors. Although Peeters submits also improvements for the stages 2 and 3, the main contribution in [48] regards stage 1. When extracting pitch information from polyphonic pieces of music, the problem of overlapping fundamentals and harmonics arises. Reviewing the strengths and weaknesses of state-of-the-art solutions for this problem (as e.g. [12] or [19]), Peeters proposes an approach inspired by the works of Cremer[8], the "Harmonic Peak Subtraction" (HPS) function. This function is supposed to remove the influence of harmonics in spectral representations (as obtained e.g. by Discrete Fourier Transformation, see below) of the music, leaving just the fundamentals[32]. This is done by identifying peaks in the spectrogram as candidates for fundamentals or harmonics; then, by comparing them with other peaks, a fundamental-likelihood for each peak is computed. Finally, the algorithm eliminates all peaks, whose fundamental-likelihood is too low, leaving a spectrogram that contains only fundamentals. An evaluation based on the MIREX-2005 key estimation contest (see below for further information on MIREX) revealed the remarkable performance of the system, albeit only for classical European music of selected eras. A different solution for pitch-tracking in multipitched, polyphonic audio signals was proposed by A. Cont in [6]. Her approach utilizes machine learning techniques to generate pitch templates for any instrument in a large learning-database. By comparing those templates with the spectral representation of a short section of a piece of music, the algorithm can identify the most similar templates and thus the played pitches. The system is thereby fast enough to track pitches in live played music. But it is questionable if this approach, which was seemingly only evaluated against pianos, is applicable for any kinds of instruments or music. Klapuri presented in 2006 another, but related solution in [23]: This approach also bases on the Discrete Fourier Transformation of the audio signal. Then weighted sums are computed for every frequency and their harmonics in the (previously flattened as introduced in [68]) spectrogram; thus, a "fundamental-score" is assigned to each frequency. Therefore, a machine learning algorithm, provided with training material consisting of randomly mixtured tones from different instruments, previously optimized the weight function used for the summation. Fundamentals are deduced (among other presented possibilities) from the now built "score-spectrogram" by taking iteratively the frequency with the highest "score" as fundamental followed by a cancellation of this frequency and all harmonics (according to the weight function) from the spectrogram. This method served as basis for an melody extraction approach concentrating on singing by Ryynänen and Klapuri[53]. To cope with the very versatile instrument that the human singing voice is, they utilized knowledge from acoustics and musicology. So, the algorithm takes – among other things – note-transition probabilities into account when estimating the current fundamental of the melody.

As seen above, there rarely is a "right" solution for any problem in MIR but a myriad of approaches, systems and algorithms with their individual evaluations. To make the performances of these measurable and comparable, the University of Illinois at Urbana-

[32]Hence, Peeters does not distinguish between fundamental and perceived pitch as we have done in 1.2.2.

Champaign founded the "International Music Information Retrieval Systems Evaluation Laboratory" (IMIRSEL), which originated the "Music Information Retrieval Evaluation eXchange" (MIREX) in 2005. MIREX is an annual contest of state-of-the-art MIR algorithms. The disciplines vary from year to year, but the general procedure is generally the same: A group of music experts selects or generates test data, assigns a "correct"-value to each sample (as e.g. the key of a piece of music) and a metric that is used to compute the score of an approach (e.g., a pitch-tracking algorithm that missed the correct value by exactly an octave did better than an algorithm missing the pitch by three semitones). Although there were only two contests to date, MIREX already proved to be very valuable for the whole field of science [39] and became a firm component of the most important conference about MIR, the ISMIR [36].

2.2 MIR frameworks

It became clear quite early that many of the above mentioned techniques have to be combined for some MIR-applications. Thus, many MIR systems were created as frameworks, i.e., as software-foundations, which provide the separate plug-ins, which contain the actual music processing algorithms, with a supporting environment and integrate their results. Our project, AudioPhield, can in the current state of development be seen as one of those. We will outline some of the most important frameworks related to AudioPhield, that is, frameworks that are also designed to extract various attributes out of sampled music, in this section.

The framework called "MARSYAS"[33] was first introduced in 1999 and has been an important pioneer in the field of attribute extracting frameworks since then. It is probably the most widely used system and is still maintained by its original author George Tzanetakis. Marsyas relies on a client-server architecture: All of the actual work regarding signal processing or pattern recognition is done by the server, which is written in C++; the Java-based client contains only the user interface. Despite its prominence, MARSYAS has still some stability issues and, with the current version number of 0.2.10, must be considered as premature. Also understanding and extending the quite efficient code may be difficult to novice users, especially since the current documentation could be more detailed [69, 34].

The IMIRSEL develops and deploys the most generic framework "Music-to-Knowledge"(M2K). Thereby, M2K is not a framework on its own, but a music-specific set of modules for the generic machine-learning framework "Data-to-Knowledge" (D2K). The "Automated Learning Group" at the University of Illinois built D2K as a flexible data mining and machine learning system that integrates generic data-analyzing algorithms with tools to visualize data. D2K provides the user with an IDE[34] that allows development and integration of new modules and intuitive and fast combination of these to complex data-mining sequences. The M2K-modules, which were written in Java just like D2K, were designed to create a "Virtual Research Lab" (VRL) for MIR prototyping and evaluation. In order to participate in the MIREX contest, researchers must build their approaches on top of this VRL. However, the license of D2K complicates the usage for researchers outside of the U.S.A. and prohibits the integration of D2K into commercial software [67, 10].

McKay introduced in 2005 "jAudio", a framework exclusively dedicated to extracting features out of audio data. jAudio was initially implemented as part of the "Autonomous Classification Engine" (ACE). ACE, as a musical classification system, tries to find near-optimal classification methodologies for arbitrary, supervised classification problems based on given data for example extracted by jAudio. Unlike ACE, which seems to be discontin-

[33] The initials "MARSYAS" stand for **M**usic**A**l **R**esearch **SY**stem for **A**nalysis and **S**ynthesis.

[34] IDE = Integrated Development Environment. IDEs are sets of tools for a specific purpose bundled together under a single user interface.

ued after its alpha-stage since there was no update since September 2005, jAudio is still under development and experienced some improvements in 2006. The Java-based jAudio was designed to integrate any possible feature extractor. Thus, it contains a sophisticated model of feature-types that allows the usage of virtually any data representation. Since many features depend on each other, jAudio is able to automatically identify dependencies between different extractors and resolve them by ordering the extractors accordingly. Also, jAudio makes a point of being as easy expandable as possible; thus, it is possible to add new extractors at runtime without recompiling jAudio [34, 33].

With its flexibility and its aim to extract any information out of musical data, jAudio is very close to AudioPhield in its current state of development.

3 Goals

The longterm goal of our software project, AudioPhield, is to create a novel user interface for visualization of and interaction with medium to large music databases. We want to utilize information about how similar different pieces of music sound to an arbitrary listener as foundation for the visualization. Therefore, we first need a way to measure the perceived similarity between two songs. Computing this measurement is the first stage in the development of AudioPhield and the primary goal of this project thesis.

From this, we deduced the following partial goals:

Identify perceived musical dimensions

When one listens to conversations comparing pieces of music, many statements like these are noticeable: "X has just the same bass line as Y", "X reminds me of Y with guitars added", "for me, X sounds like Y, because of the same dominant drums". These declarations suggest the existence of specific dimensions of music perception shared by the majority of music listeners. We want to find out, if these dimensions exist, and which the most important ones are.

Find and evaluate extractor-algorithms

As next step, we need to rate pieces of music in the previously identified dimensions. Therefore, we have to find and evaluate algorithms that extract information regarding one of those dimensions out of music pieces represented as audio signals, as tracks on a CD.

Find a weighting algorithm

At this point, we hope to be able to compute the distance between two pieces of music in various dimensions. Then, a way to combine theses separated similarities to one similarity value is necessary. We expect to achieve this by implementing an algorithm that computes a weighted sum over the extracted values.

Create a framework

To integrate all the above found algorithms and techniques, a framework is to be implemented. It is supposed to cover the complete process from decoding music from a sampled representation to computing the similarity value between two songs. Since it is to be expected that better algorithms for similarity computations are developed in the future, the framework has to be easily expandable for developers without detailed information about the framework's architecture and implementation. It should also be able to detect dependencies between different modules and prohibit redundant computations. Furthermore, the framework should be accessible over a graphical user interface (GUI) that allows easy selection of active modules and review of the created data. The GUI should also provide the possibility to manually change extracted values to improve the result for further computations.

4 Implementation of the Framework

This chapter presents the actually developed and implemented framework as well as already integrated plugins. Therefore, after a short discussion of some basic design decisions, we will examine the static architecture of the framework and the principal extraction process. A description of most implemented extraction-algorithms forms the main part of this section. Finally, we will close this chapter with a short presentation of the graphical user interface.

We decided to write AudioPhield in C#. This occurred for various reasons: C# as a state-of-the-art, object-oriented programming language assures type safety[35] and has automatic memory management through garbage collection. Hence, programming in C# does not only ease the process of development, since the programmer has not to be concerned, for example, with the destruction of objects, but also ensures less programming errors in the resulting code. Furthermore, C# also gives access to low-level programming: Code enclosed in an "unsafe" block is treated like plain C and may also contain typical C peculiarities like pointers or manual memory allocation. So, it is possible to optimize especially efficiency critical code sections aggressively and to reuse existing C code by just copying it. Moreover, by utilizing C# we are enabled to use powerful libraries like "BASS" (see below) for handling musical data or "(Managed) DirectX" for smooth graphics in a later state of the project. We believe, this cancels out the fact that programs written in C# are currently bound to the Windows platform. Finally, using C# in combination with the rich .Net-Framework and Microsoft's IDE "Visual Studio" proved to be a powerful combination (see [37] for further information about C# or the .Net-Framework).

As already mentioned above, we use the "BASS" audio library via the "BASS.Net"-wrapper to handle musical data. With "handle" we mean especially decoding and playback of compressed music. While the core-package of BASS already supports various file formats like Wave, MP3 or Ogg Vorbis, with the inclusion of the many available extensions virtually every existing digital representation of audio can be processed. BASS comes normally as a set of DLL-files with accompanying headers as API for programs written in C/C++. Although we could link AudioPhield directly to these files, we decided to use the wrapper "BASS.Net" instead: In doing so, we access BASS as if it was written in C# and gain also some error avoidance potential, because the BASS.net layer introduces stronger typing, for instance by encapsulating related constants in separate enums. BASS proved to be very fast, robust and easy to use at the same time: A few calls to BASS' procedures are sufficient to initialize the library and decode or playback music files - completely hiding the underlying complexity of various encodings, bit-rates or sample formats. Also, BASS is free to use for non-commercial purposes [70].

In the course of the development process it became clear that our initial goal, to provide single similarity values between two pieces of music, is not actually feasible. It would be necessary in this approach to re-extract all attributes from all songs, just to add one song to the database. Some of the extraction algorithms, however, are very compute-intensive – reprocessing the whole collection of songs could take weeks. Instead, we decided to postpone the problem of weighting and computing similarity values to the next stage in the development of AudioPhield, the visualization. Consequently, the new goal of the analysis framework is to generate a file containing the results of all extraction algorithms to each song for further precessing steps. To simplify the storage format and later computations,

[35] "Type safety" is not a well defined term. We understand a type safe language as a language with a complete type system; every variable has to have a type and can contain only data of this type - albeit explicit casts may be allowed. Thus, unintended assignments as program errors can be found at compilation-time or even run-time.

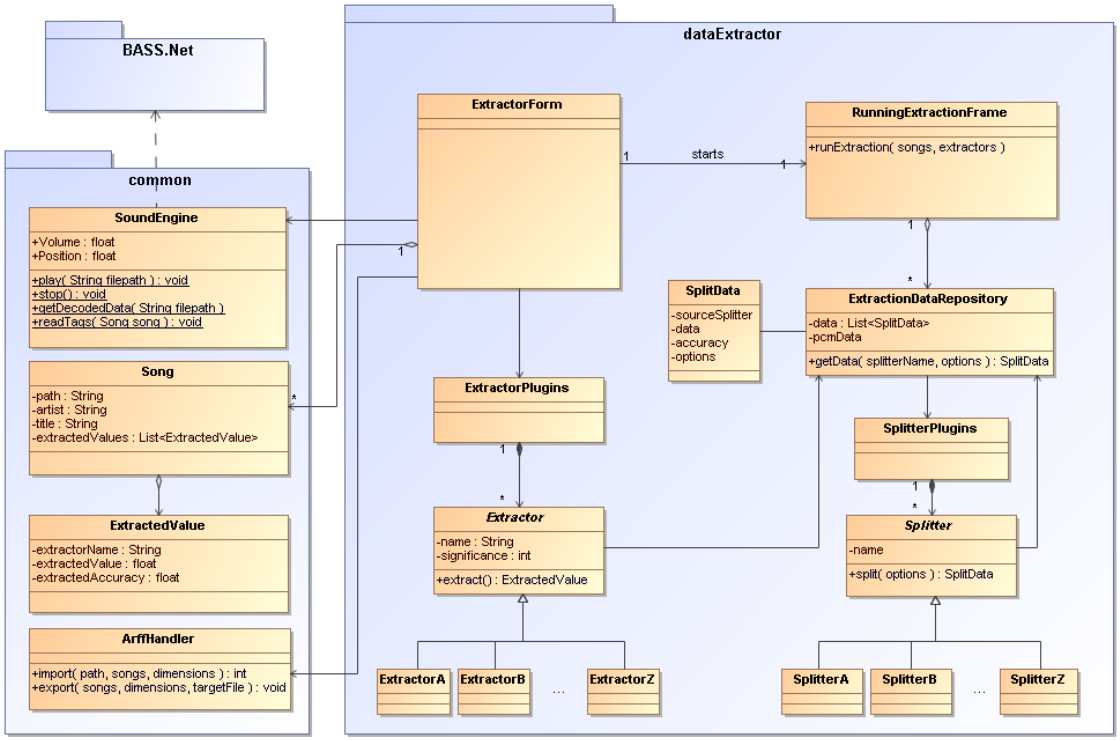


Figure 4.1: Class diagram of AudioPhield. Depicted are just classes of major importance for the extraction process.

we decided to limit the allowed output of an extraction algorithm to a single floating point value.

4.1 Architecture

This section describes the general structure of the framework in two steps: Firstly, we will examine the static architecture of AudioPhield by taking a look at the classes most important to the extraction of attributes. Secondly, we will outline the extraction process as a sequence of responsibilities of these classes.

In the following, you will find a list of the most important classes of the AudioPhield music analysis framework, as they occur in the UML class diagram in Figure 4.1, and a short description of their responsibilities.

Classes is in the package **common** are expected to be used in later implementation steps of AudioPhield as well:

- **SoundEngine**
  This class handles everything concerning audio files, especially the decoding of compressed audio to an uncompressed array of "float" values representing the actual audio signal. It does so by invoking methods of the BASS library.

- **Song**
  Objects of this class act primarily as storage objects. They contain besides generic fields like "artist" or "title" a list of all already extracted attributes (as "ExtractedValue"-objects).

- **ExtractedValue**
  This is a simple storage class to hold the output of Extractors. Note the attribute "extractedAccuracy". The purpose of this field will be discussed later in this chapter.

- **ArffHandler**
  This class is responsible for saving the database of Songs with their ExtractedValues to disk. Therefore it uses the "Attribute-Relation File Format" (ARFF) as defined in the "Weka Toolkit" for reasonable compatibility of the data with Weka and other machine learning toolkits (see [81] for a profound introduction to Weka).

Classes in the package **dataExtractor** are exclusively concerned with the extraction process:

- **ExtractorForm**
  This is the central class for managing the extraction process. ExtractorForm contains the main graphical user interface (GUI), which allows simple and intuitive selection of songs to be considered and attributes to be extracted as well as explicit manipulation of these values (see section 4.3 on page 46 for a more detailed description of this GUI). Also, the single instance of this class holds the complete data and state of the system at runtime.

- **RunningExtractionFrame**
  This class controls the extraction process: It causes every (selected) Extractor to operate on each chosen song and gives visual feedback about the extraction's progress.

- **ExtractionDataRepository**
  Objects of this class handle all data accumulated during the extraction process of one piece of music, as the basic audio signal or data produced by Splitter-algorithms. They do this on demand, i.e., SplitData (see below) is not generated until an Extractor specifically calls for it. Then, this data is stored for later reuse. Thus, the ExtractionDataRepository prohibits unnecessary and redundant computations. Also, since Splitters may depend on other splitters, this mechanism resolves the ordering problem in an easy way[36].

- **Splitter**
  This is the abstract superclass, every actual splitter-algorithm has to inherit from. Inherits of this class ("Splitters") can be seen as intermediate algorithms that process the musical data in any way without immediately producing attribute values. Splitters may depend on each other. See chapter 4.2 for a list of already implemented Splitters.

- **SplitData**
  Instances of this class are simple storage objects to hold the output of actual Splitters. They contain - besides the actual data - information about the originating Splitter as well as an estimation of how valid the data is.

- **SplitterPlugins**
  This class is one of the two extension points of the framework. It manages a list of available Splitter plug-ins as central registration for those. Splitter implementations registered in this class are accessible throughout the framework.

- **SplitterA ... SplitterZ**
  These are dummies representing real Splitter plug-ins to clarify the plug-in system.

- **Extractor**
  This is the abstract superclass, every actual Extractor has to inherit from. Inherits

[36] However, there is currently no way to prevent dead-lock causing circular dependencies. This is, in our opinion, no big problem, since these dependencies should be known at the time of development

of this class ("Extractors") compute single attribute values either directly from the audio signal or from data generated by Splitters.

- **ExtractorPlugins**
  This class is the second extension point of the framework. New Extractors can easily be integrated in the framework by registering them in this class.

- **ExtractorA ... ExtractorZ**
  Dummy Extractors to clarify the plug-in concept.

To further explain, how the extraction process in AudioPhield works, we will now describe the typical course considering a simple example schematically illustrated in Figure 4.2. In the description of the example, we will refer to this figure by adding the numbers of the messages currently explained in braces.

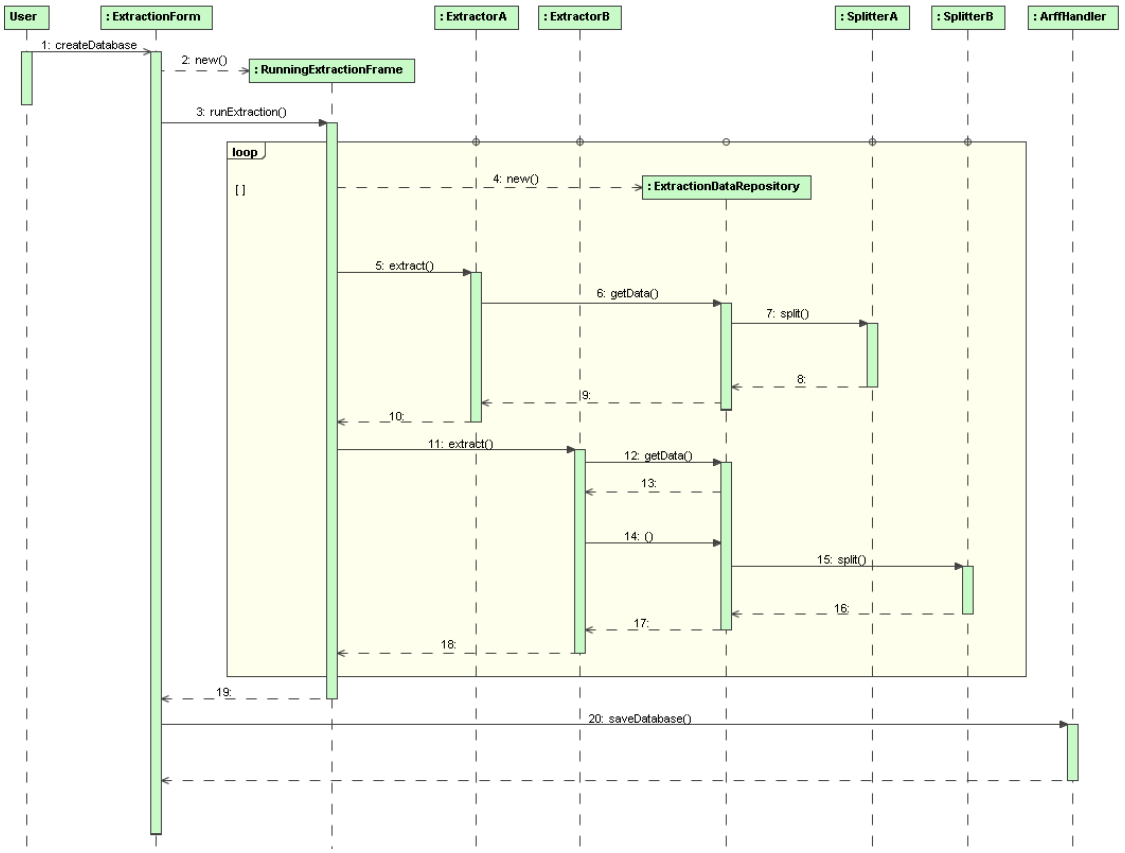


Figure 4.2: Sequence diagram in UML-notation illustrating the functioning of AudioPhield. Depicted is a simple example with two Extractor plug-ins, "ExtractorA" and "ExtractorB", depending on data from the two Splitter plug-ins "SplitterA" and "SplitterB".

Firstly, by utilizing the GUI provided by the ExtractionForm the user creates a list of pieces of music to be considered and selects the attributes that the framework is to extract from them. Then, she initiates the extraction process (1). Now, the ExtractionForm creates a new RunningExtractionFrame (2) as controller of the actual extraction and causes it to start the computation by calling its "runExtraction" method (3). The RunningExtractionFrame instance is thereby also given the lists of chosen songs and attributes. After that, the RunningExtractionFrame creates an ExtractionDataRepository object to hold all accumulated data concerning the analysis of the current song (4). In its constructor

routine, the ExtractionDataRepository object causes the SoundEngine to decode the current music file and stores the resulting PCM-data (not in the diagram). Afterward, the RunningExtractionFrame calls the first chosen Extractor, ExtractorA, to process the song (5). ExtractorA can not operate directly on the audio signal but depends on the output from SplitterA. Therefore, ExtractorA asks the ExtractionDataRepository to deliver the necessary data by calling the "getData" method (6). The ExtractionDataRepository verifies that the requested data was not generated before and calls SplitterA to compute the needed data (7). After storing the return from this splitter (8) for possible later use, the ExtractionDataRepository hands it over to ExtractorA (9), which then finishes its computations and transfers the extracted attribute – as return value – back to the RunningExtractionFrame (10). Now, the RunningExtractionFrame calls the next chosen Extractor, ExtractorB, analog to ExtractorA (11). ExtractorB depends on SplitData delivered from SplitterA and SplitterB. Therefore, it consecutively calls the ExtractionDataRepository for this data (12, 14). Since the SplitData from SplitterA is still stored in the repository, it is returned without further computations (13). The requested output from SplitterB, however, was not computed before; so, the ExtractionDataRepository operates analog to before, i.e. calls for the data (15), stores the return (16) and transfers it (17). At this point, ExtractorB completes its task and returns another extracted attribute value (18). With that, the processing of the first song is finished. If there are more pieces of music to analyze, the RunningExtractionFrame repeats all steps from (4) to (18) for each of these songs. Then, after the last song was analyzed, the RunningExtraction frame returns the music collection to the ExtractionForm (19), which, after some integrity checks, hands it over to the ArffHandler by calling its "saveDatabase()" method (20). When this is done, the extraction process is complete.

The Extractors and Splitters in this example obviously only serve as dummies for real implementations. Also, it should be clear that the framework is not limited to two Extractors and Splitters, but can contain many more. To prohibit the example from becoming too complex, we omitted the possible dependencies between different Splitters. Thus, a called Splitter (as in (7)) may not directly return results, but call the ExtractionDataRepository's "getData()" method several times, to obtain necessary data generated by other Splitters, first.

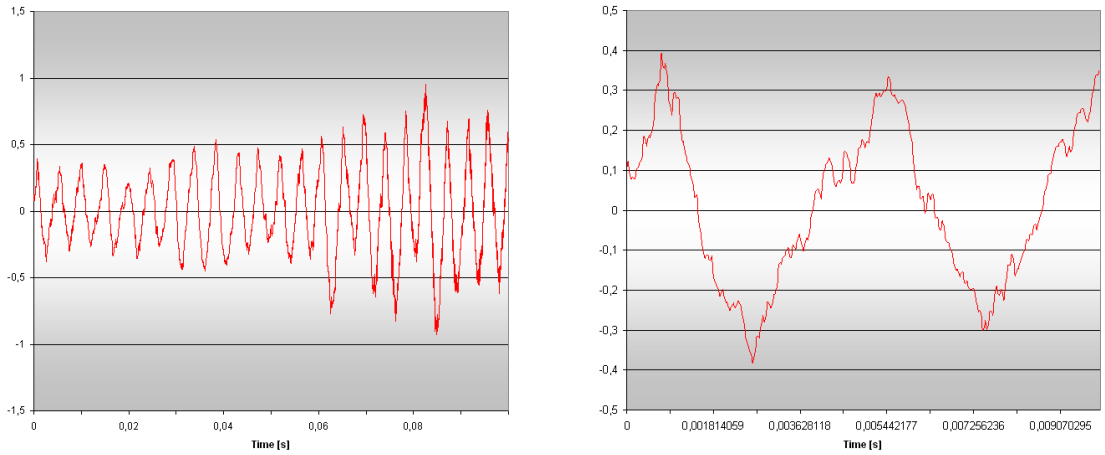
4.2 Splitters and Extractors

This chapter contains an introduction to the Splitters and Extractors already implemented and integrated into AudioPhield. Thereby, we will first explain the basic concepts of a Splitter and then discuss Extractors depending on the output of this Splitter. In the case of Extractors building on top of several Splitters, these Extractors are presented not before all associated Splitters have been introduced.

4.2.1 PCM and RMS

The generic representation of digital music is the representation as PCM-data. PCM (which is short for "Pulse-Code Modulation") is a way to digitize analog signals by sampling the magnitude of the signal at uniform intervals followed by a translation of these magnitudes to discrete symbols. This means, in the case of music, that sound pressure fluctuations at a given point are measured repeatedly (usually at 44,100 Hz[37]) and saved as a sequence of

[37]The 44,100 Hz were chosen to ensure that the encoded signal contains all audible frequencies: According to the Nyquist-theorem, the highest reproducible frequency is half the sampling rate. So, at a sampling rate of 44,100 Hz frequencies up to 22,050 Hz are encoded. This is higher than the highest frequency most people can hear (see section 1.2.2).



(a) (b)

Figure 4.3: Typical PCM data. (a) shows data representing 100 ms of a female singing voice accompanied by a piano; (b) depicts the first 10 ms of (a)

binary values (most commonly using 16 Bits to encode one sample). PCM data can also be roughly interpreted as the amplitude of a loudspeaker's membrane when playing back music (see Figure 4.3 for an example of PCM represented music).

Although PCM data is frequently used in the literature (see, e.g., [34]), we found little significance in algorithms deducing perception information from this data. The implemented "ZeroCrossingsExtractor", an algorithm that counts how often the signal averagely crosses the zero line in a given length of time, produced useless, nearly random results and was therefore deleted from AudioPhield. However, PCM data is the foundation for all Splitters.

As stated above, PCM samples contain information about the intensity of a signal: The higher the magnitudes encoded in consecutive PCM samples, the more intense the signal is. In the domain of audio, "intensity" can be understood as loudness. To measure this loudness, the so called "root mean square"[38] (RMS) is computed. It is calculated by extracting the square root from the mean of the squares of a series of n discrete values $x_1, x_2, \ldots, x_n$:

$$x_{rms} = \sqrt{\frac{x_1^2 + x_2^2 + \cdots + x_n^2}{n}}$$

The RMS-Splitter is based exactly on this formula. Thereby, the Splitter does not compute the RMS for all PCM values at once (what might yield little information), but can be instructed to use "windows", short series of consecutive samples, of various widths. Hence, the overall loudness of the signal at an arbitrary point can be extracted (see Figure 4.4).

There are two Extractors using only RMS data:

- **StaccatoExtractor**
  This simple Extractor computes the variance[39] of RMS data with a window width

[38] The root mean square is also known as "quadratic mean".

[39] "Variance" is a widely used method to measure, how strongly a set of values varies. It is computed as the average of the square of the distance of each value from the mean:

$$x_{var} = \frac{1}{n}\sum_{i=1}^{n}(x_i - \mu)^2, \text{with } \mu = \frac{1}{n}\sum_{i=1}^{n} x_i$$

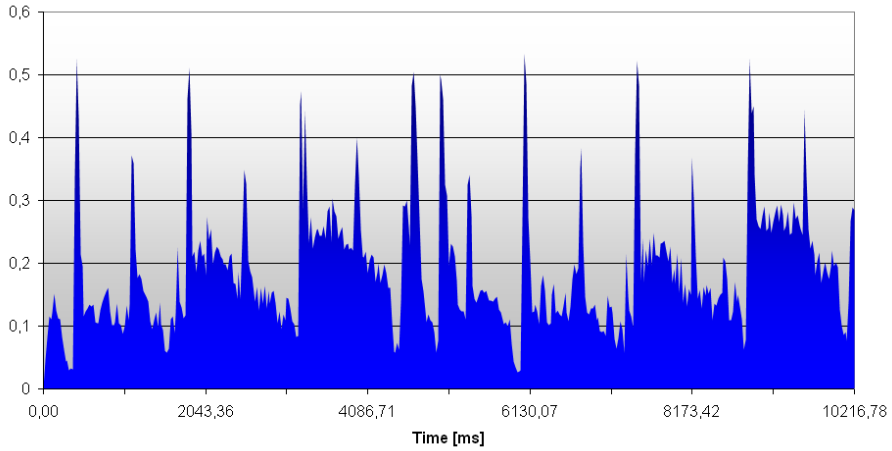


Figure 4.4: Illustration of RMS data. The diagram depicts values created by calculating the RMS of windows with a size of 1024 samples. As source for the data served a highly rhythmic Soul song.

of 2048 samples (this covers about 45 ms). A high variance is supposed to indicate music with strong percussion parts or instruments playing staccato – in contrast to more spheric sounds or instruments played legato with a low variance value.

- **BpmExtractor**
  This is a simple beat detection algorithm, which also operates on the RMS values calculated over 2048 samples. It compares one RMS value with the average of a local working set (e.g., 40 RMS values, which meets roughly 2 seconds) of previous RMS values. If this value exceeds a threshold (e.g., 1.3 times the average), the corresponding time slice is considered to contain a beat. If a beat was found, the algorithm considers the following RMS values as "masked" and does not look for new beat events until the values fall below another threshold (e.g., 0.9 times the average). The algorithm counts this events and calculates their number per time unit as return value. The name of the Extractor may be somewhat misleading, since it does not estimate the perceived beat, as described on page 17, but merely counts the frequency of comparatively loud events. This result, however is at least related to the beat and speed of a piece of music.

4.2.2 From DFT to STFFT

Information about the loudness at certain points in a piece of music is insufficient for most music analysis considerations. Instead, knowledge about the frequencies contained in the audio signal would be necessary to estimate important factors as perceived pitch or timbre. To achieve this, that is to say, to transform the audio signal from the time domain to the frequency domain, we use the "Fourier transform": As Fourier showed, any periodic function can be decomposed into a sum of weighted sinusoidal component functions (the "spectrum") with the weights, or coefficients, indicating how strong a component function's influence on the original function is (see Figure 4.5). So, if the audio signal is assumed to be roughly periodic, we can use the Fourier transform to estimate the strength of single frequencies in a piece of music. In the case of sampled audio data, the transform is called "discrete Fourier transform" (DFT), because the original periodic "function" consists of discrete data. The result of this DFT is a vector of coefficients corresponding not to arbitrary frequencies, but to discrete points in the frequency domain, whose mutual distance depends

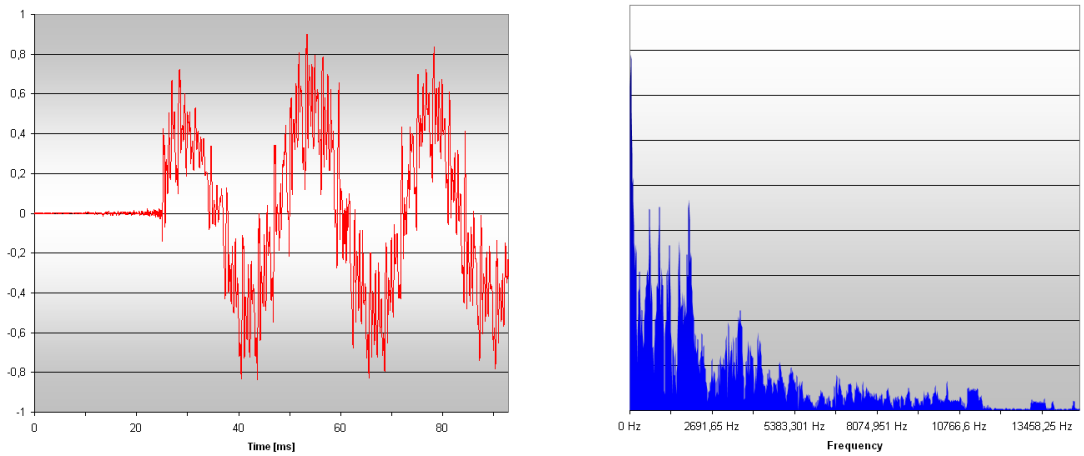


Figure 4.5: Illustration of the Fourier transform. Depicted is the same audio signal of 4096 samples represented in the time(left) and the frequency domain (right).

on the sample rate of the original data and the observed amount of samples. For example, 1000 samples, representing one second of music, are transformed by the DFT to 1000 coefficients representing strengths of frequencies separated by 1 Hz[40]; would these 1000 samples represent only 10 milliseconds, the resulting frequencies, now separated by 100 Hz, would be part of a broader spectrum with a lower frequency-resolution. The formula to calculate the DFT's N coefficients, $X_0, X_1, \ldots, X_{N-1}$, from the original N samples $s_0, s_1, \ldots, s_{N-1}$ is as follows:

$$X_k = \sum_{n=0}^{N-1} s_n e^{-\frac{2\pi i}{N} kn}, \text{ with } k = 0, 1, \ldots, N-1$$

For a better understanding of how this works, one might imagine the process as taking a sinusoid curve, multiplying it onto the signal and interpreting the result as how good these match. Note, that the transformed values, as the original samples, are complex numbers. While the imaginary part of the s_k values is always 0, the X_k values contain information about the strength of a sinus and its phase offset combined in the real and the imaginary part. So, in order to calculate the influence of a frequency to the audio signal, we need to compute the absolute value of a complex coefficient X by applying the following formula:

$$|X| = \sqrt[2]{X_r^2 + X_i^2}, \text{ with } X_i \text{ as imaginary and } X_r \text{ as real part of } X$$

The above described process is implemented in the DFT Splitter. Although the frequency analysis of a whole piece of music may provide some useful information, knowledge about the spectrum at a certain time in a song is preferable. Therefore, the Splitter analyzes short windows of samples consecutively; elements of the resulting series of spectra are also called "frames". However, it is not feasible to just take series of samples out of the PCM data; the hard cutoff at the edges of such a snippet would cause the result to be riddled with artifacts. To prevent this, a windowing function is necessary. The DFT Splitter uses therefore the "Hamming window" (see Figure 4.6 for an illustration) with the following function:

$$w(n) = \alpha - \beta \cos(\frac{2\pi n}{N-1})$$

[40]The highest possibly encoded frequency in this example is according to the Nyquist theorem at 500 Hz. Thus, it is sufficient to only take the first 500 coefficients for further considerations, because they contain already all usable information.

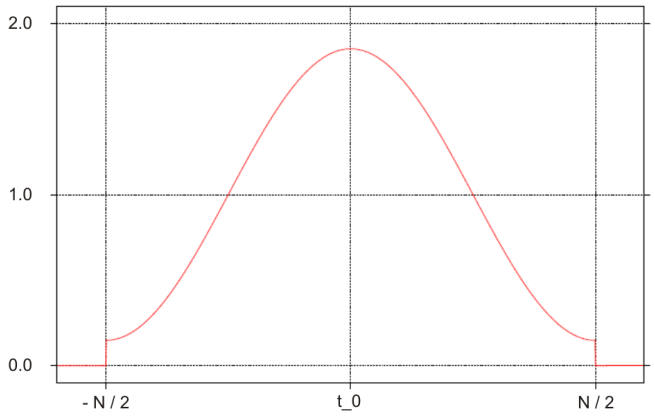


Figure 4.6: Illustration of a Hamming window at the time t_0 with a width of N.

Experiments showed, that 0.53836 for α and 0.46164 for β produce best results. So, before a window is transformed, each sample is multiplied with a factor taken from this window function. Since wider windows enable higher frequency resolution at the cost of time resolution, a compromise suitable for the individual application has to be found. The appliance of the windowing function requires consecutive frames to overlap (usually by half the window size) to prevent loss of information.

As discussed in chapter 1.2.2, the human ear is not equally sensitive for different frequencies. Manufacturers of modern music equipment - especially loudspeakers and hi-fi systems - know about this peculiarity and build their products to even it out. This, however, only ensures that sounds of different frequencies sound about equivalent loud - but it does not reflect the focus of the human audio perception system on specific regions of the audible frequencies. Since we want the Splitter to imitate the human hearing process, we have to include this feature in the algorithm. On this account, we multiply the resulting coefficients with factors calculated from the dB(B) weighting curve (see Figure 1.3 for an illustration). The decision to use the weighting curve B, which is best used for low to medium volume sounds, is based on the assumption that music is usually enjoyed at a volume of about 60 dB(SPL) - what renders the B curve as the best matching one. The specific factor for a frequency f is calculated according to the this function[41]:

$$G_B(f) = \frac{1}{\alpha}\left(\alpha - 1 + \frac{f^3 \times 5.9918 \times 10^9}{(f+129.4)^2(f+995.9)(f+76655)^2}\right)$$

The weighting factor α defines the influence on the previously computed spectra. Tests showed, that a value of 15 seems to work fine for our purposes. Now, the output of the DFT splitter is already much closer related to the "output" of the human ear. But there is another peculiarity of the ear to be considered: It reacts roughly logarithmically on different sound intensities (see page 4). To imitate this behavior, the DFT calculates finally the logarithm of each value. Recapitulating, the major processing steps of the DFT Splitter are:

1. Divide the PCM data into windows of a specific length
2. Apply the Hamming windowing function on the windows
3. Compute the discrete Fourier transform

[41]The function is an approximation to the experimental data. The used numbers match the average human listener, but most likely not the individual (see page 6).

4. Multiply the coefficients with dB(B)-factors
5. Calculate the logarithm of each coefficient

Unfortunately, the DFT Splitter proved to be extremely CPU-intensive; the time consumed by the computations exceeded on some machines the playback time of a piece of music. This is caused mainly by the third step, the computation of the Fourier transform. A procedure called "fast Fourier transform" (FFT), which simplifies the complex calculations to a simple bit-reordering, can reduce the complexity of this step by magnitudes. The "short time fast Fourier transform" (STFFT) Splitter incorporates the FFT by utilizing a modified form of the Cooley-Tukey algorithm. The structure of this algorithm, however, limits the STFFT Splitter to windows, whose size is a power of two. Otherwise, the STFFT Splitter works analog to the DFT splitter. Since some of the topics regarding these Splitters were merely touched in this section, we recommend [47] for an exhaustive introduction to DFT, FFT and windowing functions.

Data produced by the DFT or STFFT Splitters are the basis for several Extractors concerned with general properties of a piece of music:

- **AutoCorrBPM**
  This Extractor uses STFFT data with a window width of 2048 samples. Therefore, it firstly identifies a frame containing as much "energy" as possible. To compute this energy, the algorithm simply adds up all coefficients of spectrum that lie within a certain interval of bands (5 to 500 seemed to work fine). This limitation is supposed to speed up the computations and focus them on bands that typically contain the frequency spectra of percussion and bass instruments. A frame containing much energy in comparison to close neighbor frames are assumed to hold a beat (similar to the BpmExtractor above). Then, the Extractor computes the correlation, the average of the differences between the coefficients representing the same frequencies, of the found frame and nearby frames. Since the beats-per-minute of common music virtually never leaves the range from 45 to 200, it is sufficient to only include frames of this distance in the calculations. The frame with the best correlation to the original frame is then considered as repetition of the first frame, and so as the next beat. To improve the result, the Extractor repeats this process several times, deletes some outliers and takes the average of the remaining frames as return value. The advantage of this AutoCorrBPM Extractor over the BpmExtractor is that it does not just take high energy as beats, but differentiates between these beats according to their correlation. The result should therefore be closer to the actual beat of a song.

- **CentroidFrequencyExtractor**
  This Extractor tries to find the center of gravity regarding frequencies of a piece of music. It relies on STFFT data with a window width of 4096 samples. The algorithm firstly computes the overall energy contained in a spectrum before stepping up from the lowest band until half of the contained energy is collected. When the Extractor reaches this threshold, the corresponding band is regarded as center of gravity in this spectrum. The average of these M values $c_0, c_1, \ldots, c_M - 1$ is then combined to the return value $c_\emptyset$ by finding the arithmetic mean of them:

  $$c_\emptyset = \frac{1}{M} \sum_{i=0}^{M-1} c_i$$

  The result of the computation is the higher the more instruments play at the same time. This is due to the higher amount of overtones found in the upper bands. Thus, the CentroidFrequencyExtractor helps estimating how "rich" a song sounds.

- **LowAmplitudesExtractor**
  The LowAmplitudesExtractor is supposed to estimate, how much "staccato" character a piece of music has - similar to the already mentioned StaccatoExtractor. The difference is, that this algorithm limits its computations to an interval of low bands. So, the result should be closer related to percussion instruments with low pitches.

- **CleanOvertoneExtractor**
  This is the first Extractor that incorporates knowledge about tones and overtones. It basically compares the overtone spectrum found above the predominant frequency with a previously defined template. Therefore, it takes STFFT data of windows with a length of 4096 samples and identifies the strongest frequency in the frequency region that resembles the "dominant region" (see page 9). This frequency is interpreted as fundamental frequency of the predominant instrument in this frame. To gather knowledge about how "pure" this tone sounds, the algorithm creates a template of an idealized overtone spectrum like the one shown in Figure 4.7. It takes thereby the intensity of the found fundamental as reference for the template's fundamental and calculates the strengths of the other bands included in the template accordingly. Then, the Extractor computes the average of the differences between the actual data and the template; bands that do not show up in the template spectrum are thereby ignored to minimize the influence of interfering instruments. The average of these averages (with some minor statistical corrections) is the return value of the Extractor and showed to be closely related to the perceived "cleanness" of a song. Unfortunately, it also proved to be little robust against highly polyphonic or a capella music.

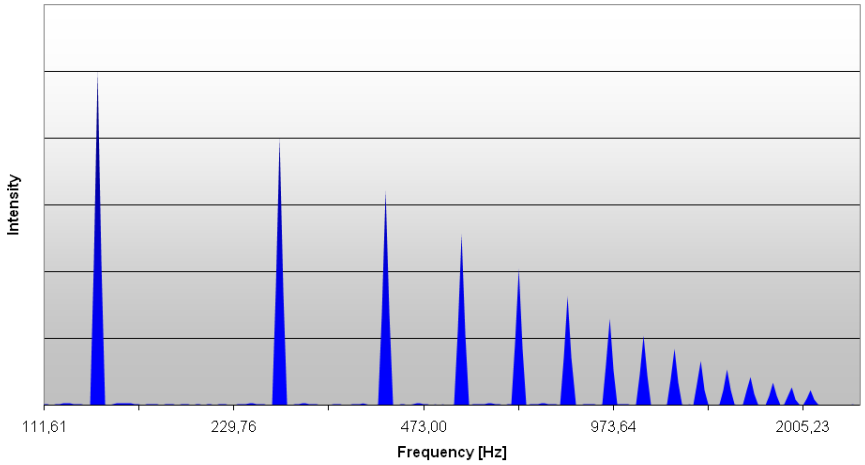


Figure 4.7: Illustration of a CQT analysis of an idealized overtone spectrum. Note the logarithmic frequency-scale.

- **OvertoneStrengthExtractor**
  The OvertoneStrengthExtractor works basically as the CleanOvertoneExtractor above. It does, however, not analyze how closely the found fundamental resembles a given template, but computes the general strength of possible overtones compared to the fundamental. The resulting value is heavily influenced by the timbre of the instrument playing the predominant melody and thus delivers valuable knowledge in similarity computations.

- **SoundCompactness50 / SoundCompactness80**
  These Extractors compute the width of an interval of bands in STFFT data containing 50 / 80 percent of the spectrum's intensity. They do so by identifying the

strongest band (in a limited frequency region) in the spectra delivered by the STFFT Splitter. This band is the start band and the initialization value of two variables l and r. The algorithm then iteratively increases r and decreases l and adds the values in the corresponding bands to a central collection variable until this variable reaches the threshold of 50 (or 80) percent of the accumulated intensity in the analyzed spectrum. The difference between the r and l values averaged over all frames is then condensed to the return value. The results are quite useful to measure the perceived "richness" of song.

- **StrongestBandVariance**
  This is a rather simple Extractor, which computes the variance of the strongest bands. Therefore, it uses STFFT data built from windows with a width of 4096 samples. The algorithm identifies the strongest band in a limited frequency region in each frame and stores them in an index vector. Then the variance (as explained above) is computed from this vector and given back as return value. The result indicates, how "monotone" a song sounds, but is only reliable under rather monophonic circumstances.

4.2.3 CQT and ACQT

Although DFT and STFFT reveal interesting insights into the frequency distributions of a song, they still have a major fault when they are used to imitate the human hearing process in the cochlea: The width of all bands in the spectrum is identical, i.e., the range of analyzed frequencies is subdivided into intervals of the same length. The cochlea, in contrast, has different resolutions in different parts of the frequency spectrum; it can be considered to subdivide the spectrum in intervals of logarithmically increasing widths (see chapter 1.2.2 for further details). Also, the width of the windows used as basis for the transformations is the same for each frequency. This number of samples determinates the frequency resolution and the time resolution of the transformation. However, different frequencies require different window widths: In low frequencies the frequency resolution needs to be high (the difference of two tones only separated by few Hertz may be audible), while the time resolution is of less importance, because the ear needs a low pitched tone to sound for a specific time to recognize its pitch. Furthermore, bass instruments rarely play fast pitch changes - and when they do, the result is usually perceived as "blurred". So, low bands require wide windows. In contrast, time resolution is of far more importance than frequency resolution on the other end of the spectrum, since tones separated by even hundreds of Hertz may sound the same and most fast pitch changes can be found here. Thus, narrow windows are preferable in high frequencies. The fixed size of the window in the DFT/STFFT Splitters is therefore inapt for the whole spectrum.

J. Brown [4] revealed 1991 a solution to these problems: The "Constant Q Transform" (CQT)[42]. The CQT is, asthe DFT, a transform from the time into the frequency domain. But, unlike the DFT, the CQT does not use constant time or frequency resolutions; instead, the width of the analyzed bands increases with rising frequencies. This is achieved by a fixed ratio of the center frequency of a band to the bandwidth: The factor Q. With this ratio, the formula of the CQT can be written as a modified DFT:

$$X_k = \frac{1}{N_k} \sum_{n=-\lfloor N_k/2 \rfloor}^{\lceil N_k/2 \rceil - 1} w_{k,n} s_n e^{-\frac{2\pi i}{N_k} Q n}, \text{ with } k = 0, 1, \ldots, N-1$$

This term already contains a window function $w_{k,n}$. This is necessary since the window width is variable and thus the window function has to be modified accordingly. The quality

[42] Brown fails to explain the "Q" in the title of her transform. Cited publications support the assumption, that the Q is short for "quality factor".

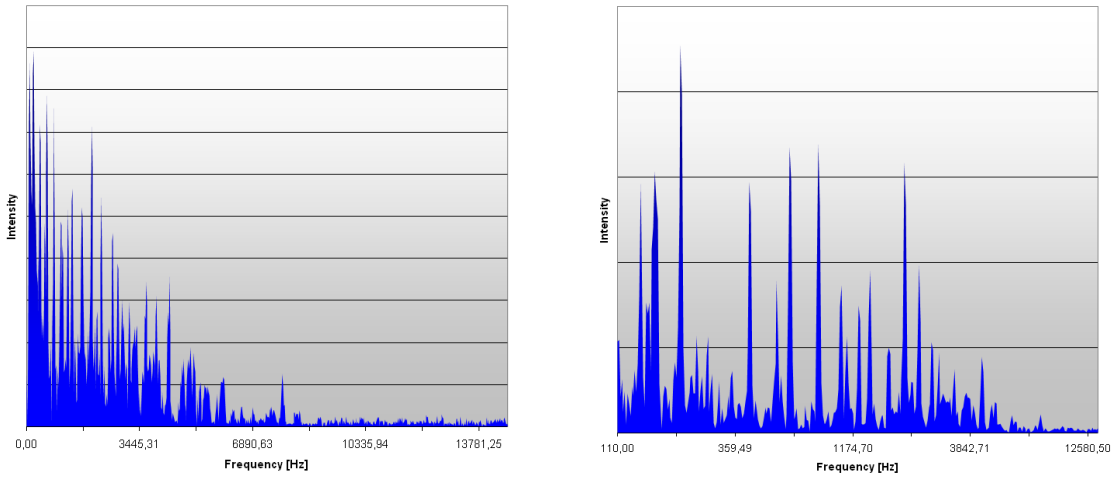


Figure 4.8: Comparison of the results of the FFT (left) and CQT (right). The FFT was calculated using the STFFT Splitter with a window size of 2048 samples.

factor Q and the window width for the currently observed frequency N_k are calculated depending on the desired resolution:

$$Q = \frac{1}{2^{1/(12m)}} \ , \ N_k = \frac{N_0}{2^{k/(12m)}}$$

The parameter m specifies the number of bands computed per halftone, i.e. the band resolution per octave; its value is usually 2 or 3. N_0 means the basic (and biggest) window width, which is the window width for the first coefficient and lowest frequency. So, whereas in the DFT the parameter k defines the number of cycles of the sinusoidal component function used in the fixed window, in the CQT the number of cycles is fixed (and given by Q) and the size of the window adapted.

We implemented this procedure in the CQT Splitter (see Figure 4.8 for a comparison of the results of the DFT and CQT). Additionally, the results of the CQT transform are adjusted analog to the DFT/STFFT by applying weak dB(B) factors and logarithmic calculus. The varying window size, however, has one problem: To not loose any tonal information, it is necessary to progress from frame to frame with the shortest used window width, which is - assuming eight computed octaves - just an eighth of the biggest used window. This means that the frames overlap strongly at low frequencies. Since these overlaps do not generate any new insights, these CPU cycles are wasted. The alternative would be to proceed with a faster pace – which instantly causes parts of the audio to be excluded from the analysis. Therefore, we implemented the "Adaptive CQT" (ACQT) in the ACQT Splitter as suggested by Stigge in [62]. In this modification, different progression rates are used for each octave: An octave's progression rate is defined by the width of the biggest (and therefore lowest) window in the octave. Thus, since the frequency doubles each octave, the window width halves. Assuming the computation of three octaves this means that the algorithm computes twice as many frames for the second octave as for the first octave and four times as many four the third octave. This modification minimizes both, redundancy and loss of tonal information - and produces frames that differ little from the CQT with maximal overlaps (see Figure 4.9).

The SplitData produced by the CQT and ACQT Splitters is the foundation for several Extractors:

- **CondensedNoiseExtractor**
  The CondensedNoiseExtractor condenses the result of the ACQT to one octave by

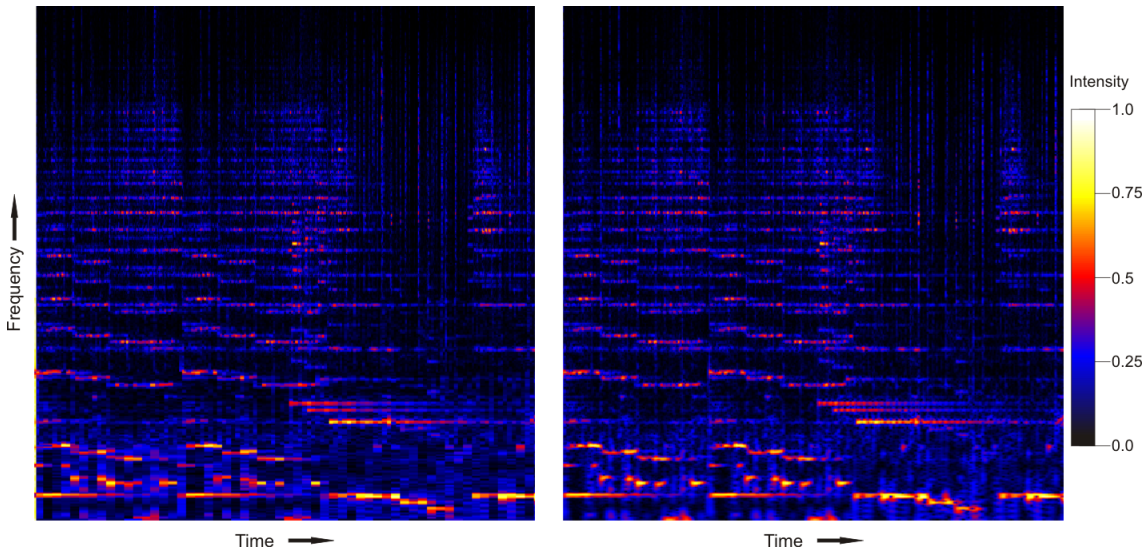


Figure 4.9: Comparison of the results of the ACQT (left) and the CQT (right). Both diagrams show analyses of the same, about 18 seconds long, snippet from a Trip Hop song.

simply adding all coefficients of the lower three octaves that are exactly one octave apart from each other. Then, the variance of these compressed values is calculated and given back as return value. This value is related to the perceived "pureness" of a song: The higher the variance (i.e. the higher the differences between strong and weak bands), the more should corresponding tones dominate the perception, and the less noisy distortions disturb the sound characteristics.

- **MelLengthExtractor**
  This Extractor is supposed to measure, how long periods of uninterrupted melody tones on the average are, that is to say, how much time lies between two pauses of the melody instrument. To compute this, the algorithm firstly computes the mean intensity of the higher octaves in the ACQT SplitData. All frames with an intensity above this value are then considered as part of the melody. Afterward, the Extractor computes the average length of episodes between regions of low intensity, which are interpreted as pauses. The MelLengthExtractor showed little robust against polyphonic interferences, but may become quite valuable in combination with the F0 Splitter or Melody Splitter(see below).

- **MelodyTrendExtractor**
  Some moods in music pieces are created by a dominant pitch trend: Lullabies, for example, usually show a distinctive down-trend. Melodies with raising melodies, in contrast, often sound uplifting or happy. The MelodyTrendExtractor was created to gather this kind of information. Therefore, the algorithm has firstly to decide, which frames are part of the melodic segments in a song. This is achieved by calculations based on the intensity contained in a frame. For the selection a growing approach is utilized: The algorithm considers all frames whose intensity is above a specific level as part of melodic segments of music. This threshold is rather restrictive at the beginning but is relaxed iteratively to take more frames into account until a specific minimal percentage is reached (e.g. 40%). Then, we estimate the predominant tone (which most likely is closely related with the perceived pitch) by finding the biggest coefficient in the first four octaves of the spectrum of each frame. Finally, the algorithm compares the indices of these coefficients and computes from that the ratio

of upward and downward steps as return value. While this approach already proved meaningful, its performance may improve considerably, if we include SplitData from the F0 Splitter (see below) in the computations.

- **NoiseExtractor**
  This Extractor is basically the same as the CondensedNoiseExtractor without the condensation step. So, it computes the average intensity contained in the weakest bands compared to the strongest bands. The result relates strongly to technical recording details and is so only of little use for similarity considerations. As the CondensedNoiseExtractor, the NoiseExtractor relies on ACQT data because of the less bands to consider compared to the STFFT, not because of its special properties regarding frequency resolutions.

- **SpectralFluxExtractor**
  This Extractor computes the average "flux", i.e., the change strength between consecutive frames. The flux is calculated as the average of the absolute differences between the N coefficients in frame i and $i+1$, $c_{i,0}, c_{i,1}, \ldots, c_{i,N-1}$ and $c_{i+1,0}, c_{i+1,1}, \ldots, c_{i+1,N-1}$:

$$F = \frac{1}{N} \sum_{n=0}^{N-1} |c_{i,n} - c_{i+1,n}|$$

  The average over all fluxes as the return value of this algorithm showed good correlation with the "dynamic" and "vitality" human listeners assign to pieces of music. This Extractor uses the CQT Splitter instead of the STFFT splitter to weight the different bands equally; high bands, with little audible differences, would otherwise dominate the result.

- **CentralSpectralFluxExtractor**
  This Extractor is basically identical to the SpectralFluxExtractor above. The difference is that this Extractor limits the flux computations to the bands within one octave of the strongest band in a CQT spectrum. Thereby the earlier frame i determines, which bands in the frames i and $i+1$ are chosen. Hence, the result correlates stronger with the perceived dynamic of the melody than the overall vivity.

- **SpectralRolloffExtractor**
  The SpectralRolloffExtractor works similar to the already discussed SpectralCentroidExtractor and delivers similar information: It also measures the spectral form of a signal to enable estimations about the perceived "richness" of a sound. The difference is that, instead of the center of gravity, the "spectral rolloff" is computed. This rolloff is usually defined (see, e.g., [7]) as frequency under which 85% of the energy contained in a spectrum is accumulated. To calculate this value, the SpectralRolloffExtractor operates analog to the SpectralCentroidExtractor. We use CQT data as basis for that instead of STFFT data to achieve faster computations (since less bands need to be taken into account) without significant different results.

4.2.4 Rhythm and Beat

The DFT and CQT (and their improvements) proved to be useful especially for low-level Extractors. For high-level Extractors, i.e., Extractors that try to incorporate knowledge from psychoacoustics and musicology (as discussed in 1.2.3), they are hardly directly applicable. Therefore, we implemented the "RhythmSplitter" to serve as basis for Extractors that gather knowledge about a song's rhythmic component. It is the task of this Splitter

to find and summarize meaningful events, as the onset of a note or a drum strike, in a piece of music. To achieve this, we developed an algorithm based on SplitData generated from the STFFT and RMS Splitters (each working with 2048 samples per window), with the following steps:

1. **Compute fluxes** In this first step, the differences between consecutive frames are computed and stored in an array. These differences, or "fluxes", are computed analog to the SpectralFluxExtractor. To make sure, that only pitch or timbre differences show up in the result, the spectra are first normalized to contain the same amount of energy. The fluxes will be used to decide, if the audible pitch in the observed bands has changed.

2. **Compute energy deltas** Now, we compute the energy difference of consecutive frames. This is done by adding up all coefficients in the analyzed band interval and comparing the sums. To obtain better robustness against local anomalies, we also include data from the RMS Splitter in the calculations. The result of this step is a vector of values, which indicate if the energy rose or dropped from one frame to the next.

3. **Find events** In this step, the algorithm identifies meaningful events in the analyzed music. An event is here regarded as meaningful, when it contains rhythmic information; so, we are looking for drum strikes or tone onsets, but not for a tone dying away. We thereby rely on three considerations: 1. When a note is played, the pitch changes suddenly; 2. Sounds generated by common instruments die away, so a drum strike or a tone onset most likely cause a sudden increase of sound energy; 3. Events are separated by a specific minimal time - otherwise, they would blur and become inaudible. Therefore, the algorithm scans the data for positions, where the flux exceeds a certain level (1.2 times the average flux seems a sensible threshold), the energy delta is clearly positive (above 1.3 times the average energy delta) and the last event is at least 0.15 seconds apart. See Figure 4.10 for an illustration of this step's results.

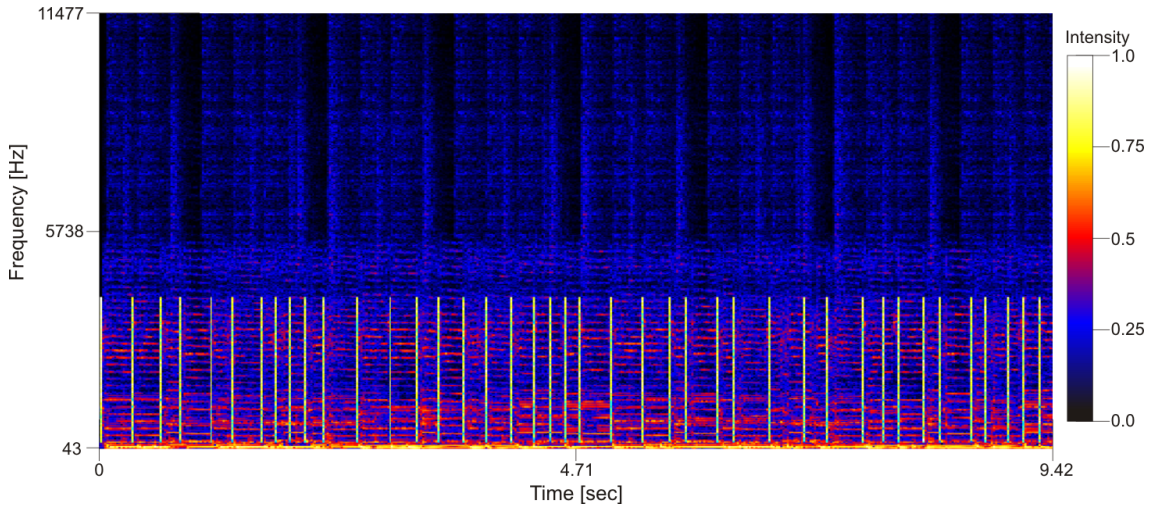


Figure 4.10: Illustration of the found events in the STFFT of the first ten seconds of *Hypocrisy Insane*'s "Tiny Monsters". The vertical, white bars indicate found events. The length of the bars reflects the analyzed frequency range from about 200 Hz to a little more than 4000 Hz.

4. **Build envelopes** Although the knowledge of these events is already quite useful, we can extract even more information if we distinguish between different kinds of events. Therefore, we assume that events of the same kind (as, e.g., the first beat in a bar) sound similar. So, the algorithm creates an "envelope" for each event. An envelope is a snippet of coefficients of the STFFT spectrum limited to a specific interval of bands. Thus, one may depict these envelopes as containers for the "timbre" of an event.

5. **Unify envelopes** As next step, the Splitter computes differences between the envelopes analog to the calculation of flux above. Envelopes with little distance between them are then considered to belong to the same event type. In this way, the envelopes are iteratively assigned to few event types (tests showed, that most kinds of music rarely contain more than ten event types).

6. **Generate SplitData** Finally, the SplitData to return is generated: The Splitter creates an array of values with each covering a window of 2048 samples to correspond to a frame of the STFFT or RMS SplitData. Then, the algorithm assigns an individual number to each event type and enlists every occurrence of an event type with this number in the array. See Figure 4.11 for an illustration.

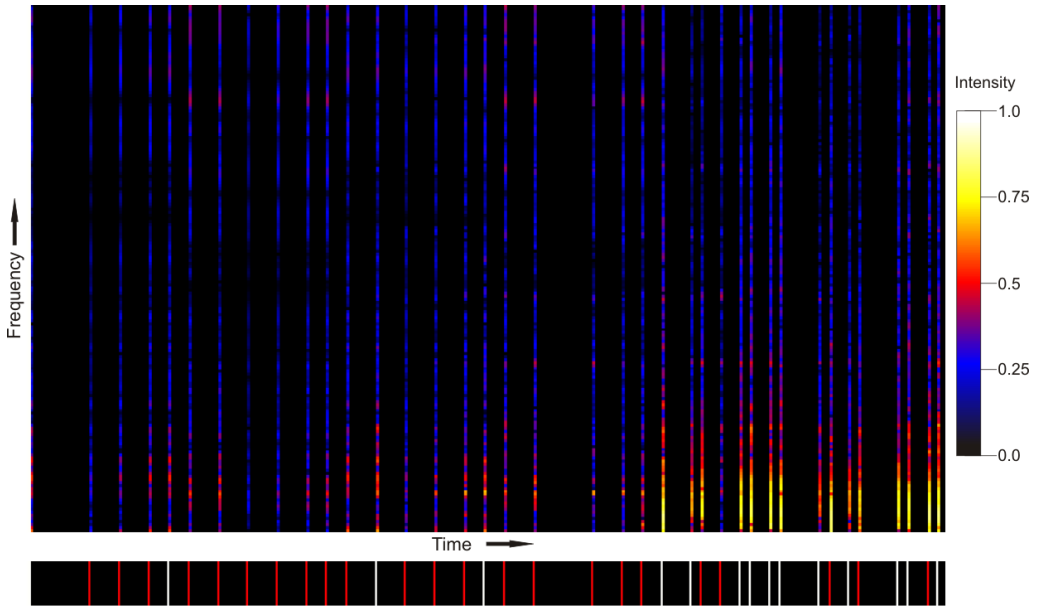


Figure 4.11: Illustration of further processing steps of the RhythmSplitter. The upper figure shows the envelopes extracted from the first 10 seconds of "Forgive me" by *:Wumpscut:*. These envelopes were combined to two event types; the lower frame shows their occurrences as stripes. Red stripes indicate events of type 1, white stripes events of type 2.

There are no Extractors yet utilizing the data from the RhythmSplitter. However, we plan to build up a template database of a few specific, widespread rhythms to gather knowledge about a song's rhythmic affiliation. Different Extractors would then calculate the likelihood, that a song incorporates a given rhythmic pattern. Furthermore, there are other imaginable Extractors utilizing this rhythmic data, as, for instance, Extractors calculating a songs tatum or beat or an Extractor to compute the variance of pauses between events to estimate how rhythmically consistent a piece of music is.

4.2.5 Fundamental frequencies

Another important source of information about how a song sounds are the notes played by melodic instruments[43]. These notes can be found as fundamental frequencies, or f_0, in digital music (see chapter 1.2.2, page 8). The identification of fundamental frequencies[44] is one of the oldest problems in the field of music information retrieval - and is still considered unsolved. Our approach, implemented in the F0 Splitter, is inspired by the PreFEst (Predominant F0 estimation model) system as presented by Goto in [15] and incorporates the same assumptions:

- Notes are played by instruments with harmonic structure in the analyzed music.
- There are two distinct melodic lines, the bass line and the melody line.
- The melody line has the most predominant harmonic structure in middle frequency regions, the bass line in low regions.
- Tones contained in these lines are audible for at least a specific period of time.
- Bass and melody line have temporally continuous trajectories.

These assumptions might look quite restrictive, but actually most Western music shows properties that match these assumptions.

The algorithm utilizes SplitData from the ACQT Splitter as well as from the RhythmSplitter instead of the "instantaneous frequencies" in [15], but its calculations resemble roughly the method of PreFEst. The separate processing steps are as follows:

1. **Create harmonic tone templates** To find F0 candidates, the F0Splitter needs to compare tones contained in the musical data with predefined tones. These template tones have idealized overtone spectra to match as many real instruments, which have their distinctive overtone spectra, as possible (Figure 1.7 on page 13 shows two tones of this kind). The Splitter generates these templates by adding up sinusoid functions with specific, exponentially decreasing amplitudes to form a short PCM signal. Finally, the algorithm analyzes this signal using the CQT technique and stores the obtained templates for later use.

2. **Estimate the tatum** Virtually all pieces of music show one shortest duration of tones, the "tatum". To estimate this tatum, the Splitter calculates all temporal distances between consecutive events first. Since the tatum usually labels a duration shorter than the beat of a song, we can safely ignore distances greater than one second in later computations. Afterward, distances that are multiples of a shorter occurring distance are added to these shorter tatum candidates. Now, the most often found distance is assumed to be the tatum of the song.

3. **Find fundamental frequencies candidates at specific points** With the tatum information, the Splitter can now start to look for fundamental frequency candidates: Experiments showed that harmonic spectra can be detected most clearly about the duration of a half tatum after an rhythmic event from the RhythmSplitter. At these

[43] We use the term "melodic instrument" for all kinds of instruments with a specific harmonic pitch - in contrast, e.g., to most percussion instruments. The focus lies thereby especially on typical melody-transporting solo instruments like the piano or the violin.

[44] This task is often referred to as "pitch tracking". As explained in section 1.2.2, this name is misleading: Although there is a strong correlation between fundamental frequency and perceived pitch, they are not identical.

moments, the noisy elements of percussion instruments and onset sounds usually already died away, whereas harmonic overtones, which tend to become silent before the corresponding fundamental, are still detectable. So, the F0Splitter looks for interpretable overtone spectra at these points first. The algorithm compares the previously generated tone templates with the ACQT data and understands the template that matches the real signal best as pointing at a fundamental frequency candidate. How good a template fits is estimated by computing the average distance of the template from the music data. These calculations are not executed on the ACQT SplitData directly, but on an average spectrum built from three frames around the start point to reduce noisy influences. All F0 candidates, that is to say, the complete spectra above the identified fundamental frequencies, are stored for the next step:

4. **Calculate the instrument template** Now, the algorithm creates a new overtone spectrum template out of the previously found candidates. This is done - after the removal of obvious outliers - by calculating the average spectrum. This spectrum should now contain the typical partials of the predominant melody or bass instrument.

5. **Apply template to whole song** In the final step, the algorithm utilizes the instrument template to find fundamental frequencies in the whole piece of music by matching the template at every sensible band with every frame of the ACQT data.

Albeit the F0Splitter showed promising results in some –rather simple – cases (see Figure 4.12), this algorithm is still to be considered experimental. The current implementation proved to be little robust against distorted instruments or highly polyphonic music. Also, pieces of music with predominant chords, as, e.g., of a rhythmic guitar, showed to cause erroneous results. Because of this preliminary state of the algorithm, there are no Extractors based on F0-SplitData implemented yet. However, the knowledge obtainable from this Splitter would be useful for a multitude of previously presented (e.g., the MelodyTrendExtractor) and potential Extractors, as, e.g., algorithms estimating the tense, monotony or repetitive character of a piece of music. Information about fundamental frequencies would also enable further Splitters, which could filter out the main theme or the used instruments.

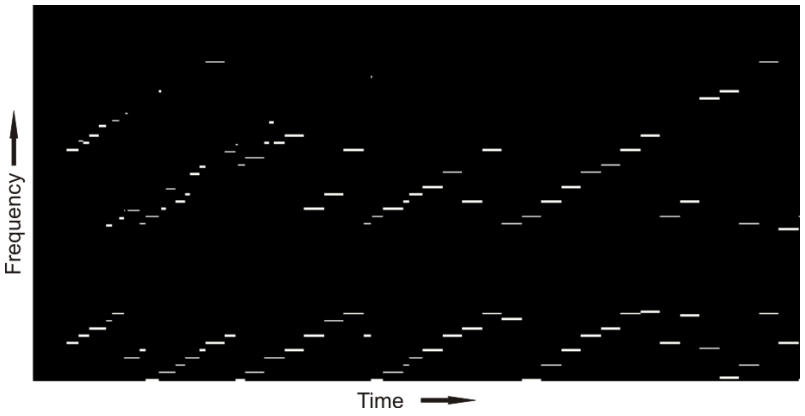


Figure 4.12: Illustration of found fundamental frequencies. Depicted is a short cutout of a simple etude played on a piano in “piano roll notation”: White bars indicate identified tones.

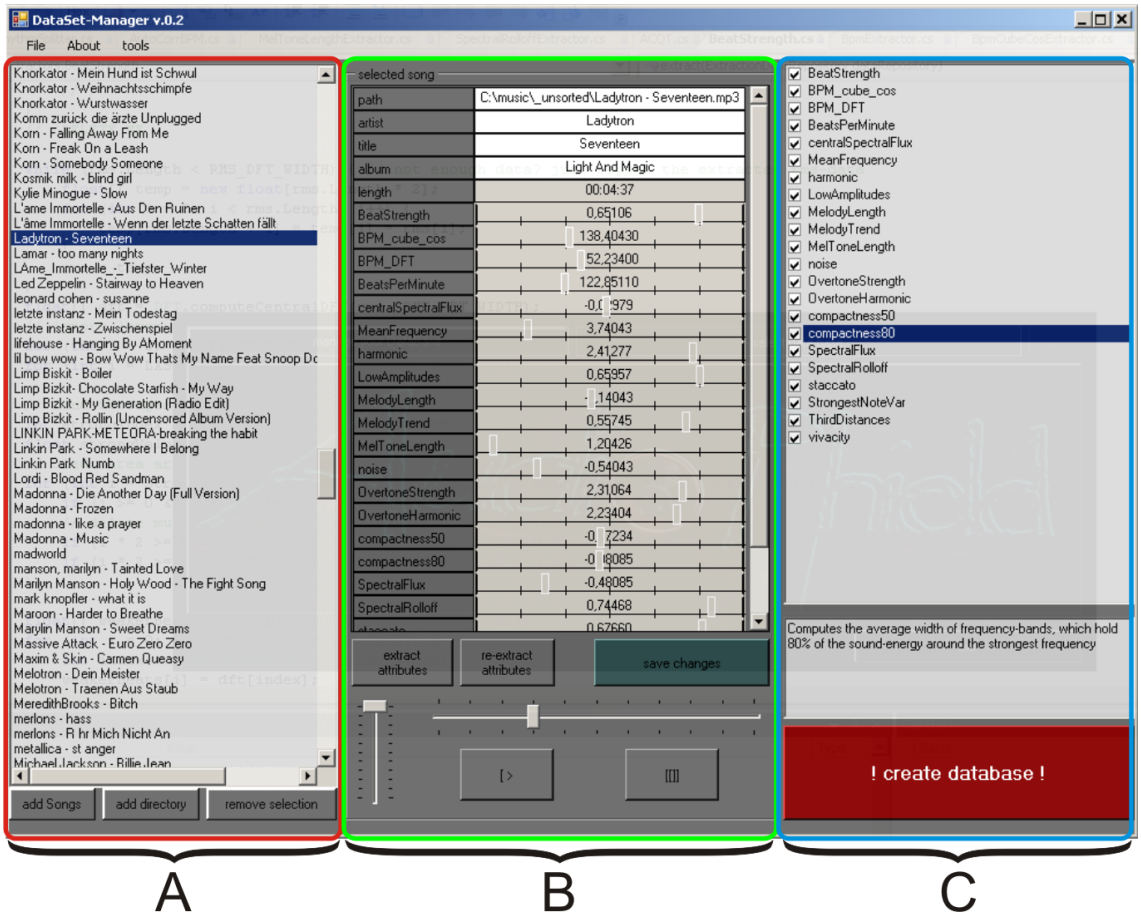


Figure 4.13: AudioPhield's main window. The colored frames highlight the three regions A, B and C, which are devoted to different tasks.

4.3 The user interface

One of our goals was to make AudioPhield easily accessible for users and developers. Therefore, we built a graphical user interface (GUI). This section will shortly outline this GUI.

The main form can roughly be divided into three separate areas (see Figure 4.13): The left area (A) is dedicated to manage, which pieces of music should be included in the database – and the calculations. It is possible to add single songs through the "add songs"-button as well as whole directories with all subdirectories ("add directory"). The middle section (B) details the currently in (A) selected song. Below generic fields showing path, artist, title and album of a song, all yet extracted values are listed and can be changed manually by adjusting the slide control next to the name of an Extractor. If changes were made, the user needs to press the "save changes" button – otherwise, all adjustments will be lost as soon as she selects another song in (A). (B) contains also buttons that start the analysis of a single piece of music in two different ways: While "extract attributes" preserves previously extracted (or manually set) values, "re-extract attributes" discards all of these. Finally, the lower part of (B) contains a simple (but fully functional) music player. The panel on the right, (C), contains a list of all currently available Extractors. In here, the user can switch the calculation of an attribute on or off; a text box below this list, which shows a description of the Extractor selected at the moment, assists in the decision. Finally, (C) contains the "create database" button, which starts the analysis of the complete database.

A second, very important window is the "SpectrumVisualizer" (see Figure 4.14). This tool was developed to visualize the results of all Splitters and Extractors. Its interface consists basically of two data frames. The upper panel depicts a sequence of frames/spectra as, for example, is created by the STFFT Splitter. The colors range from black over blue and red to yellow and indicate the intensity of bands (rows) at specific frames (columns). Below the panel are two zoom buttons, which change the amount of painted frames to improve the clearness of the illustration. The SpectrumVisualizer is also able to display, which data was used by an Extractor. These bands – and the currently selected frame – are painted inversely. To see just the underlying data, the user can also switch off this feature by unchecking the "show used data"-box. Right of the zoom-buttons is a text field that always describes the current position of the mouse pointer in terms of the represented data (e.g. in "seconds" and "Hertz"). The lower panel displays a detail view of the currently selected frame: The spectrum is represented as a sequence of columns, each associated with one band. Bands used by an Extractor are painted red instead of the normal black. At the lower edge of the form there are three controls that handle the playback of the depicted data starting from the currently selected frame – at least if the Splitter that generated the data supports the re-synthesis to PCM signals (at the moment, this is only the STFFT Splitter).

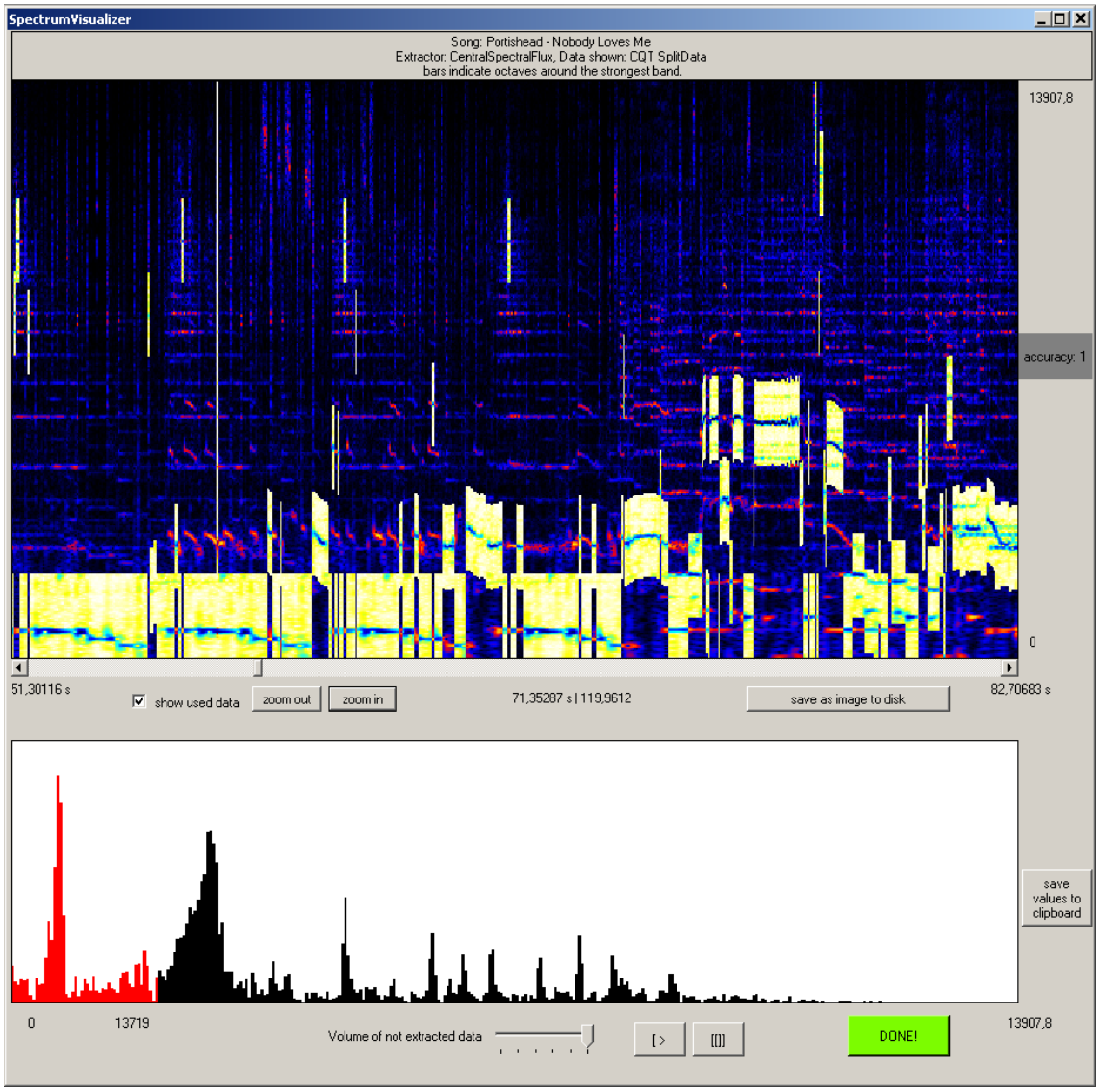


Figure 4.14: The SpectrumVisualizer window. Here, the result of the CentralSpectralFlux Extractor is shown.

5 Discussion and Future work

This project thesis has presented a variety of theoretical concepts regarding anatomical properties of the human sense of hearing, psychoacoustics and musicology. These concepts were taken to form the basis for a framework to analyze perceived similarities between different pieces of music. Also, some algorithms targeted at extracting useful information out of digitalized music were described. In this final chapter, we will discuss our achievements and compare them to our initially goals as stated in chapter 3. At last, we will close this project thesis with a short description of how we plan to continue our development of AudioPhield.

5.1 Discussion

Intensive recherches showed early, even before the implementation phase began, that there are no few major dimensions, according to them every piece of music could be aligned. Instead, there is a vast number of perceived musical dimensions, spreading from simple ones, as the presence of a particular instrument, to highly complex dimensions, as the perceived aggressiveness of a song. However, we found some algorithms that are capable of extracting data useful for similarity calculations out of digitalized music. As explained on page 27, the weighting of extracted values and combination of them to one value indicating the perceived similarity of two songs was postponed to a later implementation stage of "AudioPhield". We implemented a flexible framework, which serves as basis and communication platform for various algorithms. It is easy to expand; developers can add new algorithms by just implementing against simple interfaces – without any knowledge about framework-internal mechanisms. Also, a graphical user interface grants intuitive access to the framework's functions. Thus, in summary, this project thesis shows great progress toward the goal of similarity analysis – albeit some of our aims from section 3 proved unobtainable yet.

While the framework showed to be working very well, so far implemented Splitters and Extractors were not yet fully evaluated and may not tap their full potential: Most algorithms have a few "adjusting bolts"[45], arbitrary set constants influencing the calculation process, that can alter the performance of an algorithm greatly. If one changes, for example, the threshold values for flux and energy utilized by the RhythmSplitter, the Splitter can be adjusted to recognize very subtle changes in the audio signal. This adjustment, however, would greatly increase the risk of erroneous findings. Thus, optimal values as trade-offs between sensitivity and error-proneness have to be found.

Still, there are no perfect values for these "adjusting bolts" to be expected, because pieces of music differ just too strongly. To at least reduce the gravity of this problem, we included "accuracy"-fields in every storage class. Any algorithm that produces some kind of data has to estimate the quality of the produced result. The F0 Splitter, for example, works the better the undistorted and harmonic the playing instruments of a song are. In contrast, music primarily based on percussion sounds (which is typical, e.g., for techno music) or pieces of music containing highly distorted instruments, e.g. punk songs, will most likely cause the Splitter to produce flawed results. So, the Splitter stores - besides the actual F0 data – a 1.0 in the "accuracy" field of the returned SplitData if the audio signal was analyzed perfectly, or 0.0 if the signal was too noisy to permit sensible results. These accuracy values will be of great value at a later stage of the development, namely, when all the extracted values are to be combined to one similarity value. Accuracy estimations,

[45] Values of this type are named following the same notation throughout the whole framework: They are written in capitals and, if the variable names contain more then one word, components of the name are connected by underlines. For example: "LOWEST_FREQUENCY".

however, are of little use if a song in itself shows strong fluctuations of character: Imagine, for example, Metallica's "One"; it starts as a quiet ballad and grows in aggressiveness until it becomes a true metal song. One approach for this problem could be segmentation, as described in the next section.

5.2 Future Work

The discussion above should have made it clear that this project thesis documents just an intermediate step in the development process of AudioPhield. In this section, we will outline our mid- and long-term plans of implementation and investigation.

Our first and most obvious task is to evaluate and improve the implemented algorithms. This includes finding optimal replacements for the preliminary chosen "adjusting bolt"-values (see previous section) as well as improvements of the algorithms itself. We will focus thereby on increasing the quality of the produced results; however, we will also implement speedups where possible, since the current processing time on common hardware exceeds the playback length of a song two to three times – a time consumption that makes evaluations based on complete pieces of music almost inapplicable. Furthermore, new Extractor algorithms utilizing the data made available by the RhythmSplitter or F0Splitter need to be added to the framework.

One planed feature, a segmentation system, promises to speed up calculations as well as to improve the result quality. This system is supposed to identify different parts of a song (as, e.g., "refrain" or "interlude") and estimate its perceived memorability. Thus, Extractors can concentrate on the parts of a piece of music that are most likely to be associated with a song and represent the whole song best. This requires two steps: Segmentation of a track and estimating a segment's importance. There are already interesting approaches for both problems (see for example [46] or [2]).

With these improvements, we hope to build up AudioPhield to a system that is capable of robustly estimating perceived similarities between pieces of music. These similarity values will then serve as basis for new visualization and interaction techniques for medium to large music collections.

Inhalt der beigelegten CD

```
\source
    - Complete source code and necessary resources of AudioPhield
      (state: 2007-05-9)



\libs
    - Necessary libraries: DirectX 9 SDK (April 06), BASS, BASS.net (for .Net 1.1)
    - Further, optional BASS-plug-ins



\resources
    - Freely available documents referenced in the thesis paper in .pdf-format.
      Named according to the corresponding bibTex-entries



\thesis
    thesis.pdf
      - This project thesis in .pdf-format.
    \source
      - All LaTeX and image files necessary to build this document



content.txt
    - contents of the CD
```

References

[1] Apple Computer Inc. iTunes Music player. `http://www.apple.com/itunes`.

[2] William Birmingham. MUSART: Music retrieval via aural queries. In *International Symposium on Music Information Retrieval (ISMIR)*, 2001.

[3] Albert Bregman. *Auditory Scene Analysis*. MIT Press, 1990.

[4] Judith Brown. Calculation of a constant Q spectral transform. *The Journal of the Acoustical Society of America*, 89:425–434, 1991.

[5] Edward Burns and W. Dixon Ward. Categorical perception - Phenomenon or epiphenomenon: Evidence from experiments in the perception of melodic musical intervals. *Journal of the Acoustic Society of America 63*, pages 456–468, 1978.

[6] Arshia Cont. Realtime Multiple Pitch Observation using Sparse Non-negative Constraints. In *Proceedings of International Conference on Music Information Retrieval*, 2006.

[7] Perry Cook and George Tzanetakis. Musical genre classification of audio signals. *IEEE Transaction on Speech and Audio Processing*, 10(5), 2002.

[8] Markus Cremer and Claas Derboven. A system for harmonic analysis of polyphonic music. *AES 25th International Conference, London, UK*, pages 115–120, 2004.

[9] Stephen Downie. Music information retrieval. *Annual Review of Information Science and Technology 37*, pages 295–340, 2003.

[10] Stephen Downie, Joe Futrelle, and David Tcheng. The International Music Information Retrieval Systems Evaluation Laboratory: Governance, Access and Security. In *Proceedings of the 5th International Conference on Music Information Retrieval: ISMIR 2004. Barcelona, Spain.*, October 2004.

[11] Asif Ghias, Jonathan Logan, David Chamberlin, and Brian Smith. Query by Humming: Musical information retrieval in an audio database. In *Proceedings of the Third ACM International Conference on Multimedia ACM International Multimedia Conference*, pages 231–236, 1995.

[12] Emilia Gómez. Tonal description of polyphonic audio for music content processing. *INFORMS Journal on Computing, Special Cluster on Computation in Music, 18(3)*, 2006.

[13] Emilia Gómez, Anssi Klapuri, and Benoît Meudic. Melody Description and Extraction in the Context of Music Content Processing. *Journal of New Music Research*, 32(1), 2003.

[14] Leandro Gomes, Pedro Cano, Emilia Gómez, Madeleine Bonnet, and Eloi Batlle. Audio watermarking and fingerprinting: For which applications? *Journal of New Music Research*, 32(1):65–81, 2003.

[15] Masataka Goto. A Robust Predominant-F0 Estimation Method for Real-time Detection of Melody and Bass Lines in CD Recordings. In *Proceedings of the 2000 IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP 2000)*, pages 757–760, 2000.

[16] Masataka Goto and Yoichi Muraoka. A real-time beat tracking system for audio signals. In *Proceedings of the 1995 ICMC*, pages 171–174, 1995.

[17] Otmar Hilliges, Phillipp Holzer, Rene Klüber, and Andreas Butz. AudioRadar: A metaphorical visualization for the navigation of large music collections. In *Proceedings of the International Symposium on Smart Graphics 2006, Vancouver Canada*, July 2006.

[18] David Huron. Humdrum: Music tools for UNIX systems. *Computing in Musicology 7*, pages 66–67, 1991.

[19] Özgür Izmirli. Template based key finding from audio. In *International Computer Music Conference, Barcelona, Spain*, pages 211–214, 2005.

[20] Mari Jones and Marilyn Boltz. Dynamic attending and responses to time. *Psychological Review 96(3)*, pages 459–491, 1989.

[21] Michael Kassler. Toward musical information retrieval. *Perspectives of New Music*, pages 59–67, 1966.

[22] Anssi Klapuri. Automatic Transcription of Music. Master's thesis, Tampere University of Technology, 2001.

[23] Anssi Klapuri. Multiple Fundamental Frequency Estimation by Summing Harmonic Amplitudes. In *Proceedings of International Conference on Music Information Retrieval*, 2006.

[24] Anssi Klapuri, Antti Eronen, and Jaakko Astola. Analysis of the meter of acoustic musical signals. In *IEEE Trans. Audio, Speech, and Lang. Proc. 14*, pages 342–355, 2006.

[25] Rainer Klinke. Basic mechanisms of auditory processing in the cochlea. *Hearing Mechanisms and Speech, O. Creutzfeldt, H. Scheich, Chr. Schreiner (eds). Exp. Brain Res. Suppl. II*, pages 3–18, 1979.

[26] Stefan Kostka and Dorothy Payne. *Tonal Harmony.* McGraw-Hill Publishing Co., 2003.

[27] Martina Kremer. ars auditus - Das Mittelohr. `http://www.dasp.uni-wuppertal.de/ars_auditus/physiologie/mittelohr.htm`, 12 April 2007.

[28] Martina Kremer. ars auditus - Psychoakustik. `http://www.dasp.uni-wuppertal.de/ars_auditus/psychoak/psychoak0.htm`, 2 March 2007.

[29] Frank Kurth, Thorsten Gehrmann, and Meinard Müller. The Cyclic Beat Spectrum: Tempo-Related Audio Features for Time-Scale Invariant Audio Identification. In *Proceedings of International Conference on Music Information Retrieval*, 2006.

[30] Edward Large and John Kolen. Resonance and the perception of musical meter. *Connection Science 6*, pages 177–208, 1994.

[31] Arie Livshin and Xavier Rodet. The Significance of the Non-Harmonic "Noise" Versus the Harmonic Series for Musical Instrument Recognition. In *Proceedings of International Conference on Music Information Retrieval*, 2006.

[32] Robert Mannell. *The Perceptual and Auditory Implications of Parametric Scaling in Synthetic Speech.* PhD thesis, Macquarie University, 1994.

[33] Daniel McEnnis, Cory McKay, and Ichiro Fujinaga. jAudio: Additions and improvements. In *Proceedings of the International Conference on Music Information Retrieval*, pages 385–386, 2006.

[34] Cory McKay. jAudio: Towards a standardized extensible audio music feature extraction system. Course Paper. McGill University, Canada, 2005.

[35] Fraunhofer Institut Digitale Medientechnik. musicline.de - Die ganze Musik im Internet. `http://www.musicline.de/de/melodiesuche/input`, 4 May 2007.

[36] mf@ircam.fr. ISMIR - The International Conferences on Music Information Retrieval and Related Activities. `http://www.ismir.net/`.

[37] Microsoft. Msdn home page. `http://msdn.microsoft.com`.

[38] Richard Middleton. *Studying Popular Music*. Philadelphia: Open University Press, 1990.

[39] MIREX. MIREX 2007 - Main Page. `http://www.music-ir.org/mirexwiki/index.php/Main_Page`.

[40] Ken'ichi Miyazaki. How well do we understand absolute pitch? *Acoustical Science and Technology 25*, pages 270–282, June 2004.

[41] David Moulton. The Audio Window. `http://www.moultonlabs.com/more/audio_window/`, 4 May 2007.

[42] David Moulton, Alex Case, and Peter Alhadeff. About the Loudness of Sounds and the Risk of Hearing Damage. `http://www.moultonlabs.com/more/about_loudness/P1/`, 4 May 2007.

[43] Robert Neumayer, Michael Dittenbach, and Andreas Rauber. PlaySOM and PocketSOMPlayer, Alternative Interfaces To Large Music Collections. In *Proceedings of ISMIR 2005, London UK*, 2005.

[44] A. Michael Noll. Cepstrum pitch determination. *Journal of the Acoustic Society of America 41*, pages 293–309, 1967.

[45] Nullsoft. Winamp homepage. `http://www.winamp.com`.

[46] Bee Suan Ong. *Structural Analysis and Segmentation of Music Signals*. PhD thesis, University Pompeu Fabra, Barcelona, Spain, February 2007.

[47] Alan Oppenheim, Ronald Schafer, and John Buck. *Discrete-Time Signal Processing*. Prentice Hall, 1998.

[48] Geoffroy Peeters. Chroma-based estimation of musical key from audio-signal analysis. In *Proceedings of International Conference on Music Information Retrieval*, 2006.

[49] Dirk-Jan Povel and Peter Essens. Perception of temporal patterns. *Music Perception 2*, pages 411–480, 1985.

[50] R. Prather and R. Elliot. SML: A structured musical language. *Computers and the Humanities 24*, pages 137–151, 1988.

[51] Music Genome Project. Pandora Internet Radio. `http://www.pandora.com`.

[52] Christopher Raphael. Orchestral Musical Accompaniment from Synthesized Audio, 2003.

[53] Matti Ryynänen and Anssi Klapuri. Transcription of the Singing Melody in Polyphonic Music. In *Proceedings of International Conference on Music Information Retrieval*, 2006.

[54] Hajime Sano and Keith Jenkins. A neural network model for pitch perception. *Computer Music Journal 13*, pages 41–48, 1989.

[55] Eric Scheirer. Tempo and beat analysis of acoustic musical signals. *Journal of the Acoustic Society of America 103*, pages 588–601, 1998.

[56] Eric Scheirer. Music-listening systems, 2000.

[57] Catherine Schmidt-Jones. *Introduction to Music Theory.* Lulu Press, Inc., 2005.

[58] Peter Schneider, Vanessa Sluming, Neil Roberts, Stefan Bleeck, and André Rupp. Structural, functional and perceptual differences in the auditory cortex of musicians and non-musicians predict musical instrument preference. *Ann. NY Acad Sci. 1060*, pages 387–394, 2005.

[59] Peter Schneider, Vanessa Sluming, Neil Roberts, Michael Scherg, Rainer Goebel, Hans Specht, Günter Dosch, Stefan Bleeck, Christoph Stippich, and André Rupp. Structural and functional asymmetry of lateral Heschl's gyrus reflects pitch perception preference. *Nature Neuroscience 8*, pages 1241–1247, 2005.

[60] Jarno Seppänen, Antti Eronen, and Jarmo Hiipakka. Joint Beat and Tatum Tracking from Music Signals. In *Proceedings of International Conference on Music Information Retrieval*, 2006.

[61] Prarthana Shrestha and Ton Kalker. Audio Fingerprinting In Peer-to-peer Networks. In *Proceedings of the 5th International Conference on Music Information Retrieval, Barcelona Spain*, 2004.

[62] Roland Stigge. Programm zur Extraktion musikalischer Informationen aus akustischen Signalen. Diploma Thesis., 2004.

[63] Joel Sutton. MIRA: A PROLOG-based system for musical information retrieval and analysis. Master's thesis, University of North Carolina, Chapel Hill, 1988.

[64] Ernst Terhardt. Pitch perception. `http://www.mmk.ei.tum.de/persons/ter/top/pitch.html`, 4 May 2007.

[65] Ernst Terhardt. Frequency analysis and periodicity detection in the sensations of roughness and periodicity pitch. *Frequency Analysis and Periodicity Detection in Hearing (Plomp, R., Smoorenburg, G.F., eds.)*, pages 278–287, 1970.

[66] Ernst Terhardt. Calculating virtual pitch. *Hearing Research 1*, pages 155–182, 1979.

[67] The Board of Trustees of the University of Illinois. NCSA - Automated Learning Group. `http://alg.ncsa.uiuc.edu/`.

[68] Tero Tolonen and Matti Karjalaine. A computationally efficient multipitch analysis model. *IEEE Trans. Speech and Audio Processing, Vol. 8, No. 6*, pages 708–716, 2000.

[69] George Tzanetakis and Perry Cook. MARSYAS: A Framework for Audio Analysis. *Organized Sound, Cambridge University Press 4(3)*, 2000.

[70] un4seen developments. Un4seen developments - 2midi / bass / mid2xm / mo3 / xm-exe / xmplay. `http://www.un4seen.com/`.

[71] Hermann von Helmholtz. *On the Sensations of Tone*. New York: Dover Publications (Trans. A. J.Ellis), 1885.

[72] Nick Whiteley, A. Taylan Cemgil, and Simon Godsill. Bayesian modelling of temporal structure in musical audio. In *Proceedings of International Conference on Music Information Retrieval*, 2006.

[73] Gerhard Widmer and Patrick Zanon. Automatic Recognition of Famous Artists by Machine, 2004.

[74] Wikipedia, Die freie Enzyklopädie. Phon (Akustik). `http://de.wikipedia.org/wiki/Phon_(Akustik)`, 4 May 2007.

[75] Wikipedia, The Free Encyclopedia. A-weighting. `http://en.wikipedia.org/wiki/A-weighting`, 4 May 2007.

[76] Wikipedia, The Free Encyclopedia. Auditory masking. `http://en.wikipedia.org/wiki/Auditory_masking`, 3 May 2007.

[77] Wikipedia, The Free Encyclopedia. Ear. `http://en.wikipedia.org/wiki/Ear`, 4 May 2007.

[78] Wikipedia, The Free Encyclopedia. Fletcher-Munson curves. `http://en.wikipedia.org/wiki/Fletcher-Munson_curves`, 4 May 2007.

[79] Wikipedia, The Free Encyclopedia. Mel scale. `http://en.wikipedia.org/wiki/Mel_scale`, 4 May 2007.

[80] Wikipedia, The Free Encyclopedia. Overtone. `http://en.wikipedia.org/wiki/Overtone`, 4 May 2007.

[81] Ian Witten and Eibe Frank. *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, 2005.

[82] William Yost. The dominance region and ripple noise pitch: A test of the peripheral weighting model. *Journal of the Acoustic Society of America 72*, pages 416–425, 1982.