# NoNoise - Music Library Visualization

Manuel Keglevic*
Vienna University of Technology

Thomas Schulz†
Vienna University of Technology

November 6, 2011

## Abstract

In order to navigate through music libraries the common practice is to browse through lists of meta data. However, for libraries with thousands of songs a visual view of the library is more practical for browsing. There are standalone applications for music library visualization which use content-based features of the songs. These applications usually lack usability and features compared to sophisticated music players. We combine the positive aspects of both areas by providing a library visualization plug-in for an existing successful music player (i.e. Banshee Media Player). Our plug-in enables the user to browse a two-dimensional point representation using different levels of abstraction to accomplish the goal of creating a playlist containing songs he or she wants to listen to.

## 1 Motivation

With the growing success of digital music stores like iTunes Store or Amazon MP3 and the popularity of portable music players, personal digital music libraries become larger and larger. The typical approaches for browsing these libraries (e.g. searching for artist/track/genre/etc. or browsing a text view showing artist name, track title and sometimes album title) are based on meta data. However, meta data is likely to be incorrect or inaccurate (e.g. genres). Thus these text views are useful if you have a properly tagged music library and know which meta data you are looking for, but they do not provide a good interface for browsing the library, not knowing what exactly you are looking for.

Considering all this we decided to create a visualization view of the music library which is based on content data rather than meta data. The goal for this visualization was to be able to interactively browse through the music library and discover tracks you are currently in the mood for, while removing large areas of tracks you do not want to listen to at the moment. Finally it should be possible to use the visualization to create a playlist which contains only tracks that you want to listen to.

During our research we found some standalone visualization applications which had an integrated music player [4, 6]. However we came to the conclusion that hardly anyone would install an additional application just to visualize their music library and that no one would use these standalone applications as their main music player, because of their lack of usability and features. So instead of implementing a standalone application with re-

*e-mail:manuel.keglevic@gmail.com
†e-mail:thomas.schulz@student.tuwien.ac.at

duced usability we decided to implement a plug-in for an existing player and focus on the visualization. This also has the advantage that an existing player already has a user group which can simply test our visualization by enabling our plug-in.

The next step was to choose a music player. The factors that most affected our decision were the popularity of the player, its usability and the possibilities for writing an extension. We finally decided in favor of Banshee Media Player [1] because it is the default player in Ubuntu since 11.04 and, in contrast to most other popular players, it uses mono/C# [5] for its extensions instead of a script language.

We chose to use clutter-sharp [2, 3] to implement the main view of our plug-in, because it offers high-level access to OpenGL. Both the visualization itself and the user interface are implemented using clutter-sharp.

## 2 Data Processing

Before we can start with the visualization, we have to transform the available data (i.e. the tracks) into data which can be represented in two dimensions. Since meta data is often incorrect and inaccurate (e.g. genres or different ways of writing the same artist name or track title) we focus on content-based data. Using web services to get homogeneous meta data would be an alternative solution. However, most web services have query limits per user which makes it impossible to use them for a music player plug-in.

To get content-based data for each track, we have to analyze the music library. This is done using a slightly modified version of the Banshee Mirage [8] plug-in. This plug-in uses *Mel-Frequency Cepstral Coefficients* (MFCCs) to compute the similarity between two tracks. Since we want to get absolute data for each track and not just relative similarity to one seed track, we cannot use the similarity value computed by Mirage. Instead we use information from the MFCC matrix.

As the MFCC matrix is rather big ($1291 \times 20$) we only use certain vectors instead of the entire matrix. Depending on the user settings we either use its mean, squared mean, median, minimum or maximum vector.

This yields a 20-dimensional vector for each track in the music library, so we still have to reduce the dimensionality some more. For this task we use *Principal Component Analysis* (PCA). From the PCA we get two vectors which point in the direction of the larges variance. We then use these vectors as basis vectors to compute two-dimensional coordinates for each track.

Since the analysis of the music library takes about 2 to 2.5 seconds per track (6 to 7 hours for 10000 tracks) we have to store the data from the analysis in a database. For this we use an sqlite database with three tables. One table is used to store the five different vectors of the MFCC matrix, another table stores the two-dimensional coordinates from the PCA and the third table is used as an interface to the Banshee database and thus stores its track ids.

## 3 GUI

There are three main areas of interaction with our visualization:

- **Navigation**: the visible viewport can be moved using the mouse (dragging). Additionally, two buttons can be used to zoom in or out.

Filtering of the visible songs is done using the standard Banshee search area.

- **Selection and removal**: in selection mode, songs can be selected using free-form selection. Selected songs can either be cleared (i.e. unselected), removed or added to a playlist. Removed songs can be shown again using 'reset'.
- **Playback and playlist generation**: double clicking a song plays it (in case of a cluster all songs are added to a hidden playlist). Using 'playlist' all selected songs are added to a new playlist named 'NoNoise'.

Additionally, to update the PCA data stored in the database the menu 'Tools → NoNoise' can be used to start and pause scanning of the music library.

The user interface is shown in Figure 1. The zoom buttons are located in the upper left corner. At the top are all the other buttons. The info box in the upper right corner shows information about the circle under the cursor (i.e. songs in the cluster). When points are selected, a similar info box, which contains the information of the selected points, is shown in the bottom right corner. In the bottom left corner there is a status bar which shows important information.

# 4 Visualization

Our main goals for the visualization were simplicity and scalability. First of all, an intuitive user interface and visualization are important to quell the fears of users about trying something new. In general music players are designed for a broad audience of users who might be overwhelmed by complex interaction patterns. Secondly, music libraries with a size of 10000-20000 songs are not exceptional, therefore the visualization must stay informative and computationally efficient even for such a number of songs.

In the visualization every song is represented by a translucent white circle. The position of the circle is determined by the PCA coordinates stored in the database. However, to maintain scalability clustering is used to provide a level of detail depending on the size of the currently visible area. For simplicity we chose not to encode information about the cluster into the size or color of the circle.

To decrease computational effort the coordinates of the songs are stored in a quadtree. This also enables efficient viewport clipping for further performance optimization.

## 4.1 Clustering

When visualizing a large number of songs various abstraction levels are needed to allow a fine-grained transition from detailed information about the relationship between a handful of songs to the general structure of the whole music library. We chose to use multiple clustering levels to enable this transition. However, there were several requirements for the clustering algorithm.

Firstly, as previously noted, we chose not to encode information about the clusters (for example the number of songs) into their visible representations. This means, in order not to falsify the visualization at each abstraction level the number of songs in the clusters should be equal.

Secondly, performance is crucial. The clustering levels have to be recomputed every time the PCA coordinates change. In order to allow the user to

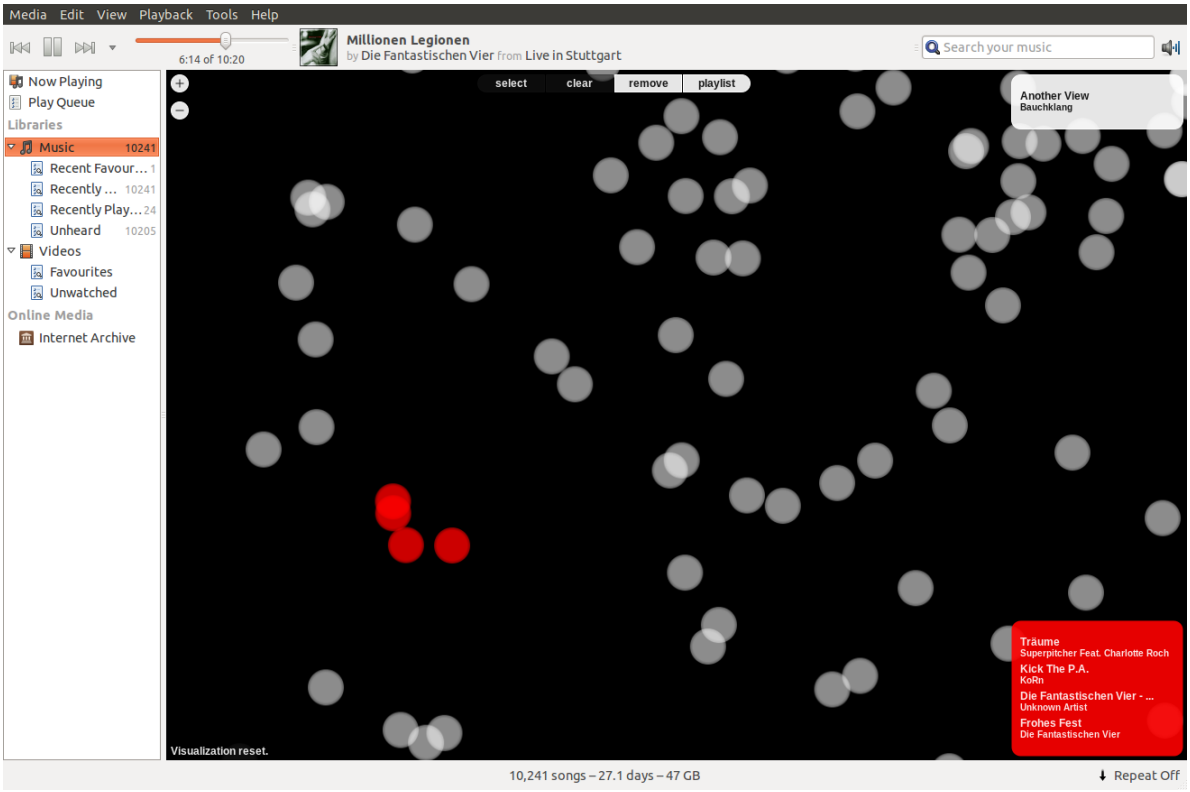


Figure 1: Banshee with NoNoise view.

change the PCA feature set, this cannot be precomputed.

Hence, after testing common clustering algorithms, we chose to use a modified hierarchical clustering. To guarantee that for each clustering level the number of songs in a cluster is always the same, the clusters of the previous level are clustered pairwise (using the euclidean distance) in each clustering step. To minimize the error introduced by this modification an optimal solution would be to minimize the accumulated distance between these pairs. However, this problem is NP hard and although this can be approximated using a heuristic the computational effort is still too high for big music libraries. Therefore, we always choose the closest pairs first for clustering. The main advantage of this method is that the songs are already stored in a quadtree which allows an efficient search for the nearest neighbor.

For simplicity, we think it is important that the visualization changes gradually and only as much as necessary. Hence, instead of recalculating the position of the clusters using the barycenters, we keep the position of one of the previous clusters. As a result, instead of removing all old circles and introducing new ones at different positions, half of the circles are faded out and the rest stays the same. The clustering of an area is shown in Figure 2.

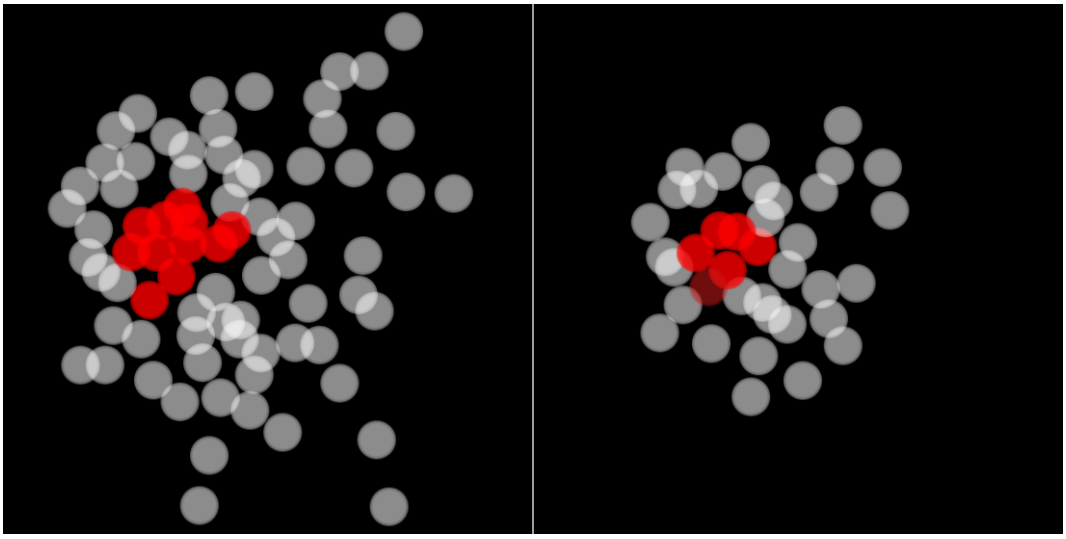

Figure 2: Comparison of two clustering/zoom levels.

### 4.2 Performance optimization

When testing our visualization with 5000-10000 songs we identified two major performance bottlenecks:

- Allocation of clutter actors (needed for rendering)
- Movement of the viewport

First of all, the allocation of 5000 clutter actors (one for each song) takes about 11s. Secondly, even though all these songs are never visible at once, because they are either clustered or outside the viewport, moving the viewport around is not smooth anymore.

To overcome these problems, we use a pool of 3000 actors. Every time a song is newly visible, an already allocated free actor is assigned to this song, moved to the right position and shown. Similarly, when a song is moved out of the viewport the associated actor is freed again. The test, if a song lies inside the viewport or outside is efficiently done using the quadtree.

This way, the rendering performance of the visualization is in $O(n \log n)$ with $n$ being the number of songs in the library.

## 5 Problems

The biggest problem we encountered writing this plug-in was the lack of documentation of both clutter-sharp and Banshee. Although clutter is well documented in C, it is not always obvious how to translate this documentation to clutter-sharp. Additionally, some clutter functions are simply not part of clutter-sharp or computationally expensive. Banshee itself is not documented at all (there are some rare inline comments), which means that even simple things like getting a song object using its id can take some painful days.

Another problem was the absence of music analysis libraries for C#/mono. There is a huge number of free-to-use libraries for C, but we did not find any C# library which was suitable for us. The only viable option for us was to use parts of Mirage. However, this yields one big cluster rather than several small ones. Even though neighboring songs sound similar, songs of the same artist or genre may spread across the domain.

## 6 Results and Future Work

A crucial point for our visualization is that similar songs are clustered together such that they are visually distinct from each other. As shown in Figure 3 on the left, songs of different genres (Country, Reggae and Hardstyle) are separated into different clusters (green, grey and red circles respectively) with the exception of two clusters (dark red) containing songs of two genres. However, adding a single Classical music compilation to the library reduces the x-axis space available for the other three genres by approximately 60%. Therefore, the separation of the genres is not as clear anymore. For example, as shown in Figure 3, Hardstyle and Reggae would not be visually distinct without the additional coloring.

In bigger libraries the songs are spread across the domain and form one big cluster rather than forming visually distinct clusters containing similar songs. In order to improve this, other features of the songs, which enable meaningful clustering, have to be used.

However, music information retrieval is a research area of its own and extracting these features requires a whole new project. Yet, after working on this project we are even more intrigued with music visualization and are looking forward to seeing further work in this area.

## References


[1] *Banshee*. URL: `http://banshee.fm/`.

[2] *Clutter*. URL: `http://www.clutter-project.org/`.

[3] *Clutter-sharp*. URL: `http://www.clutter-project.org/blog/2007/04/clutter-sharp-bindings`.

[4] Anita Lillie. *MusicBox*. URL: `http://thesis.flyingpudding.com/`.

[5] *Mono*. URL: `http://www.mono-project.com/`.

[6] Elias Pampalk. *Islands of Music*. URL: `http://www.ofai.at/~elias.pampalk/music/index.html`.

[7] Christoph Rüegg. *Math.NET*. URL: `http://www.mathdotnet.com/`.

[8] Dominik Schnitzer. *Mirage*. URL: `http://hop.at/mirage/`.

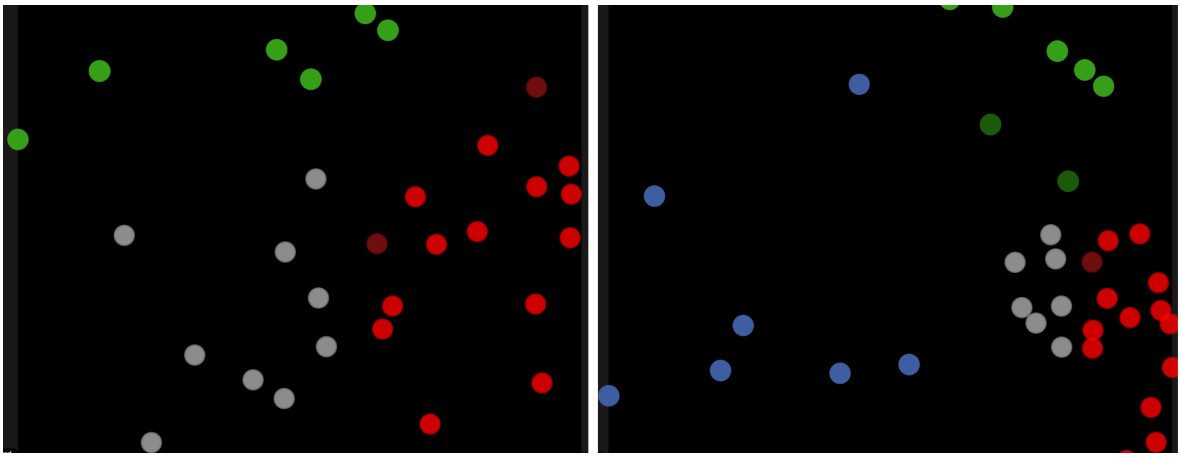

Figure 3: Comparison of our visualization with different libraries using the median vector without song duration. On the left, Country (green), Reggae (grey) and Hardstyle (red) and on the right additionally Classical music (blue). *Colors were added manually for distinction of the genres. Used albums: Kitty, Daisy & Lewis - Smoking In Heaven (Country), Bob Marley - Legend (Reggae), Dj Exorzist - Hardstyle Constructor 2011/2 (Hardstyle), The Top 100 Masterpieces Of Classical Music Vol. 8 (Classical music).*