MUME 2016 - The Fourth International Workshop on Musical Metacreation, ISBN #978-0-86491-397-5

# Creating an image and animation based body for associative music visualization in the browser

**Balazs Krich**
balazs.krich@gmail.com

**Abstract**

We present an HTML5 application which introduces novel representations of musical information in the context of the Web Audio API. The application is human-centered, i.e. it offers the possibility for the user to upload audio-files and by default aims to demonstrate the original artwork associated with the audio file in order to create a visual environment where musical information can be represented. The user also has the ability to refine this visual environment by providing additional metadata, uploading assorted images or animated GIFs. Finally, musical information is displayed real-time in this customizable context with the help of various feature extraction algorithms, including contour detection and beat tracking.

**Keywords:** Web Audio API, beat tracking, contour detection.

## 1. Introduction

Album artwork is usually much more than a simple illustration provided for popular music: attracting attention might be the cover's primal function, but it is also a work of art in its own right, often complementing, describing and even framing the music it accompanies. In the digital age providing a representation of the music that's easily used for organization and sharing (Graham, Hull 2008) became even more important as music is mostly available in non-physical formats, therefore offering visual clues can make all the difference as opposed to retrieving music only through textual information.

Although there is an increasingly high demand for systems which can process data in a "musically intelligent" way (Dixon 2001), and information visualization has become a promising alternative for representing musical metadata (Husain, Shiratuddin, Wong 2013), the illustration of musical data has mostly been limited to using abstract and geometrical forms and shapes – typically consisting of objects with various colors, positions and other attributes (Nanayakkara et al. 2007) – and it has not been a key consideration to include original artwork as a body for representation. Furthermore, music visualizations based on human computer interaction rarely offer the possibility to reuse imagery that is well established in the common domain – this way providing new layers of understanding for both the music, the referenced visuals and the created visualization experience as a self-contained unit.

Our application aims to exploit these formerly untouched concepts in music visualization by providing the user the freedom to choose any image or animated GIF as a base for visualizing music information. By default, the application will try to retrieve the original cover art belonging to the uploaded audio-file, but the user can override this by:

- Performing a free-text search for other cover art
- Uploading any other static image
- Uploading an animated GIF

After completing the editing of the visualization, predefined feature extraction algorithms perform a real-time visualization of the uploaded song on the selected imagery.

In the following sections we will briefly describe the algorithms that support the application, and provide examples of the working demo.

## 2. The application

The application takes a "systems" approach, building on prior work in beat tracking, image segmentation, online searching and the Web Audio API – a high-level JavaScript API designed for real-time processing of audio inside the browser through various processing nodes.

We decided to deploy the application in a Web environment mainly because accessibility was a key deciding factor and we also agree that the Web is ultimately becoming a complete software platform capable of results comparable to hardware-dependent systems (Rawlinson, Segal,

Fiala 2015). Unfortunately the Web Audio API currently is only supported by cutting-edge browsers such as Chrome and Safari (Choi, Berger 2013) and not all features are distributed evenly even among these browsers – the application offers full functionality only in Chrome as of speaking.

### 2.1. Collecting metadata

The user can upload any audio-file in .mp3-format – the number of songs uploaded at a time are not restricted within reasonable size-limitations (100MB / upload). If cover art, artist name and song title is included in the ID3 metadata encoded in the .mp3-file, the application will accept these as default – though in many cases data quality is insufficient for automatic processing. This is because meta information found in ID3 tags mainly come from databases like Gracenote or the FreeDB project. Those databases are generated by a large community of volunteers and their input – though very useful in many applications – is not quality assured (Baumann, Klüter, Norlien 2002). If metadata is only partially filled – for example the song title is available, but no cover art or artist name is included – the application will make subsequent requests to the Discogs RESTful API, which is one of the biggest database of audio recordings currently available online (Brooke 2013), and try to associate an album cover for the song, while performing rank-based text-transformations on the available metadata for the most probable match. If finding a match is unsuccessful, the user can still provide an artist name or a song title and make another request for the Discogs API to collect the album art, or just upload any image.

### 2.2. Generating the visualization background

When an image is successfully retrieved (either encoded in the .mp3 file, downloaded from the Discogs database or uploaded by the user), a 30 columns wide and 20 rows high hexagon grid is built with the help of the D3.js JavaScript library, allowing dynamic transforms to both generate and modify content within the standard document object model (DOM) (Bostock 2011). Simultaneously the associated cover art is sliced to the same partition (a 30 by 20 grid), and from each block an average color is extracted. Finally each hexagon receives the average color of the block in the same position within the grid, giving a pixelated, 3D-mimicked illusion of the original cover art in the browser.

Once the uploaded song is played, real-time frequency and time-domain analysis information is extracted from the audio file by setting up an AnalyserNode provided by the Web Audio API. By setting the Fast Fourier Transform (FFT) size to 512 – also natively provided by the Web Audio API – we are able to extract 256 frequency data values in real-time, and pass these values for SVG-transformations to every second hexagon object, starting from the 88$^{th}$ position in the 30 by 20 grid. As a result, the height of the selected hexagons change according to the passed frequency value, creating a full-screen frequency bar graph from the original artwork or associated image.

### 2.3. Beat tracking

Firmly agreeing to the statement that no computer program has been developed which approaches the beat tracking ability of a good musician (Dixon 2006) and considering that although there have been several comparative studies of beat tracking performance, there is no current consensus on which evaluation method to use (Davies, Degara, Plumbley 2009), introducing a novel solution regarding beat detection was no criterion for the development of the application, especially as we needed a feature functioning in real-time. Instead we settled for a good enough solution and evaluated our algorithm's results from a visual point of view.

First, we decided on a frequency level (beatMin) which would be a minimum for registering a beat – i.e. a volume less than this constant would not be considered a beat. Then we normalized the frequency data provided by the AnalyserNode and calculated an average level for all 256 frequency values (beatLevel) at each given frame. When the beatLevel was higher than the beatMin constant, we registered the first beat. With each subsequent frame we reduced the beatLevel value with a previously determined constant (beatDecayRate), defining another variable for each frame (beatCutOff). At any frame, if the beatLevel of the frame was higher than the beatMin and the beatCutOff value of the previous frame, we registered a new beat, and the beatCutOff variable became equal to the beatLevel value of that frame.

By introducing a beatHoldTime constant – which equals to the number of frames we decided to hold a beat – and registering the time passed between two detected beats, we found that this simple and quite naïve algorithm was sufficient enough to create visualizations that are in sync with the sound.

### 2.4. Animating still images to the beat

Once an image is successfully associated with a given audio file, the application performs contour detection and hierarchical image segmentation on the image with the help of the Python binding of Open Source Computer Vision Library (OpenCV). The result of this operation is saved in an SVG file format, which contains all detected objects in the image in a hierarchy, and is included in the DOM tree.

When the audio file is played and the user hovers the cursor on the associated image, the previously generated SVG objects are projected over the original image. If a

beat is detected, a custom algorithm performs transformations on a randomly selected number of SVG objects in the following manner: for each object a random value is specified from the array of the four cardinal and four intercardinal directions. The object is moved towards the selected direction by a distance determined by the beatLevel of the actual frame – the louder the volume, the further the object is moved –, then a shaking effect is applied to the object, as if a spring would try to retract it to its original position.

The result is a visualization performed on any freely associated image, where the objects detected on the image move real-time to the beat of the audio file played. A couple of examples of this behavior are available at:
http://www.balkeyplayer.com/surfsamurai
http://www.balkeyplayer.com/scotty
http://www.balkeyplayer.com/cale

### 2.5. Looping GIF's to the beat

The user also has the choice to associate an animated GIF to the selected audio-file by uploading such a file. If a .gif file containing more than one frame is detected – i.e. the GIF is animated –, the Python backend of the application will save each frame with the help of the ImageMagick software suite, and pass the path of the images and the original order of them to the browser, so they can be displayed programmatically with JavaScript on the client side.

If the user hovers the cursor on the uploaded animated GIF while the audio file is played, the sliced frames will move to the beat controlled by the following algorithm: the frame rate by which the images are looped is specified by the time passed since the last detected beat. The more frequent the beats are, the faster the images are looped – there is also a lower threshold applied, so the frame rate can not be lower than a constant specified in a variable. By applying an upper threshold value, we could also define a phenomenon we labeled as "break point" – this would be the point in dancing, when the person dancing breaks a harmonious movement and starts a new move, usually performed when the beat changes or becomes significantly faster. So if the time between two detected beats is less than the upper threshold value, the looping of the images randomly switches direction, i.e. making the animation responsive to the sequence of the beats played. After fine-tuning parameters through a number of tests we found that the current settings work out harmoniously with abstract animations and dancing figures or even short video recordings most of the time. A couple of examples are available at:
http://www.balkeyplayer.com/moonwalk
http://www.balkeyplayer.com/dynamite
http://www.balkeyplayer.com/hollie-cook
http://www.balkeyplayer.com/daftpunk
http://www.balkeyplayer.com/pulpfiction

## 3. Conclusion

We demonstrated a working demo capable of embedding musical information in a freely chosen illustration body: the user has the ability to display musical data in the original artwork, select a still image or animation, and create a highly associative audiovisual experience in the browser – this might be of interest to artists looking for new ways to combine sound and image or to music visualization researchers. Uploading audio files is available at:
http://www.balkeyplayer.com/

## References


Baumann, S., Klüter, A., and Norlien, M. 2002. Using natural language input and audio analysis for a human-oriented MIR system. In Proceedings of Web Delivering of Music, 74-81. Darmstadt, Germany: Second International Conference on Web Delivering of Music.

Bostock, M. 2011. D³ Data-Driven Documents. *IEEE Transactions on Visualization and Computer Graphics*, 17(12): 2301-2309.

Brooke, T. 2013. Descriptive metadata in the music industry: Why it is broken and how to fix it. *Journal of Digital Media Management* 2(3): 263-282.

Choi, H., and Berger, J. 2013. Waax: web audio API extension. In Proceedings of the international conference on new interfaces for musical expression, 499-502. Daejeon, Republic of Korea: International conference on new interfaces for musical expression.

Davies, M. E. P., Degara, N., and Plumbley, M. D. 2009. Evaluation methods for musical audio beat tracking algorithms, Technical Report, C4DM-TR-09-06, Centre for Digital Music, Queen Mary University of London.

Dixon, S. 2001. An interactive beat tracking and visualisation system. In Proceedings of the international computer music conference, 215-218. Havana, Cuba: International Computer Music Conference.

Dixon, S. Mirex 2006 audio beat tracking evaluation: Beatroot. *MIREX 2006*: 27-30.

Graham, J., and Hull, J. J. 2008. Icandy: a tangible user interface for itunes. In Proceedings of the Extended Abstracts on Human Factors in Computing Systems, 2343-2348. Florence, Italy: 26th CHI Conference.

Husain, A., Shiratuddin, M. F., and Wong, K. W. 2013. A proposed framework for visualising music mood using texture image. In Research and Innovation in Information Systems, 263-268. Kuala Lumpur, Malaysia: 2013 International Conference on Research and Innovation in Information Systems.

Nanayakkara, S., Taylor, E., Wyse, L., and Ong, S. H. 2007. Towards Building an Experimental Music Visualizer. In Proceedings of the sixth international conference on Information, Communication and Signal Processing, 1-5. Singapore: Sixth international conference on Information, Communications, and Signal Processing.

Rawlinson, H., Segal, N., and Fiala, J. 2015. Meyda: an audio feature extraction library for the web audio api. In Proceedings of the 2015 Web Audio Conference. Paris, France: 2015 Web Audio Conference.