About

This experiment-oriented blog attempts to bring forth music visualization in both individual and combinational forms of lines, shapes, and colors. German musician Kontra K’s latest album “Gute Nacht” is set as the experimental subject for potential visual performances. Animated representations along with short text explanations are expected to convey the musical messages to a potential audience in a compelling and clear manner, which also seek to serve as promising digital prototypes for concert lighting/stage design. Ideation and working process are detailed for others’ reference. This project is mainly credited to Kontra K, Zachary Lieberman (visual artist), and Ian Reah (software developer).

Bio

Yuqi Wang, born in 1994 in Zhejiang, China. He studied Economics/Management Science at Sun Yat-sen University. Currently majors in Communication Design at Parsons School of Design, focusing on Interaction Design. With enthusiasm for music performance and German culture, he is pursuing future study experience in Lighting/Stage Design in Germany. Lives in New York City, United States.

Post 1

Why I want to do

People take music for granted, focus on lyrics and vocal, but neglect instrumental, which actually takes the major responsibility for listeners’ visceral reaction.

Interaction design is about interact, same as music is. Visual presentation also can vivify music and thus convey clearer messages. Also, this enables

Post 1

Why I want to do music visualization & What I can do

Problem: Pop music – Center Vocal

Solution: Other senses (Visual/Touch) – Enhance auditory presentation

My approach as an interaction designer: Audio API Visualize Audio

Melody and lyrics are centered in pop music. Modern audiences are thus less appreciative of the instrumental, and apt to give major credits to the dominant musical part. Higher pitch of the melody and emphatic presentation of lyrics might account for this behavioral choice. Comparatively, classical music tends to offer listeners musical experience in a pure and simple form, enabling listeners to detect auditory nuances. Apparently, this approach is not the sole solution to this problem. Other responses are being made and assessed constantly. Sensory representation is widely adopted to addressing this issue. Visual and tactile reproductions can be experienced in concerts, musical festivals, and clubs. Lighting design as an intriguing implement displays its popularities and effectiveness in those environments. Interaction design as a digital approach has the potential to ameliorate the status quo in a fairly economical fashion. By mimicking the lighting effect, this project seeks to imbue viewers the notion of being aware of and appreciating instrumental music.

Thus, for this project, Gute Nacht as an appropriate musical subject is experimented. Musician Kontra K’s introduction (in German) from Genius is given below.

Post 2

Implement – Concept

The implement this project chiefly employs is termed Web Audio API. The following explanation from MDN details the concepts and usage of this tool.

The Web Audio API involves handling audio operations inside an audio context, and has been designed to allow modular routing. Basic audio operations are performed with audio nodes, which are linked together to form an audio routing graph. Several sources — with different types of channel layout — are supported even within a single context. This modular design provides the flexibility to create complex audio functions with dynamic effects.

Audio nodes are linked into chains and simple webs by their inputs and outputs. They typically start with one or more sources. Sources provide arrays of sound intensities (samples) at very small timeslices, often tens of thousands of them per second. These could be either computed mathematically (such as OscillatorNode), or they can be recordings from sound/video files (like AudioBufferSourceNode and MediaElementAudioSourceNode) and audio streams (MediaStreamAudioSourceNode). In fact, sound files are just recordings of sound intensities themselves, which come in from microphones or electric instruments, and get mixed down into a single, complicated wave.

Outputs of these nodes could be linked to inputs of others, which mix or modify these streams of sound samples into different streams. A common modification is multiplying the samples by a value to make them louder or quieter (as is the case with GainNode). Once the sound has been sufficiently processed for the intended effect, it can be linked to the input of a destination (AudioContext.destination), which sends the sound to the speakers or headphones. This last connection is only necessary if the user is supposed to hear the audio.

A simple, typical workflow for web audio would look something like this:

1. Create audio context
2. Inside the context, create sources — such as <audio>, oscillator, stream
3. Create effects nodes, such as reverb, biquad filter, panner, compressor
4. Choose final destination of audio, for example your system speakers
5. Connect the sources up to the effects, and the effects to the destination.

Timing is controlled with high precision and low latency, allowing developers to write code that responds accurately to events and is able to target specific samples, even at a high sample rate. So applications such as drum machines and sequencers are well within reach.

The Web Audio API also allows us to control how audio is spatialized. Using a system based on a source-listener model, it allows control of the panning model and deals with distance-induced attenuation or doppler shift induced by a moving source (or moving listener).

https://developer.mozilla.org/en-US/docs/Web/API/Web\_Audio\_API

Post 3

Implement – Interfaces

Interfaces essential for this project also need to be introduced before we get our hands dirty. The definitions from MDN are listed below.

AudioContext

The AudioContext interface represents an audio-processing graph built from audio modules linked together, each represented by an AudioNode. An audio context controls the creation of the nodes it contains and the execution of the audio processing, or decoding. You need to create an AudioContext before you do anything else, as everything happens inside a context.

AudioNode

The AudioNode interface represents an audio-processing module like an audio source (e.g. an HTML <audio> or <video> element), audio destination, intermediate processing module (e.g. a filter like BiquadFilterNode, or volume control like GainNode).

MediaElementAudioSourceNode

The MediaElementAudioSourceNode interface represents an audio source consisting of an HTML5 <audio> or <video> element. It is an AudioNode that acts as an audio source.

AnalyserNode

The AnalyserNode interface represents a node able to provide real-time frequency and time-domain analysis information, for the purposes of data analysis and visualization.

Ian Reah offers a possible approach to audio visualization with a clear workflow employing these interfaces in his blog “Real-time analysis of streaming audio data with Web Audio API,” whose contribution to this project is highly appreciated. Five major steps ought to be taken as interpreted:

1. Create AudioContext for the source-to-destination routing

var context = new AudioContext();

1. Get audio input by using Audio tag

<audio id=”id” src=”url”></audio>

var audio = document.getElementById(“id”);

var source = context.createMediaElementSource(audio);

1. Create Analyser and connect nodes

var analyser = context.createAnalyser();

source.connect(analyser);

Analyser.connect(context.destination);

1. Access the source data (frequency)

analyser.fftSize = 2048;

analyser.frequencyBinCount = 1024;

var frequencyData = new Uint8Array(analyser.frequencyBinCount);

analyser.getByteFrequencyData(frequencyData);

1. Define visual output by creating animations

http://ianreah.com/2013/02/28/Real-time-analysis-of-streaming-audio-data-with-Web-Audio-API.html

Post 4

Ideation

Data: Data.Original – Data.Instrumental = Data.Vocal

Data.Vocal w/ Data.Instrumental

Visualization:

Variables

Shape: Line, Circle, Square (Width, Height, Fill, Stroke, Border-Radius, Rotate)

Animation: Box-Shadow, Background-Color, Opacity, Top, Height

Post 5

Code – Get Data of Vocal and Instrumental

Due to the lack of original vocal tracks, songs with vocal and instrumental mixed and pure instrumental parts are both needed for the data analysis. Thus, two Audio tags should be created in HTML file for the mixed and the instrumental respectively. Sources would be dynamically defined when the “click” event is triggered on Music Tab, which is on the top right.

<audio id="mixed" src="#" preload="auto" crossOrigin="anonymous"></audio>

<audio id="instrumental" src="#" preload="auto" crossOrigin="anonymous"></audio>

Post 6

Code – Simultaneous Play & On Repeat

Audio Event (canplay, canplaythrough, ended)

Post 7

Code – Background Pattern

The HTMLElement.dataset property is used for accessing specific pattern settings when Blog Tab (on the bottom right) is clicked. Variables in use are of two groups: width, height, fill, stroke, border-radius, and rotate for defining the shape, and box-shadow, background-color, opacity, top as well as height for determining the animation. Two modes of animations for the same post are essentially in the same shape.

Post 8

Code – Visualization Based on the Chosen Pattern

Update();