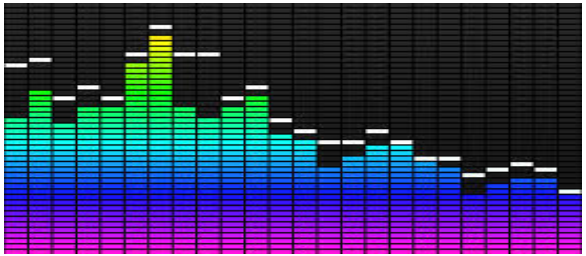
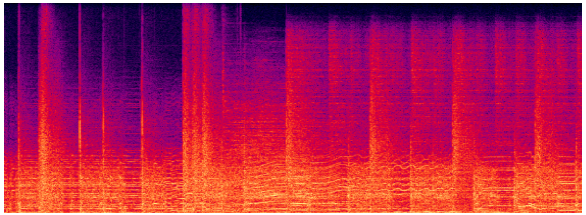
Визуализация звукового сигнала

Виды визуализации

Спектрограммы

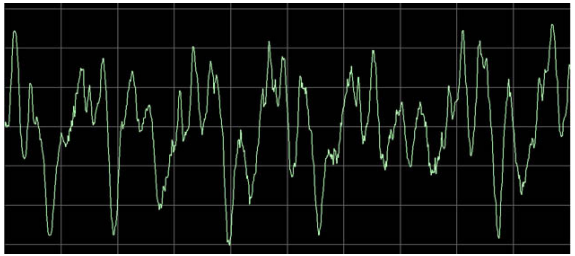


2D

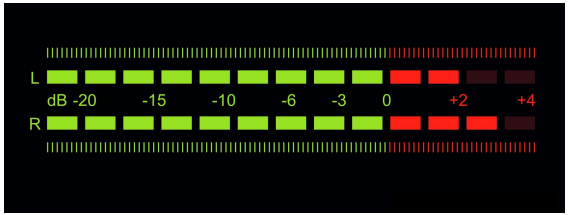


3D

Амплитудные



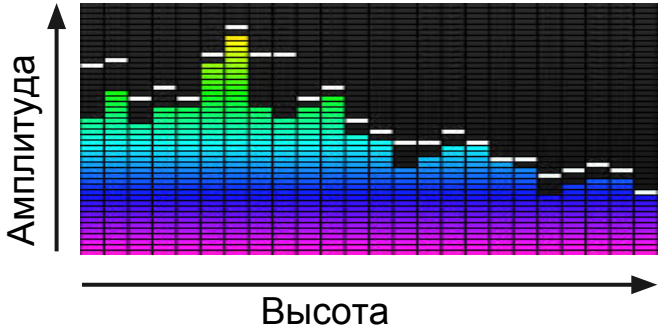
2D



1D

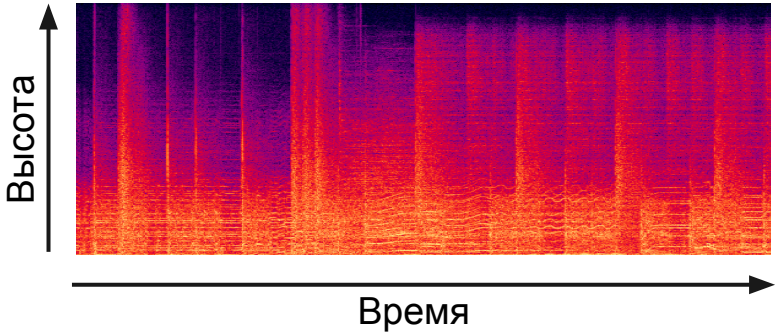
Спектрограммы

Двумерная



Трехмерная

Яркость соответствует амплитуде



Сторонние библиотеки

Web Audio API - высокоуровневая JS библиотека для обработки и создания звука в web-приложениях.

Chroma - JS библиотека для работы с цветами в визуализациях.

Построение спектрограмм

1. Применить к данным оконную функцию
2. Применить преобразование Фурье
3. Сгладить результат по времени (опционально)

Web Audio API

Главная парадигма - использование нескольких объектов AudioNode для построения пути движения звука, который и задает конечный результат.

Необходимые AudioNode

AudioBufferSource - чтение данных из файла.

Analyser - частотный и временно-областной анализ в реальном времени.

ScriptProcessor - установка интервала вызова функции отрисовки.

AnalyserNode

Вычисление частотных данных:

1. Применение оконной функции Блэкмана
2. Применение преобразования Фурье
3. Сглаживание данных по времени

Blackman window

```
function blackmanWindow(buffer) {
  var alpha = 0.16;
  var a0 = 0.5 * (1.0 - alpha);
  var a1 = 0.5;
  var a2 = 0.5 * alpha;

  for (var i = 0; i < buffer.length; i++) {
    var x = i / buffer.length;
    buffer[i] *= a0 - a1 * Math.cos(2 * Math.PI * x) + a2 * Math.cos(4 * Math.PI * x);
  }
}
```

Преобразование Фурье

$$X(k) = \sum_{t=0}^{n-1} x(t) e^{-2\pi i t k / n}.$$

Xk - сигнал по частотам, x(t) - сигнал по времени.

С помощью формулы Эйлера:

$$x(t)\, e^{-2\pi i t k/n} = \left[\mathrm{Re}(x(t)) + i\,\mathrm{Im}(x(t))\right] \left[\cos\left(2\pi \frac{tk}{n}\right) - i \sin\left(2\pi \frac{tk}{n}\right)\right]$$

Преобразование Фурье

```
static void dft(double[] inreal, double[] inimag, double[] outreal, double[] outimag) {
    int n = inreal.length;
    for (int k = 0; k < n; k++) {  // For each output element
        double sumreal = 0;
        double sumimag = 0;
        for (int t = 0; t < n; t++) {  // For each input element
            double angle = 2 * Math.PI * t * k / n;
            sumreal +=  inreal[t] * Math.cos(angle) + inimag[t] * Math.sin(angle);
            sumimag += -inreal[t] * Math.sin(angle) + inimag[t] * Math.cos(angle);
        }
        outreal[k] = sumreal;
        outimag[k] = sumimag;
    }
}
```

Сглаживание по времени

- Пусть outputBuffer - массив, содержащий данные по частотам, доступный с помощью getFloatFrequencyData или getByteFrequencyData.
- Пусть lastOutputBuffer - результат этой операции на предыдущем блоке. **Предыдущий блок** по определению - э то буфер, возвращаемый предыдущей операцией сглаживания по времени, или массив из fftSize нулей, если это первый раз выполнения операции сглаживания по времени.

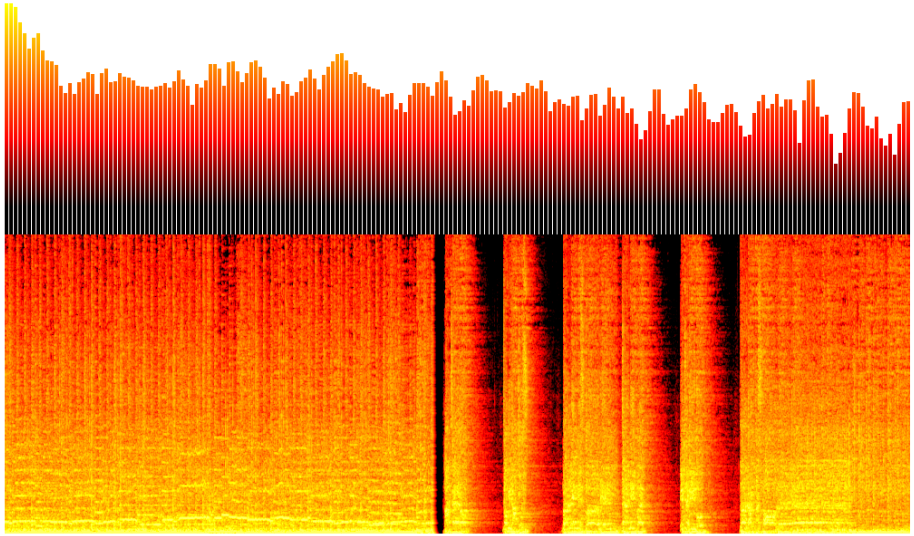
```
function smoothingOverTime(outputBuffer, lastOutputBuffer, smoothingConstant, real, imag, fftSize) {
    for (var i = 0; i < outputBuffer.length; i++) {
      var magnitude = Math.sqrt(real[i] * real[i] + imag[i] * imag[i]) / fftSize;
      outputBuffer[i] = smoothingTimeConstant * lastOutputBuffer[i] + (1.0 - smoothingTimeConstant) * magnitude;
    }
  }
```

Отрисовка

Двумерная перерисовывается с нуля со smoothingTime = 0.3

Трехмерная смещается влево на пиксель, затем рисуется пиксельная полоска, smoothingTime = 0.0

Конечный результат



Sabaton - Attero Dominatus