**Липецкий государственный технический университет**  
Факультет автоматизации и информатики  
Кафедра автоматизированных систем управления

Лабораторная работа №4

по дисциплине «Организация графических систем и систем мультимедиа»

Обработка потокового аудио

Пархомов А.С.

Студент

Группа М-АС-21

Кургасов В.В.

Руководитель

Доцент

Липецк 2022 г.

Цель работы

Изучить способы обработки и визуализации потокового аудио.

Задание кафедры

Разработать приложение для обработки и визуализации потокового аудио. Программа должна иметь графический интерфейс пользователя с возможностью применения аудио-фильтров.

Содержание

[1 Теоретические сведения 5](#_Toc103557161)

[2 Ход работы 5](#_Toc103557162)

[2.1 Использование приложения 6](#_Toc103557163)

[2.2 Создание Docker-контейнера 8](#_Toc103557164)

[Вывод 9](#_Toc103557165)

[Приложение А 10](#_Toc103557166)

1. Теоретические сведения

В качестве языка программирования для реализации проекта был выбран язык программирования JavaScript со следующим набором библиотек: Analyzer, AnalyzerWL, CanvasEqualizer, FFTNR, Shaders, SoundParticles, WaveExporterWorker. Кроме того, приложение использует WebGL для визуализации аудио-потока. Данные инструменты были выбраны для достижения высокой производительности с использованием современных технологий. Разработанное приложение упаковано в Docker-контейнер, что позволяет запустить его на любой системе с установленным Docker Engine и браузером, поддерживающим WebGL. Исходный код основной части программы представлен в приложении А.

WebGL – программная библиотека для языка JavaScript предназначенная для визуализации интерактивной трёхмерной графики и двухмерной графики в пределах совместимости веб-браузера без использования плагинов. WebGL приносит в веб трёхмерную графику, вводя API, который построен на основе OpenGL ES 2.0, что позволяет его использовать в элементах Canvas HTML5.

Для обработки аудио-потока и применения фильтров используется быстрое преобразование Фурье и алгоритм прореживания по частоте.

1. Ход работы

Разработанное приложение представляет собой интерфейс пользователя для визуализации и изменения потокового аудио. Поток может быть считан как из файла (конечный поток), так и по URL-адресу (бесконечный поток – например, онлайн радио). Кроме того, приложение поддерживает обработку аудиофайла оффлайн с его последующим сохранением в формате WAV.

* 1. Использование приложения

На рисунках 1-5 показана работа приложения с разными параметрами.

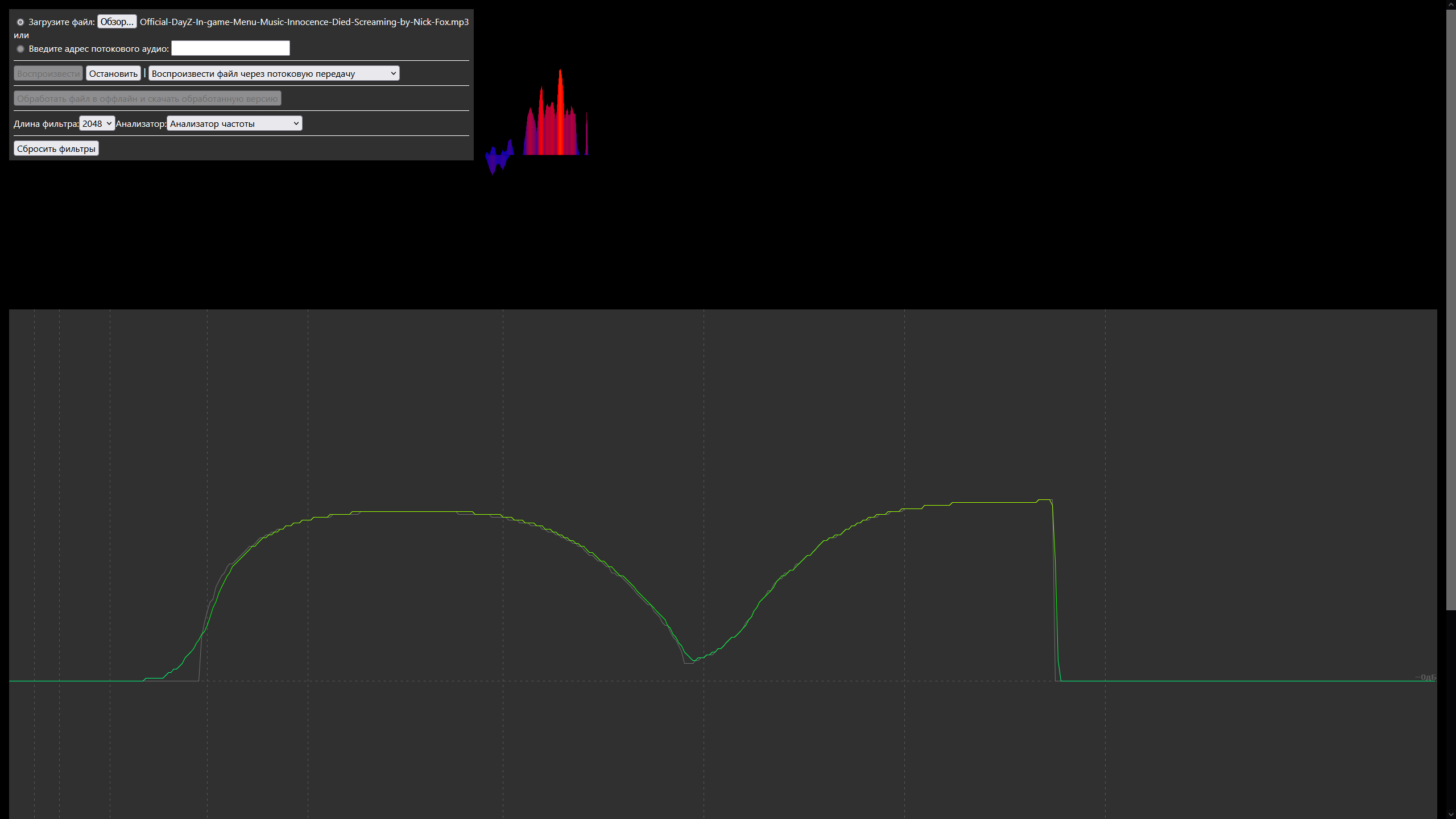


Рисунок 1 – Загруженный аудиофайл с визуализацией и фильтром по средним частотам

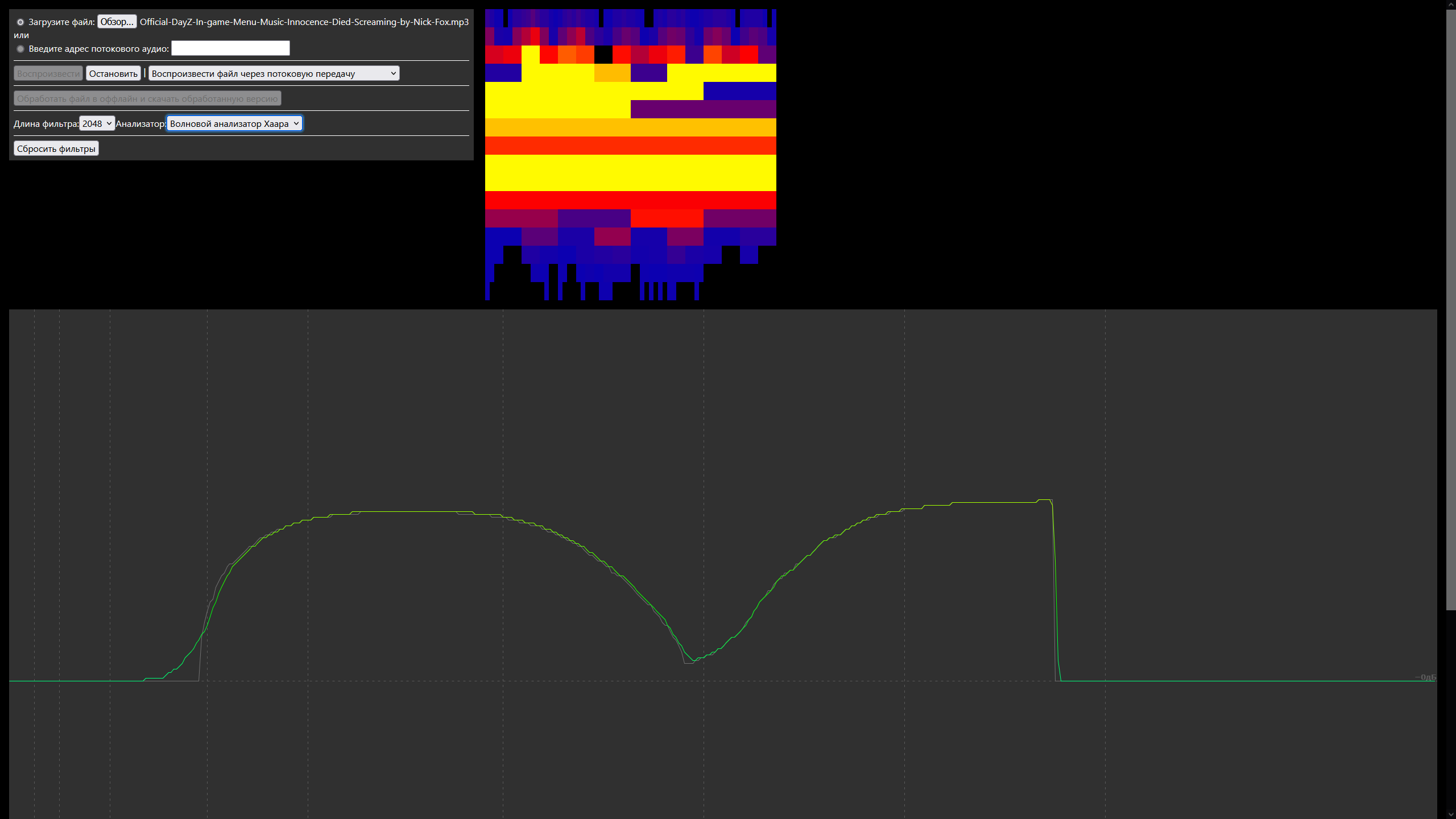


Рисунок 2 – Визуализация аудио с помощью волнового анализатора Хаара

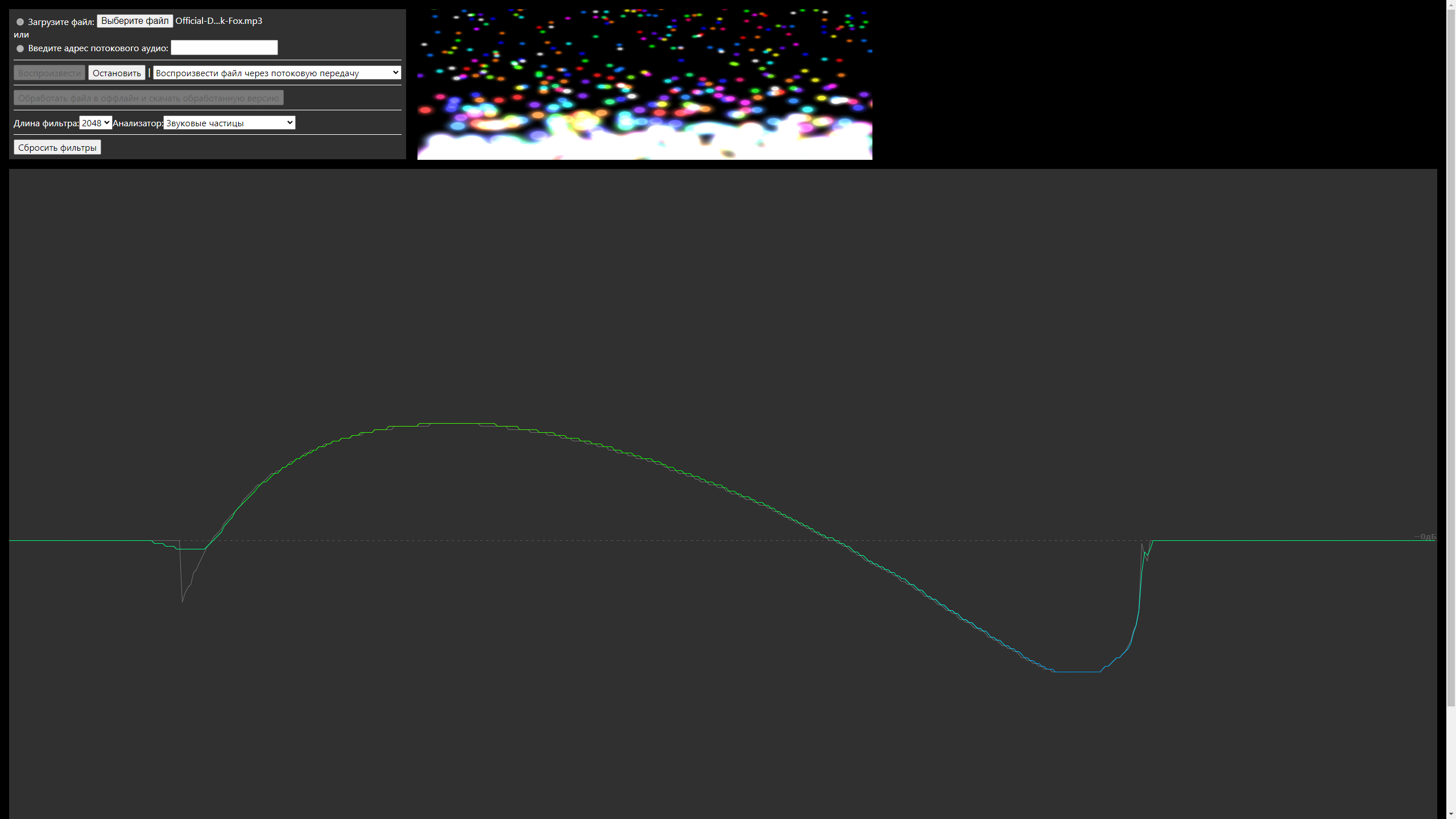


Рисунок 3 – Визуализация аудио с помощью звуковых частиц

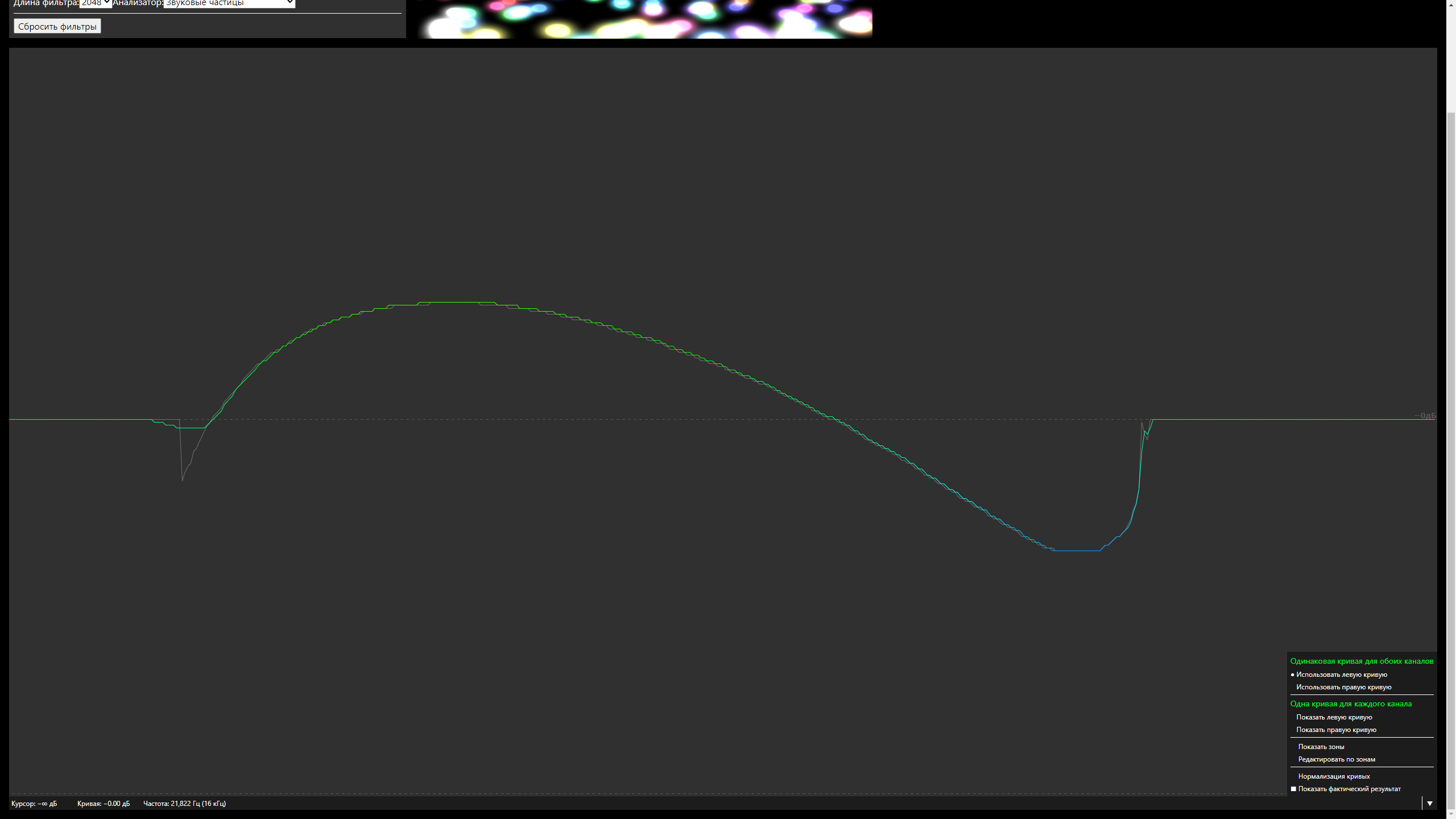


Рисунок 4 – Меню приложения с возможностью отображения частотной сетки и разделением фильтров по каналам

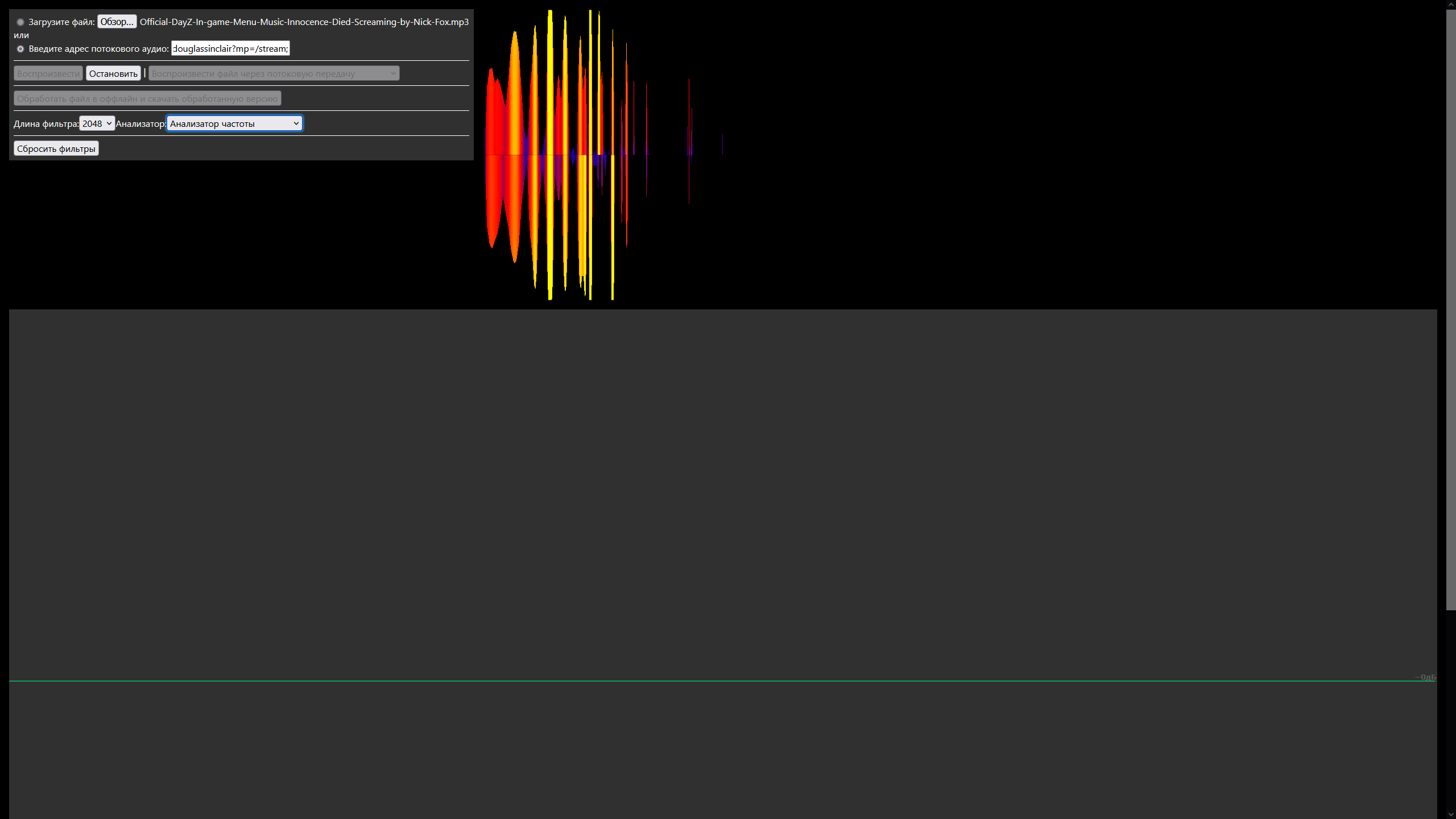


Рисунок 5 – Визуализация потокового аудио по URL-адресу

* 1. Создание Docker-контейнера

Для удобного распространения разработанного приложения был создан Docker-контейнер. Файл сборки образа контейнера:

FROM nginx:1.21.5

COPY . /usr/share/nginx/html

EXPOSE 80

Для запуска проекта необходимо выполнить следующие команды:

1. Сборка docker-образа: docker build . -t graphicslb4\_equalizer.
2. Создание и запуск контейнера с проектом на порту 8080: docker run -d -p 8080:80 --name equalizer graphicslb4\_equalizer.

Вывод

В ходе выполнения данной лабораторной работы были изучены способы обработки и визуализации потокового аудио и разработано приложение, представляющее собой интерфейс пользователя для визуализации и изменения потокового аудио. Поток может быть считан как из файла (конечный поток), так и по URL-адресу (бесконечный поток – например, онлайн радио). Кроме того, приложение поддерживает обработку аудиофайла оффлайн с его последующим сохранением в формате WAV.

Также был создан Docker-контейнер для удобного распространения разработанного приложения.

Приложение А

Исходный код основного файла приложения

Исходный код файла main.js:

"use strict";

// объявление глобальных переменных

var audioContext, source, sourceAudio, graphicEqualizer, splitter, analyzer, analyzerType,

    merger, pendingUrls, chkSource, ignoreNextConvolverChange = false;

// функция получения элемента по его id

function $(e) {

    return document.getElementById(e);

}

// заглушка аудио-контекста для избежания ошибок

function fakeAudioContext() {

}

// аудио-контекст-заглушка

fakeAudioContext.prototype = {

    sampleRate: 44100,

    createChannelSplitter: function () {

        return {};

    },

    createChannelMerger: function () {

        return {};

    },

    createBufferSource: function () {

        return {};

    },

    createBuffer: function (channels, filterLength, sampleRate) {

        if (sampleRate === undefined)

            return this.createBuffer(2, 1024, this.sampleRate);

        return {

            duration: filterLength / sampleRate,

            gain: 1,

            length: filterLength,

            numberOfChannels: channels,

            sampleRate: sampleRate,

            data: (function () {

                var a = new Array(channels), i;

                for (i = channels - 1; i >= 0; i--)

                    a[i] = new Float32Array(filterLength);

                return a;

            })(),

            getChannelData: function (index) { return this.data[index]; }

        };

    },

    createConvolver: function () {

        var mthis = this;

        return {

            buffer: null,

            context: mthis,

            normalize: true,

            numberOfInputs: 1,

            numberOfOutputs: 1

        };

    }

};

// главная функция скрипта

function main() {

    // инициализация слушателей и создание элементов управления

    pendingUrls = [];

    $("btnPlay").addEventListener("click", play);

    $("btnStop").addEventListener("click", stop);

    $("btnProcess").addEventListener("click", processAndDownload);

    $("btnReset").addEventListener("click", reset);

    chkSource = [$("chkSource0"), $("chkSource1")];

    chkSource[0].addEventListener("change", chkSource\_Change);

    chkSource[1].addEventListener("change", chkSource\_Change);

    $("txtFile").addEventListener("change", txtFile\_Change);

    $("txtURL").addEventListener("change", txtURL\_Change);

    $("cbFilterLength").addEventListener("change", filterLengthChanged);

    $("cbAnalyzer").addEventListener("change", updateConnections);

    audioContext = (window.AudioContext ? new AudioContext() : (window.webkitAudioContext ? new webkitAudioContext() : new fakeAudioContext()));

    graphicEqualizer = new CanvasEqualizer(2048, audioContext, {

        filterOptions: {

            convolverCallback: updateConnections

        }

    });

    graphicEqualizer.createControl($("equalizerPlaceholder"));

    analyzerType = null;

    analyzer = null;

    splitter = audioContext.createChannelSplitter();

    merger = audioContext.createChannelMerger();

    return true;

}

// обработчик события изменения источника данных

function chkSource\_Change() {

    var e = (chkSource[1].checked ? "disabled" : "");

    $("cbLoadType").disabled = e;

    $("btnProcess").disabled = e;

    return true;

}

// установка флага выбранного источника данных

function selectSource(index) {

    chkSource[index].checked = true;

    return chkSource\_Change();

}

// обработчик события изменения пути до файла

function txtFile\_Change() {

    return selectSource(0);

}

// обработчик события изменения url до файла

function txtURL\_Change() {

    return selectSource(1);

}

// очистка элемента анализатора звука

function cleanUpAnalyzer() {

    if (analyzer) {

        analyzer.stop();

        analyzer.destroyControl();

    }

    splitter.disconnect(0);

    splitter.disconnect(1);

    if (analyzer) {

        analyzer.analyzerL.disconnect(0);

        analyzer.analyzerR.disconnect(0);

        analyzerType = null;

        analyzer = null;

    }

    merger.disconnect(0);

    return true;

}

// активация кнопок

function enableButtons(enable) {

    var e = (enable ? "" : "disabled");

    $("btnPlay").disabled = e;

    $("btnProcess").disabled = e;

    $("btnStop").disabled = e;

    chkSource[0].disabled = e;

    chkSource[1].disabled = e;

    return true;

}

// отображение иконки загрузки

function showLoader(show) {

    $("imgLoader").className = (show ? "" : "HID");

    return true;

}

// создание объекта url

function createObjURL(obj, opts) {

    var url = (window.URL || window.webkitURL), objurl = (opts ? url.createObjectURL(obj, opts) : url.createObjectURL(obj));

    pendingUrls.push(objurl);

    return objurl;

}

// очистка созданных объектов url

function freeObjURLs() {

    if (pendingUrls.length) {

        var i, url = (window.URL || window.webkitURL);

        for (i = pendingUrls.length - 1; i >= 0; i--)

            url.revokeObjectURL(pendingUrls[i]);

        pendingUrls.splice(0, pendingUrls.length);

    }

    return true;

}

// остановка проигрывания аудио

function stop() {

    enableButtons(true);

    if (sourceAudio) {

        sourceAudio.pause();

        sourceAudio = null;

        source.disconnect(0);

        source = null;

    } else if (source) {

        source.stop(0);

        source.disconnect(0);

        source = null;

    }

    graphicEqualizer.convolver.disconnect(0);

    // освободить все созданные URL

    freeObjURLs();

    return cleanUpAnalyzer();

}

// обработчик ошибок

function handleError(e) {

    showLoader(false);

    enableButtons(true);

    // освободить все созданные URL

    freeObjURLs();

    notyAlert(e, 'error');

    return true;

}

// функция вывода ошибки на экран

function notyAlert(e, type) {

    new Noty({

        theme: 'metroui',

        timeout: 5000,

        type: type,

        text: e,

    }).show();

}

// обновление источников данных

function updateConnections() {

    var t = $("cbAnalyzer").value;

    if (!source || ignoreNextConvolverChange) return false;

    source.disconnect(0);

    source.connect(graphicEqualizer.convolver, 0, 0);

    graphicEqualizer.convolver.disconnect(0);

    switch (t) {

        case "soundParticles":

        case "fft":

        case "wl":

            if (analyzerType !== t) {

                if (analyzer) cleanUpAnalyzer();

                analyzerType = t;

                switch (t) {

                    case "soundParticles":

                        analyzer = new SoundParticles(audioContext, graphicEqualizer);

                        break;

                    case "fft":

                        analyzer = new Analyzer(audioContext, graphicEqualizer);

                        break;

                    case "wl":

                        analyzer = new AnalyzerWL(audioContext, graphicEqualizer);

                        break;

                }

                analyzer.createControl($("analyzerPlaceholder"));

            }

            graphicEqualizer.convolver.connect(splitter, 0, 0);

            splitter.connect(analyzer.analyzerL, 0, 0);

            splitter.connect(analyzer.analyzerR, 1, 0);

            analyzer.analyzerL.connect(merger, 0, 0);

            analyzer.analyzerR.connect(merger, 0, 1);

            merger.connect(audioContext.destination, 0, 0);

            return analyzer.start();

        default:

            graphicEqualizer.convolver.connect(audioContext.destination, 0, 0);

            return cleanUpAnalyzer();

    }

}

// обработчик события изменения длины используемого фильтра

function filterLengthChanged() {

    graphicEqualizer.filterLength = parseInt($("cbFilterLength").value);

    return true;

}

// функция завершения загрузки файла в память и его проигрывания

function finishLoadingIntoMemoryAndPlay(array, name, offline) {

    try {

        // декодировать массив асинхронно

        audioContext.decodeAudioData(array, function (buffer) {

            try {

                if (offline) {

                    // начинаем обработку декодированного буфера в оффлайне

                    var offlineAudioContext = (window.OfflineAudioContext ? new OfflineAudioContext(buffer.numberOfChannels, buffer.length, buffer.sampleRate) : (window.webkitOfflineAudioContext ? new webkitOfflineAudioContext(buffer.numberOfChannels, buffer.length, buffer.sampleRate) : null));

                    if (!offlineAudioContext)

                        return handleError("Автономная обработка звука не поддерживается!");

                    source = offlineAudioContext.createBufferSource();

                    source.buffer = buffer;

                    source.loop = false;

                    ignoreNextConvolverChange = true;

                    graphicEqualizer.audioContext = offlineAudioContext;

                    ignoreNextConvolverChange = false;

                    source.connect(graphicEqualizer.convolver, 0, 0);

                    graphicEqualizer.convolver.connect(offlineAudioContext.destination, 0, 0);

                    source.start(0);

                    offlineAudioContext.oncomplete = function (renderedData) {

                        var worker = new Worker("libs/WaveExporterWorker.js"),

                            leftBuffer = renderedData.renderedBuffer.getChannelData(0).buffer,

                            rightBuffer = ((renderedData.renderedBuffer.numberOfChannels > 1) ? renderedData.renderedBuffer.getChannelData(1).buffer : null);

                        worker.onmessage = function (e) {

                            showLoader(false);

                            enableButtons(true);

                            // обходной путь для сохранения файла - программно кликаем на ссылку

                            var a = document.createElement("a"), i = name.lastIndexOf("."), evt;

                            a.href = createObjURL(new Blob(e.data, { type: "application/octet-stream" }));

                            a.download = ((i > 0) ? (name.substring(0, i) + " - (Filtered).wav") : "FilteredFile.wav");

                            evt = document.createEvent("MouseEvents");

                            evt.initMouseEvent("click", true, false, window, 0, 0, 0, 0, 0, false, false, false, false, 0, null);

                            a.dispatchEvent(evt);

                            return true;

                        };

                        worker.postMessage({

                            left: leftBuffer,

                            right: rightBuffer,

                            length: renderedData.renderedBuffer.length,

                            sampleRate: (renderedData.renderedBuffer.sampleRate | 0)

                        }, [ leftBuffer, rightBuffer ]);

                        return true;

                    };

                    offlineAudioContext.startRendering();

                } else {

                    // воспроизведение декодированного буфера

                    source = audioContext.createBufferSource();

                    source.buffer = buffer;

                    source.loop = true;

                    graphicEqualizer.audioContext = audioContext;

                    updateConnections();

                    source.start(0);

                    showLoader(false);

                    $("btnStop").disabled = "";

                }

            } catch (e) {

                handleError(e);

            }

            return true;

        }, function () {

            return handleError("Ошибка чтения файла!");

        });

    } catch (e) {

        handleError(e);

    }

    return true;

}

// функция загрузки файла в память и его проигрывание

function loadIntoMemoryAndPlay(offline) {

    var r, f, done = false;

    showLoader(true);

    f = $("txtFile").files[0];

    // считать выбранный файл в память

    r = new FileReader();

    r.onload = function () {

        done = true;

        finishLoadingIntoMemoryAndPlay(r.result, f.name, offline);

        return true;

    };

    r.onerror = function () {

        return handleError("Ошибка чтения файла!");

    };

    r.onloadend = function () {

        if (!offline && !done)

            showLoader(false);

        return true;

    };

    r.readAsArrayBuffer(f);

    return true;

}

// подготовка и проигрывание потокового аудио

function prepareStreamingAndPlay() {

    if (chkSource[0].checked) {

        // создаём временный url для выбранного файла

        sourceAudio = new Audio(createObjURL($("txtFile").files[0]));

    } else {

        sourceAudio = new Audio($("txtURL").value);

    }

    sourceAudio.crossOrigin = "anonymous";

    sourceAudio.loop = true;

    source = audioContext.createMediaElementSource(sourceAudio);

    sourceAudio.load();

    graphicEqualizer.audioContext = audioContext;

    updateConnections();

    sourceAudio.play().catch(function(error) {

        notyAlert('Ошибка воспроизведения аудио-потока!', 'error')

    });

    $("btnStop").disabled = "";

    return true;

}

// воспроизведение аудио

function play() {

    if (chkSource[0].checked) {

        if ($("txtFile").files.length === 0) {

            notyAlert("Пожалуйста, выберите файл для воспроизведения!", 'warning');

            return true;

        }

    } else if (chkSource[1].checked) {

        if ($("txtURL").value.length === 0) {

            notyAlert("Пожалуйста, введите адрес файла для воспроизведения!", 'warning');

            return true;

        }

    }

    if (!window.AudioContext && !window.webkitAudioContext) {

        notyAlert("Ваш браузер не поддерживает Web Audio API!", 'error');

        return true;

    }

    stop();

    enableButtons(false);

    try {

        if (!chkSource[1].checked && parseInt($("cbLoadType").value))

            loadIntoMemoryAndPlay(false);

        else

            prepareStreamingAndPlay();

    } catch (e) {

        handleError(e);

    }

    return true;

}

// функция обработки и загрузки файла

function processAndDownload() {

    if (chkSource[0].checked) {

        if ($("txtFile").files.length === 0) {

            notyAlert("Выберите файл для обработки!", 'warning');

            return true;

        }

    } else if (chkSource[1].checked) {

        notyAlert("Извините, но невозможно обработать аудио-поток оффлайн", 'error');

        return true;

    }

    if (!window.AudioContext && !window.webkitAudioContext) {

        notyAlert("Ваш браузер не поддерживает Web Audio API!", 'error');

        return true;

    }

    if (!window.Worker) {

        notyAlert("Ваш браузер не поддерживает Web Audio API!", 'error');

        return true;

    }

    stop();

    enableButtons(false);

    try {

        loadIntoMemoryAndPlay(true);

    } catch (e) {

        handleError(e);

    }

    return true;

}

// сброс элемента управления

function reset() {

    graphicEqualizer.reset();

}