ПРАВИТЕЛЬСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ДОПУСТИТЬ К ЗАЩИТЕ

Профессор с возложенными обязанностями заведующего Кафедрой информационных систем в искусстве и гуманитарных науках

_____(Борисов Н.В.)

“_____”_____20__ г.

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

Направление 09.03.03 «Прикладная информатика»

Уровень Бакалавриат

Основная образовательная программа

«Прикладная информатика в области искусств и гуманитарных наук»

На тему

«Создание веб-ресурса с аудиовизуальной инсталляцией»

Студента *Ремизова Алексея Олеговича*

_____

(подпись студента)

Руководитель: *канд. физ.-мат. наук, доцент, Захаркина Валентина Валентиновна*

_____

(подпись руководителя)

Рецензент : *общество с ограниченной ответственностью «Русская коллекция СПб», Кудеров Дмитрий Евгеньевич*

_____

(подпись рецензента)

Санкт-Петербург

2017

АННОТАЦИЯ

выпускной квалификационной работы Ремизова Алексея Олеговича

«Создание веб-ресурса с аудиовизуальной инсталляцией»

КЛЮЧЕВЫЕ СЛОВА: HTML5, Web Audio API, ВЕБ-ПРИЛОЖЕНИЯ, ОН-ЛАЙН ВИЗУАЛИЗАТОР МУЗЫКИ, ВИЗУАЛИЗАЦИЯ ЗВУКА В БРАУЗЕРЕ, СОЗДАНИЕ ОБУЧАЮЩЕГО ВЕБ-РЕСУРСА

Целью данной выпускной квалификационной работы является исследование современных тенденций в работе со звуком в браузере и последующее создание веб-ресурса, на котором представлен интерактивный аудиовизуализатор и обучающие материалы, освещающие процесс его разработки. В ходе работы были решены следующие задачи:

- Поиск и изучение программных стандартов, необходимых для визуализации звука средствами браузера.

- Анализ программных библиотек, призванных упростить процесс разработки.
- Краткий обзор истории и подходов к визуализации звука.
- Создание веб-ресурса с интерактивным визуализатором.
- Наполнение веб-ресурса текстовыми обучающими материалами.
- Оптимизация процесса визуализации.

Разработка веб-ресурса была выполнена с использованием технологий HTML5, Web Audio API, языка клиентского программирования Javascript с применением дополнительных программных библиотек, языка серверного программирования PHP на базе CMS Drupal.

СОДЕРЖАНИЕ

ОПРЕДЕЛЕНИЯ

В этом тексте применяются следующие термины с их определениями:

Программная библиотека – сборник подпрограмм, используемых для разработки программного обеспечения. Предназначен для упрощения написания программного кода и расширения функционала в рамках определённой области.

Плагин – независимый программный модуль, расширяющий возможности основной программы. Для работы плагин должен быть загружен и установлен на компьютер.

API - application programming interface, программный интерфейс приложения. Представляет собой набор готовых средств, предоставляемых приложением для работы во внешних программах.

Визуализация – процесс сопоставления звукового потока с визуальными образами.

Визуализатор – устройство или программный продукт выполняющий визуализацию.

ВВЕДЕНИЕ

Проблема визуализации звука по праву может называться классической. Свои силы к созданию модели соотношения звуковой и визуальной информации прилагали выдающиеся умы своих лет, такие как Ньютон, Скрябин и другие. Однако, единая система представления аудиовизуальной информации, которая бы использовалась повсеместно, так и не была создана. До недавнего времени можно было выделить два подхода к процессу визуализации.

При первом подходе композитор самостоятельно пишет цветовую партитуру для своего музыкального произведения, основываясь на собственных представлениях о сочетании звука и музыки. Самым известным примером использования такого подхода является музыкальная поэма «Прометей» Александра Николаевича Скрябина, созданная благодаря его подверженности такому нейрологическому феномену как синестезия. Второй подход предполагает соблюдение формального соответствия мелодией, тональностей, гармонией и цветом. То есть, выстраивается некая математическая модель, которая описывает закономерности визуального отображения музыки. Примером реализации могут служить работы Валентина Владимировича Афанасьева, которые основаны на разработанной им же системе функциональных соотношений цветов и звуков.

Однако, вместе с развитием вычислительной техники, появился и новый, третий подход, основанный на компьютерном анализе звукового потока. Используя его, можно получить информацию о частотных характеристиках

звука и перепадах громкости, чтобы в дальнейшем использовать эти данные для визуализации.

Использование третьего подхода представляет возможность визуализации в реальном времени, реализуемую посредством алгоритмического метода. Это поистине инновационная методика, поскольку раньше для создания визуализации была необходима подготовка для каждого отдельного музыкального произведения, требовавшая подбора сочетаний цвета и музыки вручную. Теперь же появляется возможность один создать программу, которая будет заниматься визуализацией, и применять её для каждого музыкального произведения.

Более того, такой подход можно развить и дальше, разместив данную программу в Интернете и сделав доступной для широкого числа пользователей. Таким образом, мы вплотную подошли к цели данной работы – созданию веб-ресурса, посвящённому визуализации звука средствами браузера. Для реализации этого проекта были поставлены следующие задачи:

- Поиск и изучение программных стандартов, необходимых для визуализации звука средствами браузера.
- Анализ программных библиотек, призванных упростить процесс разработки.
- Краткий обзор истории и подходов к визуализации звука.
- Создание веб-ресурса с интерактивным визуализатором.
- Наполнение веб-ресурса обучающими текстовыми материалами.
- Оптимизация процесса визуализации.

В результате проведённой работы создан веб-ресурс, имеющий как материалы теоретического характера, призванные осветить технический аспект создания визуализатора, базирующегося в браузере, так и практического, представляющие интерактивный конструктор средств визуализации.

КРАТКИЙ ЭКСКУРС В ИСТОРИЮ ЦВЕТОМУЗЫКИ

Интеграция различных областей научного знания, начавшаяся в XX веке, на самом деле имеет корни, уходящие далеко в прошлое. Ведь ещё в древности люди возмещали недостаток одного вида информации другим. И в настоящее время знания о человеке всё ещё остаются недостаточными для прогнозирования поведения как одной отдельно взятой личности, так и целых групп или сообществе. Наверное, практически ничем не отличаются реакции современного человека от поведения далеких предков на такие природные явления, как гроза, извержение вулкана и т.п. В такие моменты у него практически одновременно и внезапно возникают зрительные и звуковые ощущения. Очевидно, что в результате таких воздействий определённым образом меняется его психическое и душевное состояние, может появится инстинктивное побуждение к тем или иным действиям.

Первые ростки идеи единства визуального и звукового восприятия находят своё отражение в глубине веков. Как известно, в древности искусство не делилось на роды и виды, оно было синкретическим. Цвет и звук в сознании первобытного человека соотносились с определённым предметам с вполне определённым восприятием, поэтому отблески костра, различные ритуальные пляски и песнопения являлись для него нераздельными и исполнялись лишь в соответствующих случаях. Все эти образы, мелькающие краски и звуки объединялись воедино в искусстве песни и танца.

Значительно позднее, в древнегреческом искусстве соединение света и звука перешло в театр, где игра актёров, пение, движение и световые эффекты подчинялись определенной задумке. Более того, эту тему не обошли стороной и греческие философы. Так, Аристотель писал в своём труде «О душе» о том, что

цвета по привлекательности их соотношения могут соотноситься друг с другом подобно музыкальным созвучиям и быть взаимно пропорциональными.

Античные концепции связи света и музыки исторически и логически связаны с космологическим учением “музыки сфер”, в котором признавалось изначальное совершенство мира в соответствии с определёнными эстетическими критериями. Соотношения внутри музыкальной гаммы, которые были обнаружены пифагорейцами, стали объяснением всего сущего.
Они считали, что “Гармонический” ряд определяет устройство космоса, в котором вращаются планеты издавая “музыку сфер”, строение человеческого тела, архитектуру и структуру чувственного восприятия.

Пифагорейцы были уверенны, что математика представляет собой инструмент, с помощью которого и была создана Вселенная. По их мнению, гармония базируется на числах, потому что их законы руководят гармоническими пропорциями. Ключ к ним лежит в знаменитом тетрактисе, составленном из чисел 1, 2, 3 и 4. Соотношения этих чисел вместе создают интервалы октавы, диапенте и диатессарон. В этих соотношениях скрываются представления древних греков о музыке и цвете. Первые три точки представляют тройной Белый Свет, который является Божественным Главой, содержащим потенциально как свет, так и цвет. Оставшиеся семь точек представляют собой цвета спектра и ноты.

Согласно этой системе, именно тона и цвета взятые таким образом являются основой вселенной. Эти семь точек образуют две группы, из трёх и четырёх точек соответственно. Первая группа раскрывает духовную составляющую сотворенной вселенной, из четырёх точек второй образуется иррациональная сфера, или низший мир.

В следующий раз тема взаимосвязи света и звука затрагивается в период эпохи Возрождения – времени гениальных художественных и естественно-научных открытий. Считается, что музыка цвета была изобретена в конце XVI века в Милане известным живописец Джузеппе Арчимбальдо. Имея познания в музыке, он воспроизводил своим ученикам различные тональности и

одновременно показывал разноцветные карточки, которые, как он считал, соответствовали определённым звукам.

В XVII веке Исаак Ньютон положил начало научной цветомузыке. В 1665 году, во время своих изысканий связанным с природой света, он при помощи призмы разложил его на спектр. Именно Ньютон провёл соответствие между синусами углов преломления, которые он выделили из семи цветов, с тонов в октаве и установил следующую систему ‘нота-цвет’: ‘до’ — красный, ‘ре’ — фиолетовый, ‘ми’ — синий, ‘фа’ — голубой, ‘соль’ — зелёный, ‘ля’ — жёлтый, ‘си’ — оранжевый. Безусловно, такой подход был чисто механическим, тем не менее, он давал точное установление высоты, или темперацию цветов. Конечно, нашлись и несогласные с теорией Ньютона, такие как Гёте и Бюффон. Одни считали, что соответствия между нотами и цветами случайно, другие – что как и в природе, так и для человеческих чувств звук и света не зависят друг от друга и являются самостоятельными явлениями. Тем не менее, после изобретения А. Кирхером первого в мире проекционного аппарата, получившего название “волшебный фонарь”, идея соотнесения цвета и звука получила распространение среди его современников. Устройство состояло из корпуса с размещённым в нём источником света, который был направлен на специальную пластину из стекла с нанесённым на ней изображением. Полученная таким образом картинка проецировалась на клубы дыма, стены или экраны.

В XVIII стали появляться мысли о том, что цветовая музыка является самостоятельным видом искусства. Луи-Бертран Кастель написал книгу, рассказывающую о проблеме синтеза цвета и звука, носящую название «Клавесин для глаз». В ней он говорил об возможностях воздействия на человека сочетаниями цвета и звука. Такое воздействие, по задумке Кастеля, могло быть настолько красочный, что даже глухой мог бы увидеть музыку и наслаждаться ей, а слепой – ощущать цвета, слушая музыку. В 1734 году Кастелем была предпринята попытка воплотить свою теорию в жизнь в виде цветового клавесина, построенного им самостоятельно. Однако, законченный

вариант не сохранился, тем не менее, её конструкцию получилось повторить Г. Эккартсгаузену. Это изобретение вызвало незамедлительную реакцию в среде музыкантов и учёных, повлекшую за собой оживлённое обсуждение открывшихся перспектив. В итоге это позволило взглянуть на казавшиеся обыденными явления с новых сторон, таких как представления о цвете и звуке с точки зрения физики, зрения и слуха со стороны физиологии и как они сочетаются и взаимодействуют друг с другом. Всё это способствовало общему прогрессу научных представлений о природе человека и об окружающем его мире.

В XIX веке идея о соединении звукового и визуального восприятия получила популярность у психологов, которые изучали явление синестезии. Это событие в совокупности с созданием в конце XIX века первых вариантов цветовых органов зарождает повышенную заинтересованность цветомузыкой в США и странах Европы. Например, ангийский изобретатель Александр Римингтон разработал собственную систему синтеза звука и света, а в 1893 году построил цветовой орган. Аппарат предполагался к использованию совместно с любым другим музыкальным инструментом, поскольку имел возможность воспроизводить только цветовую часть произведения. В цветовой модели Римингтона не исключалась вольность её интерпретации несмотря на то, что она основывалась на физических соответствиях.

Среди других представителей исследователей цветомузыки стоит выделить Т. Вилфреда, предпринявшего шага для самостоятельного развития цветового изобразительного искусства, отдельно от музыкального сопровождения. В своих работах он пришёл к осознанию необходимости сочетания цвета, формы и движения. Кроме того, Вилфред разработал свой вариант цветового органа. Его конструкция включала в себя клавиатуру, позволяющая вручную задавать цвета и фигуры для отображения, и экран, на который это всё выводилось.

Если рассматривать вопрос развития цветомузыкального искусства конкретно в России, то, безусловно, в первую очередь стоит упомянуть Александра Николаевича Скрябина. Безусловно, он не был единственным, кого

интересовала эта тема, так, например, незадолго до него таким же вопросом занимался Николай Андреевич Римский-Корсаков. Они оба могли воспринимать звук и цвет как одно целое, то есть имели синестезию. Однако, Римский-Корсаков придерживался другого подхода в цветомузыке основываясь в своих работах на систему, похожую на предложенную Исааком Ньютоном.

Однако именно Скрябин создал произведение, которое впервые сочетало в себе как классические музыкальные партии, так и партию цвета, записанную с ними на равных в одной музыкальной партитуре. Этим произведением была симфоническая поэма «Прометей». По задумке Скрябина такое сочетание должно было вывести впечатление от прослушивания музыки на новый уровень. К сожалению, в те годы несовершенство технической составляющей не позволило в полной мере воплотить в жизнь планы композитора – цветовая партитура не была сыграна так, как было запланировано. Лишь десятилетия спустя, уже после смерти Скрябина, «Прометей» был исполнен в своём изначальном виде, с полноценным исполнением не только музыкальной части, но и световой.

В противоположность Римскому-Корсакову, для Скрябина тональность была не представлена не одним цветом, а и смесью. Например, ‘соль’ — была красно-оранжевой, ‘ля’ — жёлто-зелёной и так далее. Тональности с большим количеством ключевых знаков он причислял к ультрафиолетовым и ультракрасным частям спектра. В своём следующем произведении «Мистерия» Скрябин хотел добиться синтеза звука и света, танцев и драматических действий. По сути, это было возвращение к синкретическому искусству древнего мира, упомянутого в начале главы. Однако, это сочинение так и не было им закончено, а его идея была реализована уже в наше время.

С приходом XX века изменилось направление идеи о совместном использовании цвета, света и музыки, она стала ближе к техническим экспериментам. В 80-е годы открываются школы цветомузыки как в пределах России, так и за границей. В разные периоды к этой теме обращаются такие деятели искусств, как Сергей Эйзенштейн, поставивший в 1940 году оперу

Вагнера со светомузыкой, проводятся эксперименты по синтезу света и звука Пьером Булезом и Жан-Мишелем Жарром.

Предложения о создании новых систем соответствия нот и цветов не раз представлялись в различном виде и после появления на свет работ Ньютона и Кастеля. Всё чаше музыка рассматривается с точки зрения физики, ведь звук сам по себе есть ни что иное как волна. Поэтому научные исследования музыки принадлежат к области акустики. Однако, до сих пор нет точного ответа на вопрос о том, что же именно делает звук музыкой. Конечно, важны и конкретные частоты колебаний, и высоты звучания. Тем не менее, именно направление на воссоздание процессов бытия позволяет отделить простой шум от музыки. Из этого следует современная формулировка музыкальных звуков – это смесь гармонических механических колебаний, частоты которых относятся как небольшие целые числа и вызывают у человека приятные ощущения, причиной которых являются соответствия определённым биоритмам. При этом, близкие, но негармоничные колебания вызывают неприятные ощущения диссонанса, а звук со сплошными спектрами частот слышится человеку как шум.

При этом, современная нейробиология не может ответить на вопрос, почему музыка так важна для человека, является неотъемлемой частью его жизни. Тем интереснее результаты исследований человеческого мозга показавшие, что в нём нет специализированного центра, отвечающего за обработку звука. В процессе прослушивания музыки у человека задействуются различные области, включая те, которые в основном задействованы для других видов восприятия. На их расположение и размер влияет даже индивидуальный опыт человека и его уровень музыкальной подготовки. Даже кратковременное обучение способно изменить характер переработки мозгом “музыкальных входов”. Хочется так же отметить, что на разные компоненты музыки, такие как частота или ритм, реагируют разные отделы головного мозга. Исходя из этих фактов, становится понятно, что использование цветовых эффектов способно не

просто усилить впечатление от музыки, а вывести его на качественно новый уровень.

Именно поэтому в наши дни так популярно использование различных устройств на всевозможных концертах и выступлениях, с помощью которых на сцену и окружающее её пространство проецируется свет, производя в сочетании с музыкой неизгладимое впечатление на зрителей. В плане технической составляющей, более широкое использование компьютеров позволило не только повысить качество исполнения световой части цветомузыкального представления, но и привело к общему увеличению интереса к этой теме. Стало необязательным приобретать дорогостоящее оборудование к себе домой, чтобы опробовать собственные возможности в плане создания светового сопровождения к музыкальному произведению. Достаточно иметь у себя дома компьютер, чтобы получить доступ к различным программам, выполняющим визуализацию звука в реальном времени с применением алгоритмического подхода. Однако, ранее такие приложения требовали предварительной загрузки и установки, а также были ограничены в настройке. Таким образом, становится понятно, что создание веб-приложения, позволяющего проводить визуализацию средствами браузера является актуальной задачей не только с технической, но и с исторической точки зрения.

ОБЗОР ПРОГРАММНЫХ СРЕДСТВ, ИСПОЛЬЗОВАННЫХ В ПРОЦЕССЕ РАЗРАБОТКИ

Изучение доступной литературы и справочных веб-ресурсов показало, что единственным подходящим для выполнения поставленных целей программным средством, предоставляющим возможность работы со звуком в браузере, является программный интерфейс Web Audio.

Web Audio API это высокоуровневый API JavaScript имеющий широчайшие возможности по синтезированию звука. Помимо управления загрузкой и воспроизведением интерфейс предоставляет возможности для синхронизации, обработки, работы с многоканальным и 3D аудио. Его отличительной особенностью так же является то, что он предоставляет прямой доступ к временным и спектральным характеристикам сигнала, что даёт широкие возможности для анализа звукового потока и визуализации.

На данный момент стандарт находится в состоянии драфта – рабочего черновика – что выражается в отсутствии поддержки некоторыми браузерами и возможных дальнейших изменениях. Последняя выпущенная версия датируется 2015-м годом. Тем не менее, Web Audio активно используется для разработки различных приложений.

API аудио предполагает модульную реализацию, при которой программа состоит из отдельных элементов, называемых аудио узлами (Audio Node). Соединяясь между собой, узлы образуют аудиосистему, воспроизводящую звук и выполняющую различные манипуляции над ним. Простейшая система включает в себя два узла. Первый – это источник звука, которым может быть, например, аудиофайл. Второй узел – приёмник, в роли которого могут выступать колонки или наушники. На следующем изображении представлено

схематическое представление простейшей аудиосистемы, выполненное на сайте Web Audio Playground. Данный ресурс представляет собой интерактивный конструктор, имитирующий устройство реальных звуковых систем любой сложности.

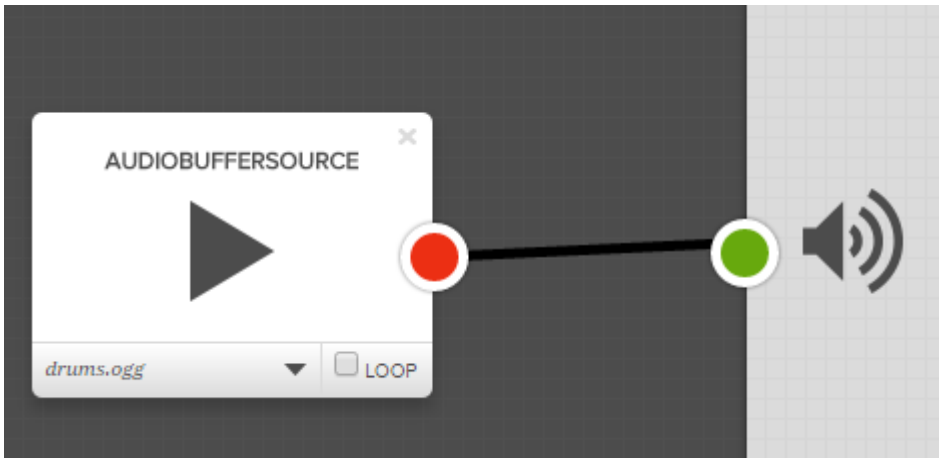

Рисунок 1 – пример простейшей аудиосистемы.

Одновременно в одной аудиосистеме может существовать любое разумное число узлов, необходимое для выполнения поставленной перед ней задачей. Таким образом, на пути между источником и приёмником звук может быть различным образом изменён или проанализирован. Например, система, представленная на следующем изображении воспроизводит одновременно три разных звуковых файла, накладывая на два из них различные эффекты и предоставляя возможность изменять уровень громкости каждого из них.

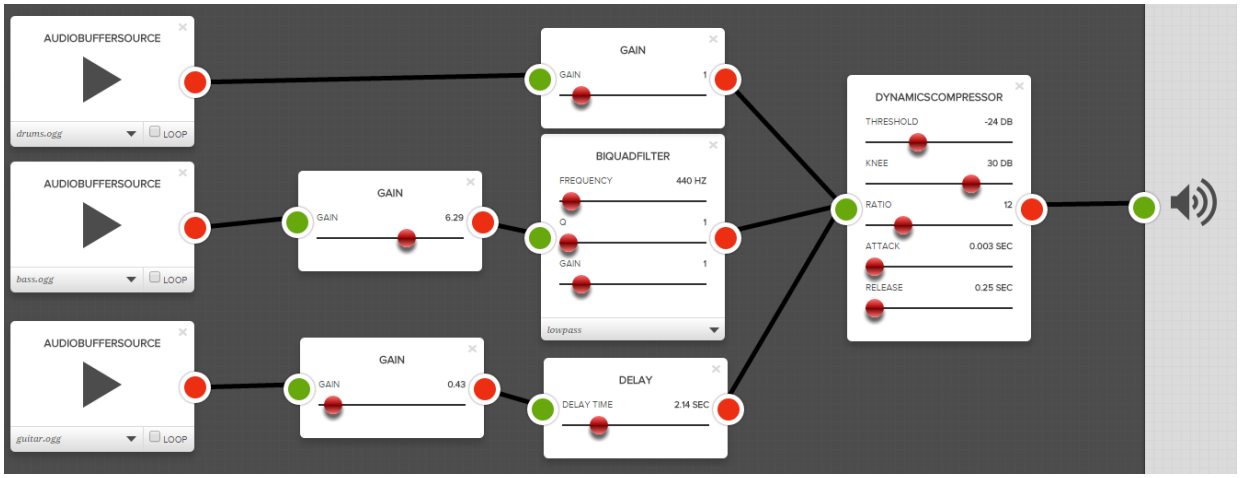

Рисунок 2 – пример сложной аудиосистемы.

Переходя к описанию программной реализации аудиосистемы на основе Web Audio, в первую очередь хочется отметить главенствующую роль класса Audio Context.

Чтобы инициализировать контекст нужно вызвать его конструктор:

```
var context = new (window.AudioContext ||
     window.webkitAudioContext) ();
```

Данный способ создания нужен для возможности его работы как в браузерах на основе движка WebKit (Chrome, Opera, Safari), так и на основе других инструментариев (Firefox). Audio Context способен поддерживать сразу несколько источников звука и обеспечивать работу сложных цепочек узлов, поэтому нет необходимости в создании более чем одного экземпляра на аудиосистему.

Одной из главных задач аудио контекста предоставление возможности для создания аудио узлов. Условно, все узлы можно разделить на несколько типов:

- Узлы-источники – источники аудиоданных. Например, аудио-буферы, микрофоны, элементы <audio> и <video>.
- Изменяющие узлы – фильтры, эхо, кроссфейдинг и тому подобные эффекты.
- Узлы-анализаторы – анализируют звуковой сигнал и предоставляют информацию о нём.
- Узлы назначения – аудиовыходы.

Источником звука в системе не обязательно должен являться аудиофайл, им может быть звук полученный с подключённого к компьютеру музыкального инструмента или микрофона, содержимое элемента <audio> или же полностью синтезированный звук. Так же, хоть звук из узла назначения обычно передаётся на устройства вывода звука, это не обязательно делать если целью является чистая визуализация.

Для создания нового узла необходимо вызвать соответствующий конструктор объекта Audio Context. После этого открывается доступ к изменению его свойств как до начала, так и во время воспроизведения. У каждого узла свой набор преобразуемых параметров, но, как правило, всё сводится к изменению полей объекта. Далее приведён небольшой перечень наиболее часто встречающихся в реальных веб-приложениях узлов.

- Gain Node, усилитель – увеличивает или уменьшает громкость звука. Создание и установка параметров узла :

```
var gainNode = context.createGain();
gainNode.gain.value = 0.7;
```

Значение поля gain может быть динамически установлено в диапазоне от 0 до 1 и определяет мощность усиления.

- Delay Node, линия задержки – задерживает звук на определённое время.

```
var delayNode = context.createDelay();
delayNode.delayTime.value = 2;
```

Значение delayTime определяет насколько секунд будет задержан звуковой сигнал.

- Biquad Filter Node, узел фильтрации – позволяет применить к звуковому сигналу самые распространённые фильтры.

```
var filterNode = context.createBiquadFilter();
filterNode.type = 1;
filterNode.frequency.value = 1000;
filterNode.frequency.Q = 1;
```

Поле type указывает на тип фильтра, имеются следующие значения:

- lowpass(1) – обрезает частоты выше заданной отметки;
- highpass(2) – обрезает частоты ниже заданной отметки;
- bandpass(3) – пропускает определённую полосу частот;

- lowshelf(4) – усиливает или ослабляет частоты ниже указанной отметки;
- highshelf(5) - усиливает или ослабляет частоты выше указанной отметки;
- peaking(6) – усиливает определённую частоту;
- notch(7) – ослабляет определённую частоту;
- allpas(8) – усиливает на определённое значение все частоты сигнала.

Поле frequency определяет частоту фильтра, измеряется в герцах. Q определяет в какой полосе будет применяться изменение сигнала, причём чем выше значение этого параметра, тем уже частота, и наоборот.

Влияние значений этих полей зависит от выбранного фильтра, и не при каждом все их них будут учитываться. Например, в данном примере не учитывается параметр gain, который указывает уровень усиления или ослабления частоты в децибелах.

- Convolver Node, узел свёртки – применяет эффект свёртки к аудио сигналу. Применяется для имитации различных акустических пространств и сложных искажений звука, например шума при проигрывании винилового диска или звучании голоса в телефонной трубке.

```
convolverNode = context.createConvolver();
convolverNode.buffer = buffer;
```

Главным параметром этого узла является импульсная характеристика, которая передаётся в поле buffer как обычный аудиофайл. Именно она определяет какой эффект будет применён к звуку.

Основной способ визуализации в Web Audio API представляет собой использование Analyzer Node для получения частотных или временных

параметров сигнала. Этот узел никак не изменяет сам файл или его звучание, поэтому его можно разместить в любом месте аудиосистемы. Данные для анализа получаются с помощью быстрого преобразования Фурье.

Anazyser Node имеет следующие параметры:

- fftSize – размер буфера со звуковым сигналом, для которого будет проводится преобразования. Может иметь размер от 32 до 32768, значение должно быть степенью двойки. Чем больше размер буфера, тем более детальным будет анализ, но и тем больше времени будет на него затрачено. То есть, слишком большие значения fftSize могут проводить к потерям производительности.
- minDecibells и maxDecibells – максимальное и минимальное значение мощности в диапазоне масштабирования для анализа данных быстрого преобразования Фурье.
- smoothingTimeConstant – может принимать значение от 0 до 1. Чем оно выше, тем сильнее результаты преобразования будут «сглаживаться» и выглядеть более единообразно.

Результаты работы анализатора сохраняются в виде массивов в формате 32-битных чисел с плавающей точкой, либо в виде 8-битных беззнаковых целых. Они создаются следующим образом – new Uint8Array(this.analyser.frequencyBinCount) или new Float32Array(this.analyser.frequencyBinCount). После создания, с помощью функций getByteFrequencyData и getByteTimeDomainData можно сохранить в них значения частотных и временных параметров сигнала.

У каждого узла могут иметься входы и выходы, благодаря которым они соединяются между собой. У узлов-источников есть только один выход и нет входов, у узлов назначения – наоборот, один вход и нет выходов. Кроме того, разработчику не стоит переживать о некоторых аспектах работы с низкоуровневыми потоками аудио, которые возникают при соединении двух

аудиоузлов, поскольку всё это обрабатывается автоматически. К примеру, если моно-аудио сигнал будет передан модулю, который на входе должен принимать стерео-сигнал, то входной сигнал будет соответствующим образом преобразован.

Для соединения узлов у Audio Context существует функция connect(), в качестве аргумента принимающая узел, которому следует подключиться. Ниже представлен фрагмент кода, который создаёт узел-анализатор и соединяет его с источником звука:

```
this.analyser = context.createAnalyser();
this.analyser.connect(context.destination);
```

В качестве назначения указан context.destination – системный звуковой вывод по умолчанию. Таким образом, получилась аудиосистема со следующей структурой:

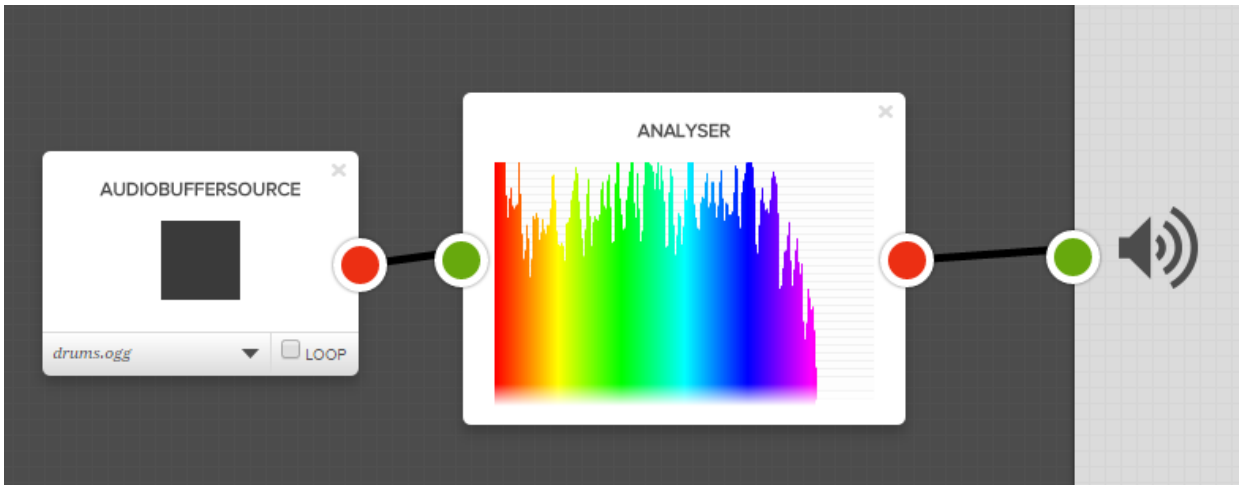

Рисунок 3 – подключение анализатора к аудиосистеме и результат его работы.

При необходимости, имеется возможность динамически менять структуру аудиосистемы с помощью функции disconnect(outputNumber). В качестве аргумента в неё передаётся номер выхода узла, нумерация начинается с нуля.

Для получения возможности использования аудиоданных в звуковой системе необходимо загрузить их в файловый буфер. Следует различать между собой буферы и аудио источники. Каждый конкретный файл может быть представлен

только в одном аудио буфере, но он может проигрываться одновременно на любом количестве аудио источников.

Загрузка аудиофайл происходит асинхронно, поэтому важно следить за тем, чтобы все части готовой программы выполнялись в положенное им время. Конкретная реализация может варьироваться, в данной программе это выполняется с помощью XMLHttpRequest(), после чего результат обрабатывается с помощью функции context.decodeAudioData(). После загрузки файла нужно создать узел-источник с помощью функции context.createBufferNode(), после чего появится возможность начать воспроизведение. Часть программного кода, которая создаёт Audio Context, загружает аудио файл с сервера и начинает его воспроизведение продемонстрирована в приложении 1.1

Являясь программным стандартом, обеспечивающим большое количество различных вариантов манипуляций со звуком средствами браузера, Web Audio API требует написания значительного количества программного кода для их реализации. В целях сокращения его количества и добавления функционала был разработан целый ряд специализированных программных библиотек. Для работы над созданием визуализатора была выбрана библиотека Pizzicato.js Она поддерживается во всех современных браузерах(Firefox 31+, Chrome 31+, Safari 7+, Opera 30+, Edge 13+) и её использование позволило значительно уменьшить объём программного кода, требующегося для написания работоспособной программы. Производимый эффект можно увидеть в приложении 1.2 В нём представлена часть программы из приложения 1.1, переписанная с использованием Pizzicato.js

Вместо создания аудио контекста и загрузки аудиофайла вручную, происходит вызов конструктора Pizzicato.Sound, который принимает на вход файл с песней и загружает его, самостоятельно создаёт контекст, источник звука и устанавливает все связи между узлами. Кроме того, передавая в качестве source различные параметры, можно получить звук, источником которого будет

осциллятор, функция или микрофон (source: ‘wave’, source: ‘script’, source: ‘input’ соответственно).

После загрузки и создания схемы узлов появляется возможность обратиться к аудио контексту, который создан Pizzicato, чтобы подключить дополнительные узлы. Для этого вызывается функция Pizzicato.context, благодаря которой предоставляется доступ к контексту, для работы с которым доступны все стандартные команды Web Audio API. После создания узла происходит его подключение к аудиосистеме командой connect.

Стоит также отметить, что изначально в Web Audio API отсутствуют функции, позволяющие поставить воспроизведение на паузу или продолжить его, поэтому их приходится прописывать самостоятельно. Однако Pizzicato позволяет значительно облегчить этот процесс. Для начала воспроизведения достаточно вызвать функцию play, для паузы и остановки – pause и stop соответственно.

ПОДХОДЫ К ВИЗУАЛИЗАЦИИ ЗВУКА НА ОСНОВЕ АНАЛИЗА ЧАСТОТ

Главным методом визуализации звука с использованием Web Audio API является использование частотных характеристик сигнала, полученных благодаря быстрому преобразованию Фурье. В программе эти данные хранятся в массивах 8-битных беззнаковых целых чисел, поэтому их можно напрямую применять в качестве входных параметров для функций рисования. Несмотря на кажущуюся простоту используемых входных данных (которые представляют из себя числа от 0 до 255), число возможных вариантов конкретной реализации визуализации аудио сигнала ограничено только фантазией разработчика и возможностями аппаратной части. Следующие примеры показывают такие ..., созданные с помощью использования языка программирования JavaScript и технологии Canvas.

Безусловно, первым примером, приходящем на ум при упоминании темы визуализации, является эквалайзер. В данном случае, значения частот используются в функции рисования четырёхугольников drawRect(), в которой они берутся в качестве высоты отрисовываемых столбцов. Поскольку функция требует для работы указания входных данных в виде переменной вида Integer, использование данных из массивов частот является возможным. Готовый программный код можно увидеть в приложении 1.3

Будучи размещённой внутри цикла, который перебирает все элементы массива частот (freqs[]), функция визуализирует буквально каждую частоту аудиофайла.

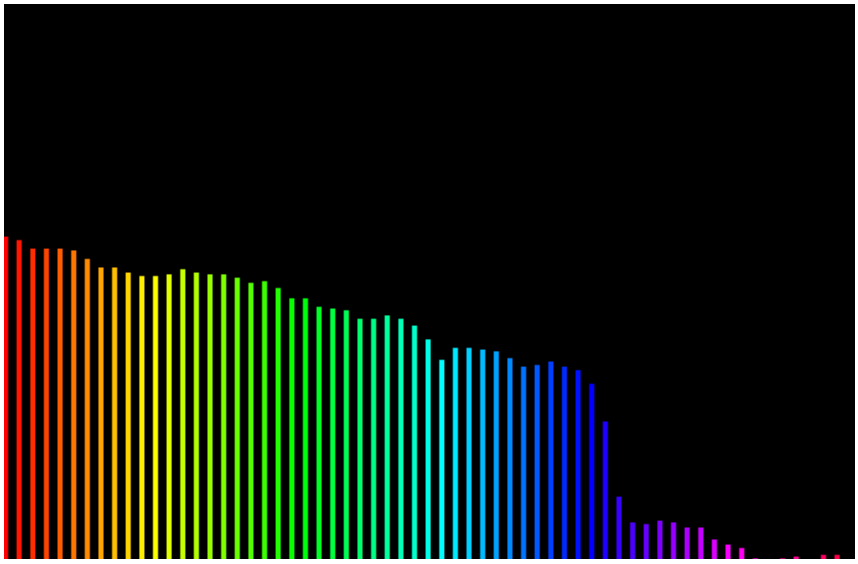

Рисунок 4 – результат работы функции.

В качестве развития этого примера, можно развернуть столбцы частот радиально вокруг определённой точки. Этого можно добиться задавая координаты начала линии функцией moveTo(), а конца – функцией lineTo(). При этом, в качестве параметров последней используются всё те же значения частот. (см. приложение 1.4)

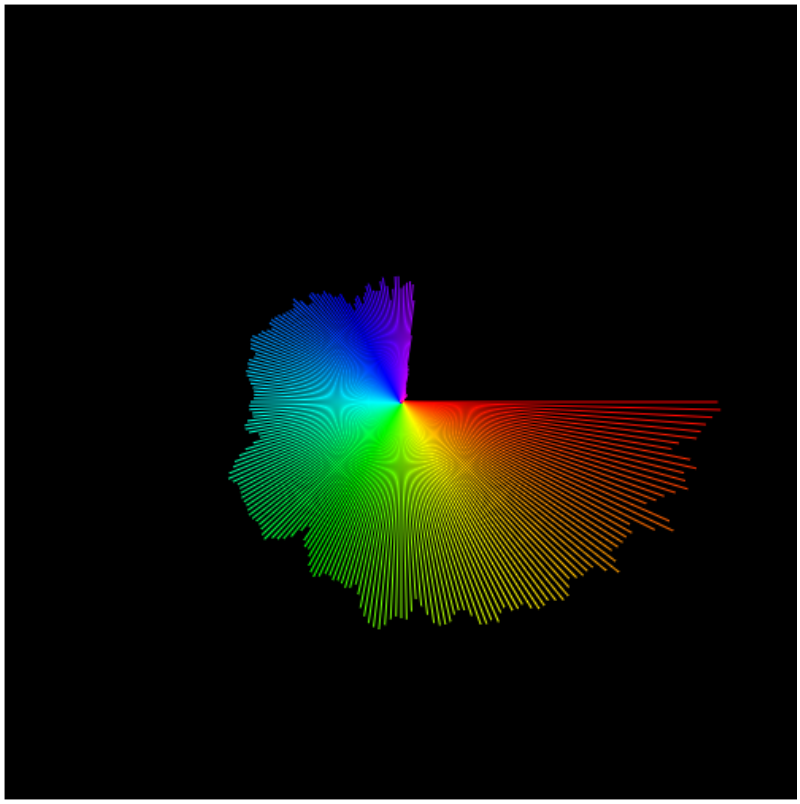

Рисунок 5 – пример визуализации.

Помимо рисования четырёхугольников, Canvas предоставляет широкие возможности для отображения графических примитивов. В данном примере используются круги, радиус которых зависит от полученной в ходе анализа частоты. (см. приложение 1.5)

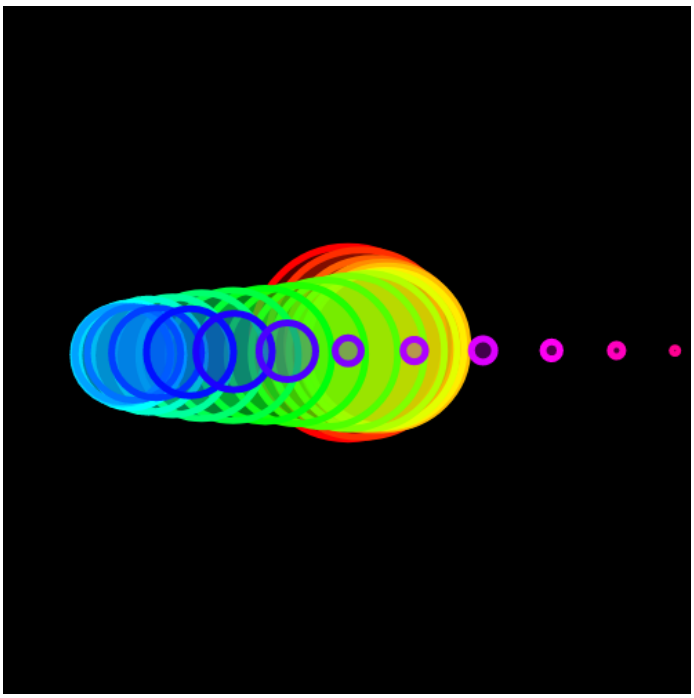

Рисунок 6 – пример визуализации.

Использование кривых Безье может привести к ещё более эффектным результатам. Задавая изменённые значения частот в качестве точек управления кривыми, можно добиться следующих результатов. (см. приложение 1.6)

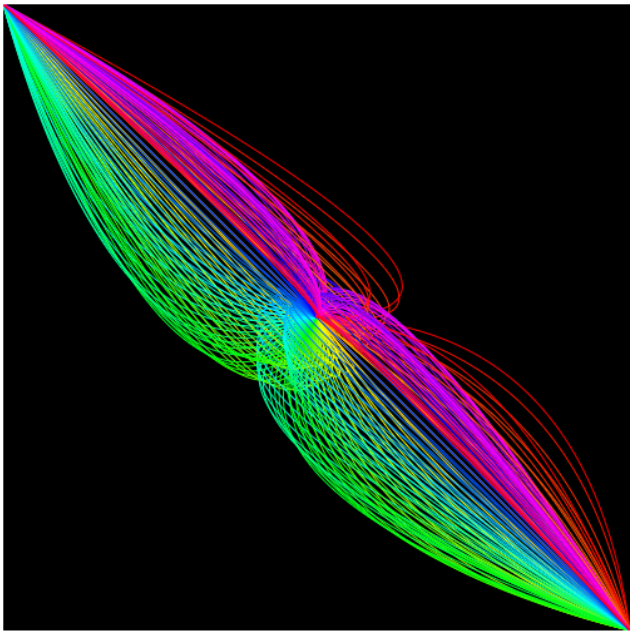

Рисунок 7 – пример визуализации

Ещё одна возможность, доступная в Canvas – использование в качестве фона для области рисования заранее нарисованных изображений. Учитывая, что по умолчанию фон прозрачен, можно достигнуть сложных визуальных композиций, «собранных» из нескольких слоёв. (см. приложение 1.7)

Рисунок 8 – пример визуализации.

Это лишь небольшая часть возможных вариантов визуализации музыки, которые можно можно получить, использую технологию рисования растровой графики Canvas.

ОПИСАНИЕ ПРОЦЕСА РАБОТЫ ПРОГРАММЫ-ВИЗУАЛИЗАТОРА

Данная глава посвящена описанию устройства и работы главной части веб-ресурса – визуализатора.

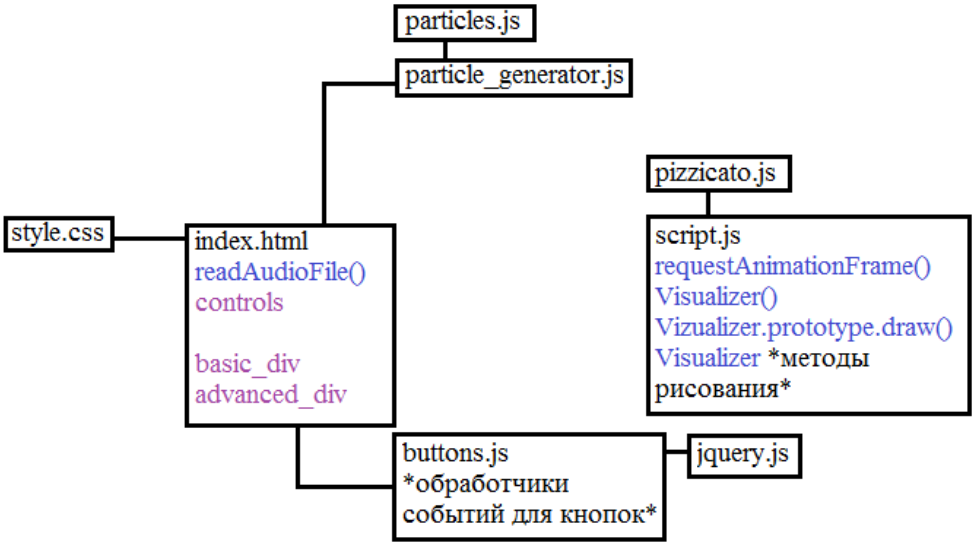

Рисунок 9 – схема устройства программы-визуализатора

На данной схеме представлена структура программы, отвечающей за визуализацию звука.

Внешнее содержимое браузера, которое будет видеть пользователь, определяется файлами index.html и style.css Первый описывает контент на странице с помощью языка разметки HTML, и является центральным узлом программы, в котором собираются воедино все данные, полученные в ходе работы других частей всей системы, а второй – определяет внешний вид отдельных элементов страницы с помощью языка описания внешнего вида документа CSS.

Условно, index.html можно разделить на несколько частей.

В начале файла находится часть программного кода, отвечающего за обработку событий загрузки. Эта часть делает возможным загрузку любого имеющегося у пользования аудиофайла для последующей визуализации. Также, этот фрагмент отвечает за вывод ошибок при загрузке, если таковые возникнут. (см. приложение 2.1)

Далее следует несколько элементов div, каждый из которых отвечает за отображение своей части интерфейса программы. Блок control отвечает за элементы управления различными аспектами работы визуализатора. Сюда относятся кнопки загрузки аудиофайлов и управления воспроизведением, ползунки настройки параметров визуализации и группа радио-кнопок, отвечающая за выбор варианта визуализации. Изначально кнопка Play, которая запускает проигрывание музыкальной композиции, заблокирована. Она становится активной по окончании загрузки файла на веб-ресурс. (см. приложение 2.2)

Непосредственно за отображение разных режимов работы визуализатора отвечают блоки basic_div и advanced_div. Они включают в себя элементы canvas, которые используются для создания на них графического изображения.

Важно отметить, что сам по себе canvas является только контейнером для графики, для непосредственно для рисования он должен использоваться совместно с инструкциями на языке JavaScript. Так же в эти блоки входят элементы checkbox, с помощью которых возможен выбор одного или нескольких алгоритмов визуализации. (см. приложение 2.3 – 2.4)

В конце index.html так же расположены теги script, подключающие к нему остальные части программы. Они включают в себя как и дополнительные части программного кода, вынесенные для лучшего структурирования в отдельные файлы, так и сторонние программные библиотеки. (см. приложение 2.5)

Среди использованных программных библиотек:

- Pizzicato.js – упомянутая раннее библиотека, оптимизирует и расширяет возможности Web Audio API.
- jquery-3.1.1.min.js – широко используемый фреймворк, чьей основной задачей является улучшить взаимодействие языков HTML и JavaScript. С помощью этой библиотеки можно легко получать доступ к элементам DOM, манипулировать ими, изменять и задавать им новые свойтсва.
- particles.js – программная библиотека, упрощающая процесс визуализации в аспекте создания и манипулирования системой частиц.

Далее в схеме устройства визуализатора следуют файлы script.js и buttons.js В них находятся фрагменты кода JavaScript выполняющему главные функции по воспроизведению, анализу и визуализации звука.

В buttons.js вынесены все операции, относящиеся к обработке событий с кнопок и ползунков, описанных в index.html Эта часть программы в полной мере использует возможности программной библиотеки jquery, реализуя их для придания интерактивности всем элементам управления визуализатора и обработки событий, возникающих в процессе манипуляций над ними. Информация о происходящих изменениях используется в частях программы, описанной в файле script.js (см. приложение 2.6 – 2.7)

Файл script.js содержит в себе весь программный код, касающийся воспроизведения аудиофайла, анализа звукового потока и его последующей визуализации. Открывает эту часть программы вызов функции requestAnimationFrame. Она представляет собой встроенный API, предназначенный для оптимизации любых видов анимации в браузере. До появления requestAnimationFrame, разработчику, если он пытался добавить элементы анимации на свой веб-ресурс, приходилось пользоваться функцией setInterval, которая автоматически вызывается каждые несколько миллисекунд, и выполняет заданное ей действие. Такой вид анимации создавал множество проблем, таких как необходимость дополнительных усилий для синхронизации анимации нескольких элементов и невозможность подстроится под уровень производительности конкретного компьютера. Использование requestAnimationFrame избавляет разработчика от всех этих проблем. Кардинальное отличие этой функции от setInterval в том, что она вызывается не через заранее определённые отрезки времени. Вместо этого, браузер старается исполнять её как только него появляется такая возможность. Таким образом, в зависимости от мощности аппаратной части и её загруженности requestAnimationFrame может автоматически выбирать необходимое количество кадров анимации в секунду. Таким образом, использование этого метода предоставляет лучшую синхронизацию с графической картой и более оптимальное использование ресурсов центрального процессора. Недостатком данного API является то, что из-за его относительной новизны функция его вызова различается у разных браузеров. (см. приложение 2.8)

За вызовом requestAnimationFrame следует описание главного класса программы – Visualizer. На его откуп ложится всё, связанное с непосредственным проигрыванием и визуализацией аудиофайлов.

Первым делом в конструкторе класса Visualizer происходит создание аудио-контекста Web Audio API и добавление в него загруженного ранее пользователем аудиофайла. В этом месте программы наглядно видно влияние Pizzicato.js – указанные выше действия проделываются с использованием

синтаксиса этой библиотеки. Вместо создания аудио-контекста и загрузки буфера с аудиофайлом вручную, в Pizzicato создаётся объект класса Sound, в котором указывается источник аудиоданных (в этом случае – файл) и путь до него. (см. приложение 2.9)

После создания объекта класса Sound, происходит вызов аудио-контекста для создания узла-анализатора Analyser Node. Для этого вновь используются возможности библиотеки Pizzicato.js Помимо создания непосредственно узла в конструкторе задаются его основные параметры – минимальное и максимальное значение децибел, массивы для хранения временных и частотных характеристик полученного сигнала. (см. приложение 2.10)

Функция draw класса Visualizer выполняет все действия, связанные с отрисовкой и анимацией аудиофайла, используя для этого его частотные характеристики.

В начале функции указываются определяются значения размерности быстрого преобразования Фурье и уровень «сглаживания частот». Изменение первой характеристики влияет на размер массива с частотами, использующийся для визуализации, что, в зависимости от характера этих изменений может как сделать получаемую визуализацию более зрелищной и требовательной к характеристикам процессора и графической карты, так и уменьшить количество отображаемых объектов, что положительно повлияет на производительность.

```
this.analyser.smoothingTimeConstant = stc;
this.analyser.fftSize = Math.pow(2, fft);
```

После определения значений этих параметров происходит обновление содержимого массивов частот и временных характеристик. Так как функция draw привязана к requestAnimationFrame обновление содержимого массива происходит при отрисовке каждого нового кадра.

```
this.analyser.getByteFrequencyData(this.freqs);
this.analyser.getByteTimeDomainData(this.times);
```

В зависимости от выбранного пользователем вида визуализации, функция draw будет работать по разным сценариям. Если выбрана опция basic, то весь будет произведён перебор всех значений массива частот и каждой из них будет своим образом использована в визуализации, в зависимости от конкретной функции отрисовки. (см. приложение 2.11)

При использовании режима advanced массив частот предварительно разбивается на части, и в дальнейшем визуализация проходит по другим алгоритмам. (см. приложение 2.12)

В конце функции draw происходит её привязка к requestAnimationFrame.

```
if (this.isPlaying) {
requestAnimFrame(this.draw.bind(this));}
```

В конце описания класса Visualizer описаны различные функции, отвечающие за конкретные алгоритмы визуализации. Их выбор настраивается пользователем и разница в зависимости от выбранного режима – basic или advanced.

Последним рассмотренным классом является в Generator, вынесенный в отдельный файл particle_generator.js Он содержит в себе все настройки, касающиеся применения системы частиц из библиотеки particles.js, которая используется в режиме визуализатора advanced. Эти настройки включают в себя количество отображаемых частиц, их цвет, размер и форму, скорость и направление их движения, и, самое главное – реакцию на различные события, генерируемые визуализатором. (см. приложение 2.13)

Итого, алгоритм работы программы-визуализатора можно сформулировать следующим образом:

- Пользователь выбирает на своём компьютере аудиофайл и загружает его на веб-ресурс с помощью элемента управления типа file.

- Это событие обрабатывается функцией readAudioFile, которая, закончив загрузку файла, передаёт его конструктору класса Visualizer.
- Конструктор создаёт новый объект класса Sound, используя как источник загруженный аудиофайл.
- После окончания загрузки активируется кнопка Play
- Пользователь выбирает вариант визуализации и отмечает в чекбоксах её конкретные алгоритмы.
- После нажатия кнопки Play объект класса Sound начинает воспроизведение аудиофайла. Функция draw начинает получать информацию из массивов частот и временных характеристик. Происходит визуализация в реальном времени.
- По окончании музыкального произведения или при постановке на паузу процесс визуализации останавливается.

Таким образом пользователь получает доступ к ключевому элементу веб-ресурса – визуализатору музыкальных произведений, работающему напрямую в браузере.

Описанная выше схема работы привод в действие следующие алгоритмы визуализации:

- BasicSpectre() – изображает спектр частот аудио сигнала, который отзеркален относительно центра. (см. Приложение 3.1)
- RadialSpectre() – представляет собой эквалайзер, столбцы которого расположены радиально относительно центральной точки объекта Canvas. Дополнительно, положене столбцов изменяется динамически основываясь на показаниях внутреннего счётчика времени, прошедшего с начала воспроизведения аудиофайла. (см. Приложение 3.2)

- Circle() – частотные характеристики представляются в виде окружностей, чей радиус изменяется в зависимости от из величины. (см. Приложение 3.3)
- Bezier() – визуализация, основанная на изображении многих кривых Безье. Параметры кривых зависят от выходных данных анализа частот. (см. Приложение 3.4)
- BezierRotation() – основывается на функции Bezier(), добавляет анимацию вращения полученной фигуры. (см. Приложение 3.5)
- AdvSpectre1-3 – алгоритмы блока advanced, сочетают в себе визуализацию спектра частот аудиофайла и использование системы частиц. (см. Приложение 3.6 – 3.8)

Поскольку рисование растровой графики является ресурсоёмким процессом, были введены следующие меры для оптимизации процесса:

- Избегание лишних изменений состояния объекта Canvas. Это включает в себя, например, изменение процесса отрисовки графических примитивов – изображение всех объектов одного цвета последовательно, а не в произвольном порядке, изменяя цвет для каждого графического элемента.
- Использование готовых изображений в качестве фона.
- Использование функции fillRect() вместо переустановки размеров Canvas для очищения экрана перед новой отрисовкой.

ВЫБОР И НАСТРОЙКА ПЛАТФОРМЫ ДЛЯ РАЗМЕЩЕНИЯ ВЕБ-РЕСУРСА

Параллельно с работой над созданием программы-визуализатора, был проведён обзор доступных вариантов хостинга для будущего веб-ресурса. В качестве

итогового для размещения была выбрана площадка hostland.ru Выбор был обусловлен следующими факторами:

- Возможность опробовать хостинг в тестовом режиме в течении 30-и дней.
- Возможность размещения веб-сайта на техническом домене, не требующем регистрации для первоначального тестирования.
- Приемлемая стоимость абонентской платы за пользование услугами хостинга и возможность бесплатно зарегистрировать доменное имя при оплате подписки на год.
- Достаточное количество выделенного места на сервере.
- Возможность автоматической установки приложений на веб-сайт.

По окончании работы над программной частью веб-ресурса, посвящённого визуализации звука средствами браузера, готовая программа, а так же все сопутствующие текстовые материалы были размещены на данном хостинге по адресу retroviz.ru

Для оптимизации процесса заполнения и обновления веб-ресурса на сервере была установлена CMS Drupal. Данная CMS является свободно распространяемой и имеет большое количество готовых программных модулей, позволяющих подстроить внешний вид и функционал сайта под любые необходимые цели. Следующие модули были выбраны для установки на веб-сайт и доказали свою полезность в деле создания специализированного ресурса:

- Administration menu и Overlay – улучшают функционал и внешний вид инструментов администрирования. Добавляют панель управления сайтом с выпадающими меню, позволяющую получить быстрый доступ ко всем возможностям администратора.

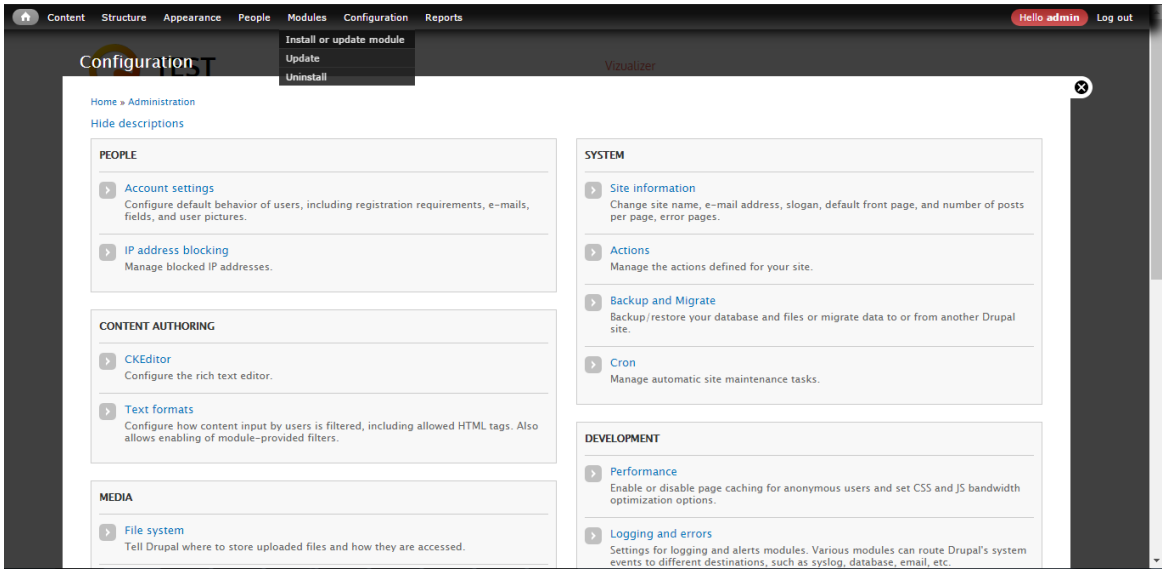

Рисунок 10 – меню администрирования.

- Update Manager – автоматически следит за обновлениями модулей, упрощает установку новых моделей.
- CKEditor – текстовый редактор с широким списком возможностей. Позволяет создавать и редактировать HTML-код страницы прямо в браузере, а так же задавать пользовательские стили для текста.
- JQuery Update – автоматически добавляет поддержку JQuery всем страницам на сайте и следит за обновлениями этой программной библиотеки.
- Backup and Migrate – автоматически создаёт бэкапы базы данных веб-ресурса и позволяет упростить перенос данных на другой домен, если такая необходимость вознкнет.
- Path – позволяет устанавливать собственные URL для страниц на ресурсе.

В качестве темы для оформления были рассмотрены два варианта:

- Zen – тема, обладающая огромными возможностями для кастомизации внешнего вида сайта. Однако, не включает в себя никакого базового дизайна, заставляя создавать его целиком с нуля.

- Nexus – тема, включающая в себя базовые элементы дизайна и внешнего вида сайта, такого как слайдеры изображений и выпадающие списки. Так же, включает в себя заданные по умолчанию стили CSS. Было решено остановиться именно на этом варианте.

После установки всех необходимых модулей и настройки веб-страницы с визуализатором, были подготовлены и загружены обучающие материалы, содержащие в себе типичные, использованные для создания данного программного продукта.

Получившийся веб-ресурс являет собой пример использования современных методов разработки и дизайна страниц в Интернете и предоставляет возможности для своего улучшения и расширения.

ПЕРСПЕКТИВЫ РАЗВИТИЯ

Данный веб-ресурс может в дельнейшем развиваться в нескольких областях.

Во-первых, поскольку поддержка и написание новые спецификаций стандарта Web Audio продолжается, будет продолжать увеличиваться и актуализироваться раздел сайта, отвечающий за предоставление обучающих материалов по данной теме.

Во-вторых, будет проводиться оптимизация старых и разработка новых методов визуализации аудиофайлов.

ЗАКЛЮЧЕНИЕ

По итогам проделанной работы, были выполнены все поставленные в начале выпускной квалификационной работы задачи, а именно:

- Изучены программные стандарты, требующиеся для создания веб-ресурса с аудио-визуальной инсталляцией.
- Проанализированы программные библиотеки, призванные упростить процесс разработки.
- Проведён краткий обзор истории визуализации звука.
- Создан и заполнен обучающими материалами веб-ресурс, посвящённый визуализации звука средствами браузера.
- Проведена оптимизация алгоритмов визуализации.

Результаты теоретических изысканий были успешно применены для создания практического проекта.

СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

1. Гоше, Х. Д. HTML5 для профессионалов / Х. Д. Гоше . – 2-е изд. – СПб.: Питер, 2015. – 560 с.: ил. – (Серия «Для профессионалов»).

2. Smus, Boris Web Audio API: Advanced Sound for Games and Interactive Apps / Boris Smus. - O'Reilly Media, 2013. – 76 с.: ил.

3. Web Audio API W3C Working Draft 08 December 2015 [Электронный ресурс] / www.w3.org – URL: https://www.w3.org/TR/webaudio/ (дата обращения 04.05.2017).

4. Getting Started with Web Audio API [Электронный ресурс] / www.html5rocks.com – URL: https://www.html5rocks.com/en/tutorials/webaudio/intro/ (дата обращения 04.05.2017).

5. Web Audio API [Электронный ресурс] / developer.mozilla.org – URL: https://developer.mozilla.org/ru/docs/Web/API/Web_Audio_API (дата обращения 04.05.2017).

6. Web Audio API – новые возможности генерации, обработки и объемного распределения звука в браузере. [Электронный ресурс] / html5.by – URL: http://html5.by/blog/audio/ (дата обращения 04.05.2017).

7. Exploring the HTML5 Web Audio: visualizing sound [Электронный ресурс] / www.smartjava.org – URL: http://www.smartjava.org/content/exploring-html5-web-audio-visualizing-sound (дата обращения 04.05.2017).

8. Start using Web Audio with Pizzicato.js [Электронный ресурс] / https://codepen.io – URL: https://codepen.io/gregh/post/start-using-the-web-audio-with-pizzicato-js (дата обращения 04.05.2017).

9. Particles.js: Introduction [Электронный ресурс] / https://code.tutsplus.com – URL: https://code.tutsplus.com/tutorials/particles-js-introduction--cms-26285 (дата обращения 04.05.2017).

10. Reading files in JavaScript using File APIs [Электронный ресурс] / https://www.html5rocks.com – URL: https://www.html5rocks.com/en/tutorials/file/dndfiles/ (дата обращения 04.05.2017).

11. Improving HTML5 Canvas Performance [Электронный ресурс] / https://www.html5rocks.com – URL: https://www.html5rocks.com/en/tutorials/canvas/performance/ (дата обращения 04.05.2017).

12. Хромосемантика музыки [Электронный ресурс] / http://www.ka2.ru – URL: http://www.ka2.ru/nauka/chenikov_3.html (дата обращения 04.05.2017).

ПРИЛОЖЕНИЕ

1.1

```
var context = new window.AudioContext();
var buffer, source;

var loadSoundFile = function(url) {
  var request = new XMLHttpRequest();
  request.open('GET', url, true);
  request.responseType = 'arraybuffer';
  request.onload = function(e) {
    context.decodeAudioData(this.response,
    function(decodedArrayBuffer) {
      buffer = decodedArrayBuffer;
       play();
    }, function(e) {
      console.log('Error decoding file', e);
    });
  };
  request.send();
  console.log('loaded');

function play(){
  source = context.createBufferSource();
  source.buffer = buffer;
  source.connect(context.destination);
  source.start(0);
}

loadSoundFile('r_e_m_losing_my_religion.mp3');
```

1.2

```
var sound = new Pizzicato.Sound({
    source: 'file',
    options: { path: 'r_e_m_losing_my_religion.mp3' }
}, function() {});

sound.play();
```

1.3

```
function eq(i) {
     var hue = i/this.analyser.frequencyBinCount * 360;
     drawContext.fillStyle = 'hsl(' + hue + ', 100%, 50%)';
     drawContext.fillRect(i * 8, 500, 3, - this.freqs[i]);
     drawContext.stroke();
}
```

1.4

```
function eqRad(centerPoint, angle, lineLength, i, drawContext) {
     drawContext.beginPath();
     drawContext.moveTo(centerPoint[0], centerPoint[1]);
     drawContext.lineTo(centerPoint[0] + lineLength *
Math.cos(angle), centerPoint[1] + lineLength * Math.sin(angle));
     var hue = i/this.analyser.frequencyBinCount * 360;
     drawContext.strokeStyle = 'hsl(' + hue + ', 100%, 50%)';
     drawContext.stroke();
}
```

1.5

```
function circles(centerPoint, angle, angle3, lineLength, i,
drawContext, circleX, circleY, circleInc1, circleInc2) {
     drawContext.beginPath();
```

```
    drawContext.arc(centerPoint[0] + Math.cos(angle) * angle * 50
, centerPoint[1] - Math.cos(angle3) * angle3, lineLength / 3,
Math.PI * 2, false);

    drawContext.lineWidth = 5;

    var hue = i/this.analyser.frequencyBinCount * 360;

    drawContext.strokeStyle = 'hsl(' + hue + ', 100%, 50%)';

    drawContext.fillStyle = 'hsla(' + hue + ', 100%, 50%, 0.3)';

    drawContext.fill();

    drawContext.stroke();

}
```

1.6

```
function bezier(centerPoint, angle, lineLength, i, drawContext) {

    drawContext.beginPath();

    drawContext.moveTo(centerPoint[0], centerPoint[1]);

    drawContext.quadraticCurveTo(centerPoint[0] + lineLength *
Math.cos(angle), centerPoint[1] + lineLength * Math.sin(angle), 0,
0);

    drawContext.moveTo(centerPoint[0], centerPoint[1]);

    drawContext.quadraticCurveTo(centerPoint[0] + lineLength *
Math.cos(angle), centerPoint[1] + lineLength * Math.sin(angle),
500, 500);

    drawContext.lineWidth = 1;

    var hue = i/this.analyser.frequencyBinCount * 360;

    drawContext.strokeStyle = 'hsl(' + hue + ', 100%, 50%)';

    console.log(hue);

    drawContext.stroke();

}
```

1.7

```
Function advEq(drawContext, freqs, fbc) {

    var begin = 0,

    size = fbc / 8,
```

```
     end = begin + size;
     x = 16,
     y = 576,
     width = 63;
     for (var i = 1; i <= 8; i++) {
          var chunk = freqs.slice(begin, end),
               height = 0;
          for (var j = 0; j < size; j++) {
               height += chunk[j];
          }
          height = height / size;
          drawContext.fillStyle = 'hsla(' + height + ', 100%, 50%,
0.4)';
          drawContext.fillRect(x, y, width, -height * 2);
          drawContext.fillRect(1280 - x - 63, y, width, -height *
2);
          drawContext.stroke();
          begin += size;
          end += size;
          x += width + 16;
     }
}
```

2.1

```
<script>
     function readAudioFile() {
          var reader = new FileReader();

reader.readAsDataURL(document.getElementById('audioReader').files[
0]);
```

```
          reader.onload = function(event) {
               vis = new Visualizer(event.target.result);
               var g = new Generator();
          };
          reader.onerror = function(event) {
               console.error("Ошибка... " +
event.target.error.code);
          };
     };
</script>
```

2.2

```
<div id="controls">
     <p>
          <input type="file" id="audioReader"
onchange="readAudioFile()"></input>
     </p>
     <p>
          <input type="button" id="playButton" value="Play"
disabled="true">
     </p>
     <p>
          Режим отображения:<Br>
          <input type="radio" name="mode" id="basic_radio" checked>
Basic
          <input type="radio" name="mode" id="advanced_radio">
Advanced
     </p>
     <p>
          <div>FFT Size(5-15) <input type="range" id="fftSize"
min="5" max="15" step="1" value="6" /> <input type="number"
id="fftValue" value="6" readonly></div>
```

```
          <div>Smoothing Time Constant(0.0-1.0) <input type="range"
id="stc" min="0" max="1" value="0.8" step="0.1"/> <input
type="number" id="stcValue" value="0.8" readonly></div>

     </p>

</div>
```

2.3

```
<div id="basic_div" >

     <form>

          <input type="checkbox" id="func1"> drawLine <br>

          <input type="checkbox" id="func2"> drawBezier <br>

          <input type="checkbox" id="func3"> drawBezierTest <br>

          <input type="checkbox" id="func4"> drawBezierRotate<br>

          <input type="checkbox" id="func5"> drawBezierRotate1<br>

          <input type="checkbox" id="func6" > circles<br>

     </form>

     <canvas id="basic_canvas" width="500" height="500"
style="background-color: black;"></canvas>

</div>
```

2.4

```
<div id="advanced_div" hidden>

     <form>

          <input type="checkbox" id="func7" > equalizer1<br>

          <input type="checkbox" id="func8"> equalizer2<br>

          <input type="checkbox" id="func9"> equalizer3<br>

     </form>

     <div id="particles-js">

          <canvas id="canvas" width="1280" height="720"
style="position: absolute; z-index: 0"></canvas>

          <div id="fg"></div>

          <div id="p1"></div>
```

```
          <div id="p2"></div>
          <div id="p3"></div>
     </div>
</div>
```

2.5

```
<script src="Pizzicato.js"></script>
<script src="jquery-3.1.1.min.js"></script>
<script src="script.js"></script>
<script src="buttons.js"></script>
<script src="particles.js"></script>
<script src="particle_generator.js"></script>
```

2.6

```
$("#playButton").click(function(){
     if(vis.isPlaying == false) {
          vis.isPlaying = true;
          vis.sound.play();
          requestAnimFrame(vis.draw.bind(vis));
          $("#playButton").prop('value', 'Pause');
     } else {
          vis.isPlaying = false;
          vis.sound.pause();
          $("#playButton").prop('value', 'Play');
     }
});
$('#basic_radio').click(function(){
     basic_mode = true;
     $('#advanced_div').hide();
     $('#basic_div').show();
});
```

```
$('#advanced_radio').click(function(){
     basic_mode = false;
     $('#basic_div').hide();
     $('#advanced_div').show();
});
```

2.7

```
$('#func1').click(function() {if($('#func1').prop('checked')) {
func1 = true;
} else {
     func1 = false;
}});
$('#func2').click(function() {if($('#func2').prop('checked')) {
     func2 = true;
} else {
     func2 = false;
     }});
$('#func3').click(function() {if($('#func3').prop('checked')) {
     func3 = true;
} else {
     func3 = false;
}});
```

2.8

```
window.requestAnimFrame = (function(){
return  window.requestAnimationFrame ||
  window.webkitRequestAnimationFrame ||
  window.mozRequestAnimationFrame    ||
  window.oRequestAnimationFrame      ||
  window.msRequestAnimationFrame     ||
  function( callback ){
```

```
  window.setTimeout(callback, 1000 / 60);
};
})();
```

2.9

```
function Visualizer(source) {
    this.sound = new Pizzicato.Sound({
    source: 'file',
     loop: 'true',
    options: { path: source }
}, function() {

document.getElementById("playButton").removeAttribute("disabled");
});
```

2.10

```
this.analyser = Pizzicato.context.createAnalyser();
this.sound.connect(this.analyser);
this.analyser.minDecibels = -140;
this.analyser.maxDecibels = 0;
this.freqs = new Uint8Array(this.analyser.frequencyBinCount);
this.times = new Uint8Array(this.analyser.frequencyBinCount);
```

2.11

```
if (basic_mode) {
     var canvas = $("#basic_canvas")[0];
     var drawContext = canvas.getContext('2d');
     drawContext.clearRect(0, 0, 500, 500);
  for (var i = 0; i < this.analyser.fftSize; i++) {
```

```
    var b = this.analyser.frequencyBinCount - 1 - i;

    var times = this.times[i];

    var value = this.freqs[i];

    SPEED = this.freqs[value];

    var percent = value / 256;

    var percentTimes = times/256;

    var lineLength = maxLength * percent;

    var lineLengthTimes = maxLength * percentTimes;

    var angle = i/this.analyser.frequencyBinCount * 2 * Math.PI;

    var angle1 = i/this.analyser.frequencyBinCount * 2 * Math.PI
* seconds;

    var angle3 = b/this.analyser.frequencyBinCount * 2 * Math.PI;

    if (func1) {

         this.drawLine(centerPoint, angle, lineLength, i,
drawContext);

   }

    if (func2) {

         this.drawBezier(centerPoint, angle, lineLength, i,
drawContext);

    }

    if (func3) {

         this.drawBezierTest(centerPoint, angle, lineLength,
times, i, drawContext);

    }

    if (func4) {

         this.drawBezierRotate(centerPoint, angle, angle3,
lineLength, i, drawContext, lineLengthTimes);

    }

    if (func5) {

         this.drawBezierRotate1(centerPoint, angle, angle3,
lineLength, i, drawContext, this.circleX, this.circleY,
this.circleInc1, this.circleInc2);

         console.log(this.freqs[i]);
```

```
     }

     if (func6) {

          this.circles(centerPoint, angle, angle3, lineLength, i,
drawContext, this.circleX, this.circleY, this.circleInc1,
this.circleInc2);

     }

};}
```

2.12

```
if (!basic_mode){

     var canvas = $("#canvas")[0];

     var drawContext = canvas.getContext('2d');

     drawContext.clearRect(0, 0, 1280, 720);

     if (func7) {

          this.equalizer1(drawContext, this.freqs,
this.analyser.frequencyBinCount);

     }

     if (func8) {

          this.equalizer2(drawContext, this.freqs,
this.analyser.frequencyBinCount);

     }

     if (func9) {

          this.equalizer3(drawContext, this.freqs,
this.analyser.frequencyBinCount);

     }

     if (((maxI - minI) > 185) && (bubble == 15)) {

          $(".particles-js-canvas-el").click();

          bubble = 0;

     } else if ((maxI - minI) > 185){

          bubble+=1;

     }

}
```

2.13

```
function Generator() {
 particlesJS('p1',
  {
    "particles": {
      "number": {
        "value": 30,
        "density": {
          "enable": true,
          "value_area": 800
        }
      },
      "color": {
        "value": "#ff00ff"
      },
      "shape": {
        "type": "circle",
        "stroke": {
          "width": 0,
          "color": "#f00000"
        },
        "polygon": {
          "nb_sides": 5
        },

      },
      "opacity": {
        "value": 0.5,
        "random": true,
        "anim": {
```

```
          "enable": true,
          "speed": 1,
          "opacity_min": 0.1,
          "sync": false
}
```

3.1

```
Visualizer.prototype.basicSpectre = function (i, drawContext) {
     var hue = i/this.analyser.frequencyBinCount * 360;
     drawContext.fillStyle = 'hsl(' + hue + ', 100%, 50%)';
     drawContext.fillRect(i * 4 + 252, 500, 1, - this.freqs[i]);
     drawContext.fillRect(248 - i * 4, 500, 1, - this.freqs[i]);
     drawContext.stroke();
}
```

3.2

```
Visualizer.prototype.radialSpectre = function(centerPoint,
angleTime, lineLength, i, drawContext) {
     drawContext.beginPath();
     drawContext.moveTo(centerPoint[0], centerPoint[1]);
     drawContext.lineTo(centerPoint[0] + lineLength *
Math.cos(angleTime), centerPoint[1] + lineLength *
Math.sin(angleTime));
     var hue = i/this.analyser.frequencyBinCount * 360;
     drawContext.strokeStyle = 'hsl(' + hue + ', 100%, 50%)';
     drawContext.stroke();
}
```

3.3

```
Visualizer.prototype.circles = function(centerPoint, angle,
angle3, lineLength, i, drawContext, circleX, circleY, circleInc1,
circleInc2) {
     if (i%3 == 0) {
     drawContext.beginPath();
```

```
     drawContext.arc(centerPoint[0] + Math.cos(angle) * angle * 50
, centerPoint[1] - Math.cos(angle3) * angle3, lineLength / 3,
Math.PI * 2, false);

     drawContext.lineWidth = 5;

     var hue = i/this.analyser.frequencyBinCount * 360;

     drawContext.strokeStyle = 'hsl(' + hue + ', 100%, 50%)';

     drawContext.fillStyle = 'hsla(' + hue + ', 100%, 50%, 0.3)';

     drawContext.fill();

     drawContext.stroke();

     }

}
```

3.4

```
Visualizer.prototype.bezier = function(centerPoint, angle,
lineLength, i, drawContext) {

     drawContext.beginPath();

     drawContext.moveTo(centerPoint[0], centerPoint[1]);

     drawContext.quadraticCurveTo(centerPoint[0] + lineLength *
Math.cos(angle), centerPoint[1] + lineLength * Math.sin(angle), 0,
0);

     drawContext.moveTo(centerPoint[0], centerPoint[1]);

     drawContext.quadraticCurveTo(centerPoint[0] + lineLength *
Math.cos(angle), centerPoint[1] + lineLength * Math.sin(angle),
500, 500);

     var hue = i/this.analyser.frequencyBinCount * 360;

     drawContext.strokeStyle = 'hsl(' + hue + ', 100%, 50%)';

     drawContext.stroke();

}
```

3.5

```
Visualizer.prototype.bezierRotation = function(centerPoint, angle,
angle3, lineLength, i, drawContext, circleX, circleY, circleInc1,
circleInc2) {

     drawContext.beginPath();

     drawContext.moveTo(centerPoint[0], centerPoint[1]);
```

```
     drawContext.quadraticCurveTo(centerPoint[0] + lineLength *
Math.cos(angle), centerPoint[1] + lineLength * Math.sin(angle),
circleX[circleInc1], circleY[circleInc1]);

     drawContext.moveTo(centerPoint[0], centerPoint[1]);

     drawContext.quadraticCurveTo(centerPoint[0] + lineLength *
Math.cos(angle3), centerPoint[1] + lineLength * Math.sin(angle3),
circleX[circleInc2], circleY[circleInc2]);

     drawContext.lineWidth = 1;

     var hue = i/this.analyser.frequencyBinCount * 360;

     drawContext.strokeStyle = 'hsl(' + hue + ', 100%, 50%)';

     drawContext.stroke();

}
```

3.6

```
Visualizer.prototype.advSpectre1 = function(drawContext, freqs,
fbc) {

     var begin = 0,

          size = fbc / 8,

          end = begin + size;

          x = 24,

          y = 576,

          width = 133;


     for (var i = 1; i <= 8; i++) {

          var chunk = freqs.slice(begin, end),

               height = 0;

          for (var j = 0; j < size; j++) {

               height += chunk[j];

          }

          var hue = i / this.analyser.frequencyBinCount * 360;

          height = height / size;

          drawContext.fillStyle = 'hsla(' + hue + ', 100%, 50%,
0.5)';
```

```
            drawContext.fillRect(x, y, width, -height * 2);
            drawContext.stroke();
            begin += size;
            end += size;
            x += width + 24;
        }
}
```

3.7

```
Visualizer.prototype.advSpectre2 = function(drawContext, freqs,
fbc) {
    var begin = 0,
            size = fbc / 8,
            end = begin + size;
            x = 16,
            y = 576,
            width = 63;

    for (var i = 1; i <= 8; i++) {
            var chunk = freqs.slice(begin, end),
                height = 0;
            for (var j = 0; j < size; j++) {
                height += chunk[j];
            }
            height = height / size;
            drawContext.fillStyle = 'hsla(' + height + ', 100%, 50%,
0.4)';
            drawContext.fillRect(x, y, width, -height * 2);
            drawContext.fillRect(1280 - x - 63, y, width, -height *
2);
            drawContext.stroke();
            begin += size;
```

```
          end += size;
          x += width + 16;
     }
}
```

3.8

```
Visualizer.prototype.advSpectre3 = function(drawContext, freqs,
fbc) {
     var begin = 0,
          size = fbc / 16,
          end = begin + size;
          x = 16,
          y = 576,
          width = 63;

     for (var i = 1; i <= 16; i++) {
          var chunk = freqs.slice(begin, end),
               height = 0;
          for (var j = 0; j < size; j++) {
               height += chunk[j];
          }
          height = height / size;
          drawContext.fillStyle = 'hsla(' + height + ', 100%, 50%,
0.4)';
          drawContext.fillRect(x, y, width, -height * 1.5);
          drawContext.stroke();
          begin += size;
          end += size;
          x += width + 16;
     }
}
```