**ПРАВИТЕЛЬСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ**

**НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ**

**«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»**

Факультет компьютерных наук

Образовательная программа бакалавриата «Программная инженерия»

|  |  |  |
| --- | --- | --- |
| СОГЛАСОВАНО  Руководитель проекта,  профессор департамента больших  данных и информационного поиска  ФКН, доктор физ.-мат. наук  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ А. Б. Шаповал  « 17 » 05 2021 г. |  | УТВЕРЖДАЮ  Академический руководитель образовательной программы «Программная инженерия»  профессор департамента программной инженерии, канд. техн. наук  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ В.В. Шилов  «\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_ 2021 г. |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  | | --- | --- | | ***Подп. и дата*** |  | | ***Инв. № дубл.*** |  | | ***Взам. инв. №*** |  | | ***Подп. и дата*** |  | | ***Инв. № подл*** |  | | **Аудиоплагин для создания стереозвука**  **Пояснительная записка**  **ЛИСТ УТВЕРЖДЕНИЯ**  **RU.17701729.05.02-01 81 01-1-ЛУ** | | |
|  |  | |
| Исполнитель  студент группы БПИ 196  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ /М.С. Шестаков /  « 17 » 05 2021 г. | |
|  | | |
|  | |  |

**Москва 2021**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| УТВЕРЖДЕН  RU.17701729.05.02-01 81 01-1-ЛУ |  | |  | |
| |  |  | | --- | --- | | ***Подп. и дата*** |  | | ***Инв. № дубл.*** |  | | ***Взам. инв. №*** |  | | ***Подп. и дата*** |  | | ***Инв. № подл*** |  | | **Аудиоплагин для создания стереозвука**  **Пояснительная записка**  **RU.17701729.05.02-01 81 01-1**  **Листов 27** | | | | |
|  | |  | | |
|  | | |
|  | | | | |
|  | | | |  |

**Москва 2021**

**СОДЕРЖАНИЕ**

Оглавление

[1. ВВЕДЕНИЕ 4](#_Toc72439323)

[1.1 Наименование программы 4](#_Toc72439324)

[1.2 Документы, на основании которых ведется разработка программы 4](#_Toc72439325)

[2. НАЗНАЧЕНИЕ И ОБЛАСТЬ ПРИМЕНЕНИЯ 5](#_Toc72439326)

[2.1 Назначение программы 5](#_Toc72439327)

[2.1.1 Функциональное назначение 5](#_Toc72439328)

[2.1.2 Эксплуатационное назначение 5](#_Toc72439329)

[2.2 Краткая характеристика области применения 5](#_Toc72439330)

[3. ТЕХНИЧЕСКИЕ ХАРАКТЕРИСТИКИ 6](#_Toc72439331)

[3.1 Постановка задачи на разработку программы 6](#_Toc72439332)

[3.2 Применяемые математические методы в программе 6](#_Toc72439333)

[3.2.1 Оконное преобразование Фурье 6](#_Toc72439334)

[3.2.2 Получение фазы и амплитуды 6](#_Toc72439335)

[3.2.3 Обратное преобразование 7](#_Toc72439336)

[3.3 Описание и обоснование алгоритма и функционирования программы 7](#_Toc72439337)

[3.3.1 Накопление блоков данных 7](#_Toc72439338)

[3.3.2 Применение преобразования Фурье 7](#_Toc72439339)

[3.3.3 Выделение амплитуды и фазы для каждой из частот 8](#_Toc72439340)

[3.3.4 Разделение частот на стерео 8](#_Toc72439341)

[3.3.5 Преобразование амплитуды и фазы обратно в комплексное число 9](#_Toc72439342)

[3.3.6 Обратное преобразование Фурье 9](#_Toc72439343)

[3.3.7 Применение WOLA 9](#_Toc72439344)

[3.3.8 Накопление полученных данных и возврат результата в DAW 9](#_Toc72439345)

[3.4 Описание и обоснование выбора метода организации входных и выходных данных 10](#_Toc72439346)

[3.4.1 Описание метода организации входных и выходных данных 10](#_Toc72439347)

[3.4.2 Обоснование метода организации входных и выходных данных 10](#_Toc72439348)

[3.5 Описание и обоснование выбора состава технических и программных средств 10](#_Toc72439349)

[4. ТЕХНИКО-ЭКОНОМИЧЕСКИЕ ПОКАЗАТЕЛИ 11](#_Toc72439350)

[4.1 Предполагаемая потребность 11](#_Toc72439351)

[4.2 Экономические преимущества разработки по сравнению с отечественными и зарубежными образцами или аналогами 11](#_Toc72439352)

[ПРИЛОЖЕНИЕ 1 ТЕРМИНОЛОГИЯ 12](#_Toc72439353)

[ПРИЛОЖЕНИЕ 2 ОПИСАНИЕ КЛАССОВ В ПРОГРАММЕ 13](#_Toc72439354)

[ПРИЛОЖЕНИЕ 3 СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ 26](#_Toc72439355)

[ЛИСТ РЕГИСТРАЦИИ ИЗМЕНЕНИЙ 27](#_Toc72439356)

# ВВЕДЕНИЕ

## Наименование программы

Наименование программы ̶ "Аудиоплагин для создания стереозвука"

Наименование программы на английском языке ̶ "Audioplugin for creating stereo"

Краткое наименование программы – "Стерео-плагин" ("Stereo Plugin”)

## Документы, на основании которых ведется разработка программы

Основанием для разработки является учебный план подготовки бакалавров по направлению 09.03.04 "Программная инженерия" и утвержденная академическим руководителем тема курсового проекта.

# НАЗНАЧЕНИЕ И ОБЛАСТЬ ПРИМЕНЕНИЯ

## Назначение программы

### Функциональное назначение

Программа предназначена для преобразования моно звука в стерео без значительных потерь качества при обратном преобразовании.

### Эксплуатационное назначение

Программа представляет собой плагин для цифровой звуковой рабочей станции (DAW), предназначенный для использования создателями электронной музыки.

## Краткая характеристика области применения

В современном мире большой процент музыки люди слушают в наушниках. А поскольку наушники способны выдавать два отдельных звуковых потока (стерео звук), то современным композиторам нужно создавать музыку с двумя аудиодорожками. Соответственно, возникает задача как сделать из одной аудиодорожке две так, чтобы они звучали как можно более объёмно. Разрабатываемая программа призвана решить данную проблему.

При этом важно понимать, что при прослушивании музыки не в наушниках, звуковые потоки из правой и левой дорожки будут попадать в оба уха, то есть, по сути, происходит обратное преобразование стерео звука в моно. Поэтому разрабатываемая программа должна стремиться к тому, чтобы получающийся в результате обратного преобразования звук был максимально близок к исходному.

# ТЕХНИЧЕСКИЕ ХАРАКТЕРИСТИКИ

## Постановка задачи на разработку программы

Программа должна принимать из DAW блоки данных со звуком (посредством технологии VST), с помощью математических методов преобразовывать их в стерео (то есть разбивать на две звуковые дорожки), и возвращать обратно в DAW. В программе должна быть предусмотрена возможность менять параметры преобразования посредством пользовательского интерфейса.

## Применяемые математические методы в программе

### Оконное преобразование Фурье

Для преобразования звука из временного пространства в частотное в программе используются оконные преобразования Фурье [3]. Для расчёта единоразового преобразования Фурье в момент времени t для частот используется следующая формула:

В данной формуле – значение звукового сигнала в точке n (звуковой сигнал центрирован относительно момента времени t, N – количество значений сигнала, обрабатываемых за одно преобразование, w[n] – некоторая весовая (оконная) функция, R – сдвиг между соседними преобразованиями.

Для оконной функции выбрано прямоугольное окно в связи с его максимально широким спектром принимаемых частот и меньшими потерями при внесении изменений в . [3]

Для количества значений, обрабатываемых за единичное преобразование, было взято значение N = 2048. В качестве сдвига между соседними преобразованиями Фурье было выбрано число R=1024, таким образом достигается перекрытие в 50%.

### Получение фазы и амплитуды

Пусть имеется некоторая частота . Тогда соответствующее ей значение будет равно:

Где N – количество сэмплов (значений звукового сигнала), обрабатываемых за одно преобразование Фурье, а – количество сэмплов звуковой волны, проигрываемых в секунду. Данная формула следует из частоты Найквиста [4].

Амплитуда для данной частоты в момент времени t будет равна:

Фаза, в свою очередь, равна:

### Обратное преобразование

Для обратного преобразования используется следующая формула [1]:

Где нормирующая константа K в общем случае вычисляется как , но для прямоугольного окна получается равна 1.

Затем полученные смешиваются в с помощью метода weight-overlap-add (WOLA) [2].

В качестве весовой функции для WOLA была взята функция Хеннинга [3]:

При пересечении в 50% соседние функции в сумме дают 1 и не нуждаются в дополнительной нормировке.

## Описание и обоснование алгоритма и функционирования программы

Процесс обработки данных в программе состоит из нескольких этапов:

* 1. Накопление блоков данных
  2. Применение преобразования Фурье
  3. Выделение амплитуды и фазы для каждой из частот
  4. Разделение частот на стерео
* Генерация маски
* Применение маски к амплитудам
  1. Преобразование амплитуды и фазы обратно в комплексное число
  2. Обратное преобразование Фурье
  3. Применение WOLA
  4. Накопление полученных данных и возврат результата в DAW

### Накопление блоков данных

Согласно спецификации формата VST, входные данные поступают в аудиоплагин блоками переменной длины. Поскольку для применения преобразований Фурье необходимы блоки одинаковой длины, то данные необходимо накапливать. Это поведение реализовано с помощью класса DataAccumulator. (см. Приложение 2)

Сложность данного решения по времени – O(N), по памяти O(1), где N – количество поступающих сэмплов.

### Применение преобразования Фурье

В силу особенностей работы человеческого слуха человек слышит не всю волну целиком, а воспринимает отдельные частоты. Поэтому для создания стерео звука также необходимо представить звук в виде частот. Для этого были выбраны преобразования Фурье, которые обладают оптимальной скоростью и качеством, а также очень распространены при анализе звука.

Алгоритмическая сложность для быстрых преобразований Фурье – O(K log K), где K – выбранная длина для преобразований Фурье.

Алгоритмическая сложность на промежутке в N сэмплов долей пересечения x – O(N / x). (K в программе было выбрано константой)

### Выделение амплитуды и фазы для каждой из частот

В результате преобразований Фурье получаются комплексные числа, модуль и аргумент которых задают амплитуды и фазы. Также необходимо по частоте в герцах определять соответствующую ей амплитуду в массиве амплитуд. За это отвечает функция *MainProcessor::FFTIndexToFreq*. . (см. Приложение 2)

### Разделение частот на стерео

Наложенное на программу ограничение – сохранение обратного преобразования в моно и отсутствие каких-либо дополнительных частот в стерео – оставляет лишь единственный способ преобразования звука в стерео: направить некоторые частоты из исходного звука в одну звуковую дорожку, а остальные в другую. При этом допускается смешанный вариант, когда часть частоты идёт в левую звуковую дорожку, а оставшаяся часть в правую.

Для удобства данное преобразование было разделено на два этапа:

1. Генерация маски

Для более равномерного звучания частоты распределяются по функции синус. При этом в программе существует два режима разделения:

* Динамический

В этом случае распределение частот динамически меняется с течением времени в зависимости от распределения амплитуд частот. При этом синус быстрее колеблется в тех точках, где амплитуда больше, и медленнее, где амплитуда меньше. В результате достигается баланс громкости между результирующими звуковыми дорожками.

При этом для плавности используется не мгновенная амплитуда, а амплитуды сглаженная. Для этого высчитывается массив средних амплитуд по формуле:

*mediumMagnitude[i] = weight \* mediumMagnitude[i] + (1 – weight) \* magnitude[i]*, где:

*magnitude[i]* – текущая частота,

*weight* – параметр сглаживания, задаваемый пользователем с помощью параметра “*Attack*”.

* Статический

В этом режиме распределение частот статично и не меняется с течением времени. Поскольку в музыке для частот принята логарифмическая шкала (то есть нота в каждой следующей октаве имеет частоту в два раза больше предыдущей), то распределение также

В обоих режимах пользователь может регулировать быстроту колебаний синуса с помощью регулятора “*Frequency Spread*”.

Выбор режима разделения определяется пользователем с помощью кнопки “*Dynamic Split*”.

Также для создания более естественного разделения, в программе есть возможность добавить в маску случайный шум. Чтобы шум был более плавный при генерации очередного значения для частоты x в временной промежуток t, его значение смешивается предыдущими (x-1, t) и (x, t-1). Силу применения шума к маске определяет пользователь с помощью регулятора “*Noise*”.

Генерация маски реализована посредством функции *MainProcessor::generateMask*. (см. Приложение 2)

1. Применение маски к амплитудам

Пусть имеется маска mask со значениями от 0 до 1 и исходные частоты data. Тогда разделение будет посчитано по следующей формуле:

*res1[i] = {data[i].magnitude \* mask[i], data[i].phase};*

*res2[i] = {data[i].magnitude \* (1 - mask[i]), data[i].phase};*

Применение маски реализовано посредством функции *MainProcessor::processSplit*.

### Преобразование амплитуды и фазы обратно в комплексное число

Для данного преобразования используется функция из стандартной библиотеки C++ -*std::polar*.

### Обратное преобразование Фурье

Обратное преобразование Фурье имеет параметры и сложность, аналогичные прямому преобразованию Фурье. При этом данные, полученные после данного преобразования накапливаются в промежуточном буфере *MainProcessor::* *fftOutAccumulators,* который реализован с помощью класса DataAccumulator. (см. Приложение 2)

### Применение WOLA

Для вычисления результата необходимо скомпоновать данные, полученные в результате многочисленных преобразований Фурье. При этом данные на краях каждого из преобразований получаются менее точными, чем в середине. Поэтому для смешивания используется весовая функция Хэмминга. Само смешивание происходит в методе *MainProcessor::process*. (см. Приложение 2)

### Накопление полученных данных и возврат результата в DAW

Данные в WOLA также генерируются блоками постоянного размера, в то время как DAW может запрашивать блоки разного размера. Поэтому получаемые данные накапливаются в специальный *DataAccumulator*, данные из которого передаются в DAW. В результате работы плагина формируется некоторая задержка (латентность) между входными и выходными данными, равная длине преобразований Фурье (2048). Эта задержка сообщается в DAW с помощью функции *getLatencyInSamples*. (см. Приложение 2)

## Описание и обоснование выбора метода организации входных и выходных данных

### Описание метода организации входных и выходных данных

Входными данным для программы являются:

* Входные данные, передаваемые из DAW посредством технологии VST.
* входящий аудиопоток (в формате моно или стерео).
* значения параметров плагина.
* Действия пользователя.
  + взаимодействие с пользовательским интерфейсом.

Выходными данными программы являются:

* Выходные данные, передаваемые в DAW посредством технологии VST:
* результирующий аудиопоток (в формате стерео)
* базовая информация о плагине (название плагина, параметры и другие значения, которые требуется передавать согласно формату VST)
* значения параметров плагина
* Выходные данные для пользователя.
* изображение пользовательского интерфейса

### Обоснование метода организации входных и выходных данных

Метод организации входных и выходных данных определила спецификация формата VST.

## Описание и обоснование выбора состава технических и программных средств

Для корректной и стабильной работы, программе необходимы следующие технические и программные средства:

* Графическая карта: с поддержкой OpenGL 3.0 – необходима для быстрой отрисовки интерфейса плагина;
* Оперативная память: не менее 200 Мб свободной памяти – необходима для хранения промежуточных данных, используемых при преобразованиях, а также для отрисовки;
* Постоянная память: не менее 50 Мб свободной памяти на используемом накопителе – необходима для хранения исполняемого файла плагина;
* Периферийные устройства: Клавиатура, мышь, а также устройство, способное выводить две звуковые дорожки раздельно (лучше всего наушники) – необходимы при работе со стерео;
* Аудиокарта – необходима для вывода звука;
* USB – необходим для подключения мыши, клавиатуры, а так же для подключения USB-накопителя с установщиком плагина;

Также должны быть выполнены все требования, необходимые для используемой совместно с программой DAW.

# ТЕХНИКО-ЭКОНОМИЧЕСКИЕ ПОКАЗАТЕЛИ

## Предполагаемая потребность

В современном мире огромный процент музыки слушается через наушники. Как следствие, при создании электронной музыке нужно каким-то образом разделять звук на два аудиопотока. Поэтому данный плагин может быть полезен практически любому композитору.

## Экономические преимущества разработки по сравнению с отечественными и зарубежными образцами или аналогами

Одним из основных методов для придания звуку объёмности является вставка задержки между левой и правой аудиодорожкой. Данный метод работает за счёт того что в реальной жизни звук тоже приходит в одно ухо немножко раньше, чем в левое, то есть между тем как мы услышим звук одним ухом и другим ухом есть некоторая задержка. Но у этого метода есть два серьёзных недостатка:

1. На самом деле в реальном мире звук отражается от стен, причём волны разных частот и направлений распространяются в воздухе по-разному. Поэтому на самом деле звук в левом ухе и правом ухе отличается не только задержкой. Из-за этого при использовании данного метода человек всё ещё может ощущать некоторую искусственность и нереалистичность.
2. Данный метод сильно искажает музыку, если преобразовать её обратно в моно звук. Задержка создаёт сильный эффект эхо, как будто бы музыку записывали в тесном помещении.

Именно этот метод используется в большинстве аналогов.

Создаваемая программа предполагает использование другого метода для преобразования, поэтому она не должна иметь данных недостатков.

Также стоит отметить, что данная программа будет распространяться бесплатно, в отличие от большинства аудиоплагинов.

# ПРИЛОЖЕНИЕ 1 ТЕРМИНОЛОГИЯ

**VST (Virtual Studio Technology)**  формат зависимых от среды выполнения (native) плагинов реального времени, которые подключаются к звуковым редакторам, секвенсорам, цифровым звуковым рабочим станциям.

**Аудиопоток**  звуковая волна, представленная в электронном виде.

**Аудиоплагин**  программа в специальном формате, которая расширяет или улучшает функционал работы со звуком в другой программе.

**Моно-звук**  звук, представленный в виде одного канала.

**Стерео-звук**  звук, представленный в виде двух каналов, которые воспроизводятся раздельными динамиками.

**Цифровая звуковая рабочая станция / DAW (Digital Audio Workstation)**  компьютерная система, предназначенная для записи, хранения, редактирования и воспроизведения цифрового звука.

# ПРИЛОЖЕНИЕ 2 ОПИСАНИЕ КЛАССОВ В ПРОГРАММЕ

|  |  |
| --- | --- |
| Классы | Назначение |
| DataAccumulator | Класс для накопления (буфферизации) данных для их последующей обработки |
| MainProcessor | Основной класс, производящий преобразование моно-звука в стерео |
| NoiseGenerator | Класс для генерации |
| PluginEditor | Класс, отвечающий за пользовательский интерфейс |
| PluginProcessor | Основной класс программы, задающий все параметры преобразования |
| ProcessingParams | Класс для хранения параметров преобразования |
| ProcessingRendering | Класс для хранения параметров, используемых при отрисовке |
| SpectrumCanvas | Класс для отрисовки результатов обработки в пользовательском интерфейсе |
| SyncedArray | Класс для пересылки данных из потока, преобразующего аудио, в поток, отвечающий за отрисовку интерфейса |

Таблица 3.1

Описание полей, методов и свойств класса DataAccumulator

|  |  |  |  |
| --- | --- | --- | --- |
| Поля класса DataAccumulator | | | |
| Имя | Модификатор доступа | Тип | Назначение |
| buffer | private | kfr::univector<kfr::f32> | Внутренний буффер с данными |
| buffer\_pos | private | int | Текущая позиция во внутреннем буффере |
| buffer\_size | private | int | Текущий размер внутреннего буффера |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Методы класса DataAccumulator | | | | |
| Имя | Модификатор доступа | Тип | Аргументы | Назначение |
| reset | public | void | int maxSamplesPerBlock, int maxRequestLength | Приводит буффер в пустое состояние |
| Update | public | void |  | Обновляет состояние вебсокета |
| addDataSamples | public | void | const kfr::univector\_ref<kfr::f32>& data | Добавляет новые данные в буффер |
| getData | public | kfr::univector\_ref<kfr::f32> | int left, int right | Добавляет новые данные в буффер |

Таблица 3.2

Описание полей, методов и свойств класса MainProcessor

|  |  |  |  |
| --- | --- | --- | --- |
| Поля класса MainProcessor | | | |
| Имя | Модификатор доступа | Тип | Назначение |
| params | private | ProcessingParams& | Ссылка на параметры для преобразования |
| rendering | pubic | ProcessingRendering& | Ссылка на параметры для отрисовки |
| inAccumulators | private | DataAccumulator[2] | Буфферы для накопления входных данных |
| inSumAccumulator | private | DataAccumulator | Буффер для накопления суммы |
| fftOutAccumulators | private | DataAccumulator[2] | Буффер для накопления данных после преобразований для WOLA |
| outAccumulators | private | DataAccumulator[2] | Буффер для накопления выходных данных |
| dftPlan | private | kfr::dft\_plan\_real<kfr::f32> | Хранение кэша для вычисления преобразований Фурье |
| dftPlanTemp | private | kfr::dft\_plan\_real<kfr::f32> | Промежуточный массив для вычиления преобразований Фурье |
| dftWindow | private | kfr::univector<kfr::f32> | Оконная функция для преобразований Фурье |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Методы класса MainProcessor | | | | |
| Имя | Модификатор доступа | Тип | Аргументы | Назначение |
| reset | public | void |  | Сбрасывает состояние класса до изначального |
| process | public | void | kfr::univector<kfr::univector<kfr::f32, 0>, 2> | Производит преобразование звука in\_place |
| processFFTForAccumulator | private | void | DataAccumulator& inAccumulator, DataAccumulator& outAccumulator1,  DataAccumulator& outAccumulator2 | Выполняет преобразования Фурье, преобразования звука и обратное преобразование |
| performIFFTAndWrite | private | kfr::univector\_ref<kfr::f32> | vecfft& in, DataAccumulator& outAccumulator | Применяет обратное преобразование Фурье |
| applyConversionsToFFT | private | void | vecfft& fftData,  vecfft& res1,  vecfft& res2 | Применяет преобразования к комплексным числам, полученным в преобразованиях Фурье |
| processSplit | private | void | vecfft& data, vecfft& res1, vecfft& res2 | Производит разеделние частот на две группы: для левой и правой звуковых дорожек |
| generateMask | private | void | vecfft& data,  kfr::univector<kfr::f32, 0> mask | Создаёт маску для разделения звука на стерео |
| freqToFFTIndex | private | int | int frequency | Преобразует частоту в герцах в индекс для FFT |
| FFTIndexToFreq | private | int | int index | Преобразует индекс в FFT в частоту в герцах |

Таблица 3.3

Описание полей, методов и свойств класса NoiseGenerator

|  |  |  |  |
| --- | --- | --- | --- |
| Поля класса NoiseGenerator | | | |
| Имя | Модификатор доступа | Тип | Назначение |
| distribution | private | std::uniform\_real\_distribution<T> | Случайное распределение |
| values | private | std::array<T, size> | Текущие значения случайного шума |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Методы класса MainProcessor | | | | |
| Имя | Модификатор доступа | Тип | Аргументы | Назначение |
| nextGenerate | public | void | float deltaX, float deltaY | Генерирует следующие значение шума |
| float deltaX, float deltaY | public | T | const size\_t& x | Возвращает значение шума в заданной точке |

|  |  |  |  |
| --- | --- | --- | --- |
| Таблица 3.4  Описание полей, методов и свойств класса PluginEditor | | | |
| Поля класса PluginEditor | | | |
| Имя | Модификатор доступа | Тип | Назначение |
| audioProcessor | private | PluginProcessor& | Ссылка на обработчик звука |
| values | private | std::array<T, size> | Текущие значения случайного шума |
| tooltipWindow | private | std::unique\_ptr<TooltipWindow> | Вспомогательное окно |
| spectrum\_canvas | private | std::unique\_ptr<SpectrumCanvas> | Указатель на компонент, на котором рисуется процесс обработки |
| strengthSlider | private | std::unique\_ptr<juce::Slider> | Указатель на регулятор параметра, отвечающего за силу эффекта |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Методы класса PluginEditor | | | | |
| Имя | Модификатор доступа | Тип | Аргументы | Назначение |
| PluginEditor | public | Конструктор | PluginProcessor& | Создаёт окно с пользовательским интерфейсом |
| parameterChanged | public | void | const String& parameterID, float newValue | Обработчик изменения параметров плагина |

|  |  |  |  |
| --- | --- | --- | --- |
| Таблица 3.4  Описание полей, методов и свойств класса PluginProcessor | | | |
| Поля класса PluginProcessor | | | |
| Имя | Модификатор доступа | Тип | Назначение |
| rendering | public | ProcessingRendering | Параметры для отрисовки |
| params | public | ProcessingParams | Параметры для преобразования |
| mainProcessor | public | MainProcessor | Основной класс, выполняющий преобразования |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Методы класса PluginProcessor | | | | |
| Имя | Модификатор доступа | Тип | Аргументы | Назначение |
| prepareToPlay | public | void | double sampleRate, int samplesPerBlock | Подготавливается к обработке звука с данными параметрами |
| releaseResources | public | void |  | Освобождает ресурсы, которые были задействованы при обработке |
| isBusesLayoutSupported | public | bool | const BusesLayout& layouts | Проверяет, поддеживается ли данная раскладка аудиодорожек |
| processBlock | public | void | juce::AudioBuffer<float>&, juce::MidiBuffer& | Основной метод. Производит преобразование звука |
| processBlockBypassed | publuc | void | juce::AudioBuffer<float>&, juce::MidiBuffer& | Обрабатыавет данные из DAW, но не преобразует их |
| createEditor | public | juce::AudioProcessorEditor\* | - | Создаёт редактор с пользовательским интерфейсом |
| hasEditor | public | bool | - | Возвращает true, поскольку у плагина есть редактор |
| getName | public | const juce::String | - | Возвращщает имя плагина |
| getStateInformation | public | juce::MemoryBlock& destData | - | Возвращает параметры плагина в виде бинарных данных |
| setStateInformation | public | void | const void\* data, int sizeInBytes | Устанавливает параметры на основе бинарных данных |

Таблица 3.5

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Описание полей, методов и свойств класса ProcessingParams | | | | |
| Поля класса ProcessingParams | | | | |
| Имя | Модификатор доступа | Тип | Назначение | |
| tree | public | std::unique\_ptr<juce::AudioProcessorValueTreeState> | Внутреннее представление параметров | |
| leftCutoff | public | std::atomic<float>\* | Частота, начиная с которой происходят преобразования | |
| rightCutoff | public | std::atomic<float>\* | Частота до которой происходият преобразования | |
| strength | public | std::atomic<float>\* | Сила стерео эффекта | |
| frequencySpread | public | std::atomic<float>\* | Распределение эффекта по частотам | |
| attack | public | std::atomic<float>\* | Скорость реакции плагина на изменения в частотах | |
| noise | public | std::atomic<float>\* | Сила применения шума к стерео эффекту | |
| dynamicSplitEnabled | public | std::atomic<float>\* | Должен ли плагин динамически реагировать на изменения в частотах | |
| sampleRate | public | float | Количество сэмплов, которые поступают из DAW в секунду | |
| maxSamplesPerBlock | public | int | Максимальное количество сэмплов, которое может поступить в одном блоке | |
| Методы класса ProcessingParams | | | | |
| Имя | Модификатор доступа | Тип | Аргументы | Назначение |
| ProcessingParams | public | Конструктор | AudioProcessor& processor | Создаёт параметры для плагина |

Таблица 3.6

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Описание полей, методов и свойств класса ProcessingRendering | | | | |
| Поля класса ProcessingRendering | | | | |
| Имя | Модификатор доступа | Тип | Назначение | |
| lastMask | public | SyncedArray<float> | Маска, сгенерированная при последнем преобразовании | |
| lastSound | public | SyncedArray<float> | Частотное разложения звука, использванного в последнем преобразовании | |
| Методы класса ProcessingRendering | | | | |
| Имя | Модификатор доступа | Тип | Аргументы | Назначение |
| ProcessingRendering | public | Конструктор | - | Создаёт параметры для отрисовки |

Таблица 3.7

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Описание полей, методов и свойств класса SpectrumCanvas | | | | |
| Поля класса SpectrumCanvas | | | | |
| Имя | Модификатор доступа | Тип | Назначение | |
| params | private | ProcessingParams& | Параметры для преобразования | |
| rendering | private | ProcessingRendering& | Параметры для отрисовкки | |
| maskPath | private | Path[2] | Векторное представление маски, которая применялась для левой и правой аудиодорожек | |
| soundPath | private | Path | Векторное представление звука, который проигрывался в последний раз | |
| mask | private | std::vector<float> | Промежуточный буффер для хранения маски | |
| sound | private | std::vector<float> | Промежуточный буффер для хранения частотного представления звука | |
| Методы класса SpectrumCanvas | | | | |
| Имя | Модификатор доступа | Тип | Аргументы | Назначение |
| SpectrumCanvas | public | Конструктор | ProcessingParams& \_params, ProcessingRendering& \_rendering | Создаёт холст для отрисовки параметров |
| ~SpectrumCanvas | public | Деструктор | - | Освобождает ресурсы, использованные холстом |
| timerCallback | public | void | - | Обрабатыает событие для отрисовки следующего кадра |
| paint | public | void | juce::Graphics& | Производит саму отрисовку |

Таблица 3.8

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Описание полей, методов и свойств класса SyncedArray | | | | |
| Поля класса SyncedArray | | | | |
| Имя | Модификатор доступа | Тип | Назначение | |
| valuesLock | private | juce::CriticalSection | Блокировка для синхронного доступа к данным | |
| values | private | std::vector<T> | Значения | |
| smoothFactor | private | float | Параметр, показывающий насколько нужно смешивать старые и новые значения | |
| Методы класса SyncedArray | | | | |
| Имя | Модификатор доступа | Тип | Аргументы | Назначение |
| SyncedArray | public | Конструктор | float \_smoothFactor | Создаёт массив для синхронной передачи данных |
| addValues | public | void | T\* newValues, int valuesCount | Добавляет новые значения в массив |
| getValues | public | void | std::vector<T>& array | Копирует значения из массива в массив, переданный по ссылке |
| reset | public | void | - | Сбрасывает состояние массива для передачи данных |
| size | public | size\_t | - | Возвращает размер данных, которые хранятся в массиве |

# ПРИЛОЖЕНИЕ 3 СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ

1. Inverse short-time Fourier transform [Электронный ресурс] / MathWorks. Режим доступа: <https://www.mathworks.com/help/signal/ref/istft.html>, свободный (дата обращения: 17.05.2021).
2. Julius Orion Smith, Spectral Audio Signal Processing, WOLA Processing Steps [Электронный ресурс] / Stanford University. Режим доступа: <https://ccrma.stanford.edu/~jos/sasp/WOLA_Processing_Steps.html>, свободный (дата обращения: 17.05.2021).
3. Оконное преобразование Фурье [Электронный ресурс] / Википедия. Режим доступа: <https://ru.wikipedia.org/wiki/Оконное_преобразование_Фурье>, свободный (дата обращения: 17.05.2021).
4. Частота Найквиста [Электронный ресурс] / Википедия. Режим доступа: <https://ru.wikipedia.org/wiki/Частота_Найквиста>, свободный (дата обращения: 17.05.2021).

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Лист регистрации изменений | | | | | | | | | |
| Номера листов (страниц) | | | | | Всего листов (страниц в докум.) | № документа | Входящий  № сопроводит ельного докум. и дата | Подп. | Дата |
| Изм. | Изменен ных | Заменен ных | Новых | Аннули рованных |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |

# ЛИСТ РЕГИСТРАЦИИ ИЗМЕНЕНИЙ