СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 6](#_Toc484388270)

[**1** ОБЗОР ЛИТЕРАТУРЫ 8](#_Toc484388271)

[**1.1** Цифровые аудио-станции и плагины 8](#_Toc484388272)

[**1.2** Octaver 10](#_Toc484388273)

[**1.3** Delay 13](#_Toc484388274)

[**1.4** Reverb 14](#_Toc484388275)

[**2** СИСТЕМНОЕ ПРОЕКТИРОВАНИЕ 16](#_Toc484388276)

[**2.1** Структура программного модуля 16](#_Toc484388277)

[**2.2** Выбор программных средств 17](#_Toc484388278)

[**3** ФУНКЦИОНАЛЬНОЕ ПРОЕКТИРОВАНИЕ 21](#_Toc484388279)

[**3.1** Блок интерфейса пользователя 21](#_Toc484388280)

[**3.2** Блок настройки параметров обработки 30](#_Toc484388281)

[**3.3** Класс DelayLine 33](#_Toc484388282)

[**3.4** Блок эффекта «delay» 35](#_Toc484388283)

[**3.5** Блок эффекта «reverb» 36](#_Toc484388284)

[**3.6** Фильтры 39](#_Toc484388285)

[**3.7** Блоки преобразования входного и выходного сигналов 42](#_Toc484388286)

[**4** РАЗРАБОТКА ПРОГРАММНЫХ МОДУЛЕЙ 43](#_Toc484388287)

[**4.1** Функции класса ShimmerAudioProcessorEditor 43](#_Toc484388288)

[**4.2** Функции класса ShimmerAudioProcessor 44](#_Toc484388289)

[**4.3** Функции класса Delay 52](#_Toc484388290)

[**4.4** Функции класса SReverb 53](#_Toc484388291)

[**5** ПРОГРАММА И МЕТОДИКА ИСПЫТАНИЙ 57](#_Toc484388292)

[**6** РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ 62](#_Toc484388293)

[**6.1** Системные требования 62](#_Toc484388294)

[**6.2** Процесс инсталляции 62](#_Toc484388295)

[**6.3** Пользовательский интерфейс 66](#_Toc484388296)

[**7** ТЕХНИКО-ЭКОНОМИЧЕСКОЕ ОБОСНОВАНИЕ ЭФФЕКТИВНОСТИ РАЗРАБОТКИ И РЕАЛИЗАЦИИ ПРОГРАММНОГО ПРОДУКТА 70](#_Toc484388297)

[**7.1** Характеристика программного продукта 70](#_Toc484388298)

[**7.2** Расчет сметы затрат на разработку и отпускной цены программного продукта 70](#_Toc484388299)

[**7.3** Расчёт экономического эффекта от реализации ПО 75](#_Toc484388300)

[**7.4** Расчет показателей экономической эффективности инвестиций в разработку и реализацию программного продукта 76](#_Toc484388301)

[ЗАКЛЮЧЕНИЕ 80](#_Toc484388302)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 81](#_Toc484388303)

[ПРИЛОЖЕНИЕ А 82](#_Toc484388304)

[ПРИЛОЖЕНИЕ Б 110](#_Toc484388305)

[ПРИЛОЖЕНИЕ В 111](#_Toc484388306)

# ВВЕДЕНИЕ

Развитие вычислительной математики и техники привело к тому, что звуки, используемые человечеством для повседневной жизни, стали переводиться из аналоговой формы в цифровую. Цифровой звук имеет перед аналоговым огромные преимущества, хотя нельзя забывать и об определенных его недостатках. Основной ценностью цифрового звука является возможность бесконечно долгого хранения и бесконечного тиражирования материала без потери исходного качества, тогда как у аналогового звука качество теряется при каждой записи-перезаписи. Немаловажно и то, что цифровая техника, в отличие от аналоговой, позволяет добиваться идентичности параметров систем при их массовом производстве и сохраняет эту идентичность при эксплуатации, в то время как характеристики аналоговых изделий обычно отличаются на разных экземплярах и ухудшаются со временем. Кроме того, облегчаются передача звука и его обработка современными цифровыми средствами, в первую очередь, специализированными компьютерами.

Программные модули синтеза и обработки звука широко используются в уже давно популярных жанрах электронной музыки, создавая звуки, непривычные человеческому уху, а потому необычные, интересные. Но это только самое очевидное и, конечно же, не единственное приложение. Прогресс приводит к тому, что становится возможным создание модулей обработки звука, эмулирующих настоящие звукозаписывающие студии с любым оборудованием: педалями эффектов, гитарными усилителями и гитарными кабинетами, микрофонами, комнатами. Кроме того, каждый коммерческий и большинство некоммерческих музыкальных произведений проходит ряд обработок на стадии сведения и мастеринга, где исправляются некоторые дефекты записи, корректируются параметры для соответствия стандартам, добавляются эффекты и другое.

Ещё одним достоинством цифрового звука является возможность эмуляции любого аналогового прибора. В настоящее время любой реальный аналоговый прибор можно эмулировать с довольно высокой точностью, которая определяется в большей степени работой по сбору информации о приборе, нежели ограничениями компьютера. Более того, программные модули позволяют получить эффекты, которые невозможно создать с помощью аналоговых приборов. В настоящее время вычислительных ресурсов даже рядовых персональных компьютеров достаточно, чтобы запускать сразу несколько десятков программных модулей обработки звука в реальном времени.

Прогресс электротехники достиг такого уровня, что недостатки цифрового звука становятся ничтожными и меркнут перед достоинствами. Ведь уже несколько лет изготавливаются и широкодоступны АЦП и ЦАП с параметрами достаточными для того, чтобы разница между исходным аналоговым и оцифрованным звуком не была заметна человеческому уху.

Гибкость, удобство, универсальность, дешевизна программного обеспечения для обработки звука и хорошее качество цифрового звука практически не оставляют причин музыкантам, звукорежиссёрам и мастеринг-инженерам не переходить с аналогового оборудования на цифровое.

Таким образом, исходя из вышеперечисленных достоинств цифрового звука и его цифровой обработки, становится понятным, что создание программного обеспечения для обработки звука является актуальной темой. Этот факт, а также моя любовь к музыке обусловили выбор темы дипломного проекта. Цель проекта: разработать программный модуль обработки звуковой дорожки в реальном времени.

# ОБЗОР ЛИТЕРАТУРЫ

## Цифровые аудио-станции и плагины

Цифровая звуковая рабочая станция – электронная или компьютерная система, предназначенная для записи, хранения, редактирования и воспроизведения цифрового звука. Предусматривает возможность выполнения на ней законченного цикла работ, от первичной записи до получения готового результата. Плагины обычно выполняются в виде разделяемых библиотек [1]. Наиболее часто плагины для обработки звука используются в хост-программе DAW (Digital Audio Workstation – цифровая звуковая рабочая станция), где они применяются к отдельным дорожкам или их группам. Примеры наиболее популярных DAW с кратким описанием в таблице 1.1.

|  |  |
| --- | --- |
| Таблица 1.1 – Примеры и описание наиболее популярных DAW | |
| DAW | Описание |
| FL Studio | Цифровая звуковая рабочая станция (DAW) и секвенсер для написания музыки. Музыка создаётся путём записи и сведения аудио- или MIDI-материала. |
| Cubase | Программное обеспечение для создания, записи и микширования музыки. Обладает мощным аудиодвижком и встроенными профессиональными инструментами |
| Nuendo | Отличается от другой популярной линии программных продуктов — Steinberg Cubase тем, что ориентирована не только на музыкантов, но и на любую профессиональную деятельность по озвучиванию в том числе фильмов, телевизионных программ, реклам, радиопередач и прочего. |
| Ableton Live | Программа используется как для студийной работы (аранжировка, сведение), так и для живой игры (импровизация, DJ-инг), и имеет два режима: «Arrangement View» и «Session View». |
| REAPER | Аудиостанция обладает широкими функциональными возможностями и является развитой, профессиональной рабочей системой для создания, записи, редактирования и микширования аудио и MIDI материала, а также мастеринга композиций. При этом программа имеет относительно небольшие размеры. |
| Logic Pro X | Logic Pro X включает огромную коллекцию высококачественных музыкальных сэмплов, инструментов, эффектов и циклов — всё, что нужно для создания композиций профессионального уровня. |

В программах обработки звука плагины выполняют обработку и создание звуковых эффектов, например, мастеринг, применение эквалайзера и сжатие динамического диапазона [2]. Некоторые плагины изменяют технические характеристики звука: глубину, частоту дискретизации и прочее. Практически все аудио-плагины имеют графический пользовательский интерфейс. В GUI (graphical user interface – графический интерфейс пользователя) присутствуют элементы управления (чаще всего это какие-нибудь ручки), меняющие то, как плагин обрабатывает входящие данные. Часто плагин или standalone (автономное) приложение уже имеет встроенный набор пресетов (от английского preset – сохранённый набор настроек), в которых хранятся положения ручек и других параметров. Также часто имеется возможность сохранять свои собственные пресеты. Наиболее распространённый формат аудио-плагинов – VST.

Обычно виды обработок аудио дорожки классифицируют образом, отражённым в таблице 1.2.

|  |  |
| --- | --- |
| Таблица 1.2 – Типы и виды обработок звука | |
| Тип обработки | Вид обработки |
| Частотная | * filter, equalizer, pitchshifter. |
| Динамическая | noise gate, compressor, limiter, soft clipper, expander. |
| Модуляционная | * chorus, flanger, phaser. |
| Частотно-динамическая | multiband compressor. |
| Пространственная | delay, echo, reverb. |
| Искажения | overdrive, distortion, fuzz. |

Программные модули для обработки звука могут работать либо со стерео, либо с моно дорожкой, либо и с той и другой.

Virtual Studio Technology (VST) — формат ресурсозависимых (native) плагинов реального времени, которые подключаются к звуковым редакторам и музыкальным редакторам, секвенсорам. Формат был разработан совместно Propellerhead и Steinberg, впоследствии Propellerhead отказался от дальнейших работ над VST и дальнейшая разработка осуществлялась исключительно Steinberg. В настоящее время в этом формате существуют тысячи плагинов, он стал одним из самых распространённых для звуковых программ. Приложения VST отличаются от плагинов DirectX по нескольким параметрам; в частности, они существуют для Windows, Mac OS X и Linux. Кроме того, в отличие от ранних версий DirectX, плагины VST обладают развитым интерфейсом автоматизации [3].

В данном дипломном проекте разрабатывается программный модуль обработки моно и стерео дорожек с использованием следующих основных эффектов: delay и reverb. Дополнительной задачей является разработка эффекта octaver.

## Octaver

Octaver – звуковой эффект или соответствующее устройство, добавляющее к сигналу его копию на октаву или две ниже или выше основного тона. Само обозначение применяется преимущественно к обработке звука электрогитары [4].

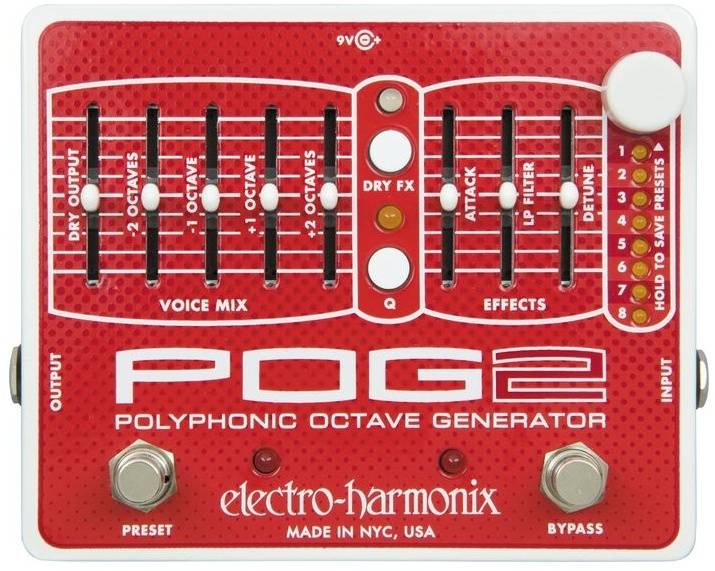


Рисунок 1.1 – Electro-Harmonix POG 2

Octaver является частным случаем эффекта pitch shifter (сдвиг тона). При использовании данного эффекта создаётся впечатление, что вместо одного инструмента играют два в разных октавах. В некоторых педалях эффектов, таких, как, например, Boss OC-2, имеется возможность добавлять два дополнительных тона одновременно – один на октаву ниже основного, другой на две октавы ниже основного. При этом каждый из них можно регулировать по громкости, смешивая в желаемой пропорции. Большинство педалей эффектов типа octaver монофонические, то есть не могут достраивать октавы к аккордам. Полифонические (цифровые) эффекты могут обрабатывать сигнал, содержащий несколько нот. Примером цифрового эффекта octaver является педаль Electro-Harmonix POG 2 (рисунок 1.1).

Для реализации этого эффекта необходимо получить представление звукового сигнала в частотной области, затем обработать его и перевести во временную область.

Дискретное преобразование Фурье (ДПФ) – одна из двух наиболее распространённых и мощных процедур цифровой обработки сигналов (другая процедура – цифровая фильтрация). ДПФ позволяет анализировать, преобразовывать и синтезировать сигналы такими способами, которые невозможны при непрерывной (аналоговой) обработке [5].

ДПФ – это математическая процедура, используемая для определения гармонического или частотного состава дискретных сигналов. Истоком ДПФ является непрерывное преобразование Фурье X(f), которое определяется как

|  |  |
| --- | --- |
| , | (1.1) |

где – некоторый непрерывный сигнал во временной области.

С появлением и развитием цифровых вычислительных машин, работающих с дискретным представлением сигналов, было разработано ДПФ. Оно определяется как дискретная последовательность X(m) в частотной области:

|  |  |
| --- | --- |
| , | (1.2) |

где – дискретная последовательность значений, полученных дискретизацией во временной области непрерывной переменной ;

– основание натурального логарифма;

– константа;

– мнимая единица.

Точные значения частоты разных синусоид зависят как от частоты дискретизации , с которой был дискретизирован исходный сигнал, так и от количества отсчётов N. Все частоты, соответствующие , кратны основной частоте. N разных частот анализа ДПФ определяются выражением

|  |  |
| --- | --- |
|  | (1.3) |

Практический интерес представляют амплитуда и фаза каждого отсчёта . Если представить произвольный отсчёт ДПФ как сумму действительной и мнимой частей

|  |  |
| --- | --- |
| , | (1.4) |

то амплитуда вычисляется как

|  |  |
| --- | --- |
| , | (1.5) |

а фазовый угол вычисляется как

|  |  |
| --- | --- |
| . | (1.6) |

Чтобы получить исходный сигнал во временной области необходимо выполнить обратное дискретное преобразование Фурье (ОДПФ). Выражение для ОДПФ имеет вид:

|  |  |
| --- | --- |
| . | (1.7) |

В 1965 году была опубликована статья Кули и Тьюки [6], описывающая эффективный алгоритм реализации ДПФ. Этот алгоритм сегодня известен как быстрое преобразование Фурье (БПФ).

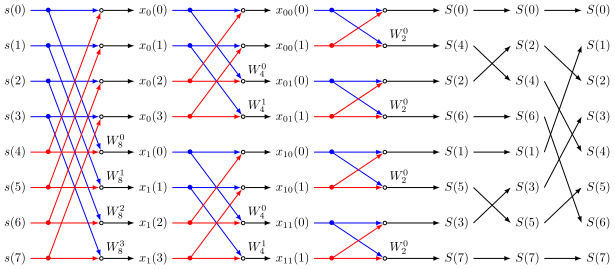


Рисунок 1.2 – Полный граф алгоритма БПФ с прореживанием по частоте для

Алгоритм БПФ по основанию 2 – это эффективный алгоритм вычисления ДПФ, когда длина ДПФ равна натуральной степени двух. Из-за специфической формы элементарных элементов графа он получил название «бабочка». Граф состоит из бабочек. Данная процедура объединения является основной при построении алгоритмов БПФ по основанию два. На рисунке 1.2 представлен полный граф алгоритма БПФ с прореживанием по частоте для N=8 [7].

Операция бабочки в данном алгоритме выполняется в соответствии с формулами 1.8 и 1.9.

|  |  |
| --- | --- |
| ; | (1.8) |

|  |  |
| --- | --- |
| . | (1.9) |
|  |  |

## Delay

Delay – звуковой эффект или соответствующее устройство, имитирующие чёткие затухающие повторы (эхо) исходного сигнала. Эффект реализуется добавлением к исходному сигналу его копии или нескольких копий, задержанных по времени. Под термином delay обычно подразумевается однократная задержка сигнала, в то время как эффект «эхо» - многократные повторы. По принципу действия является частным случаем ревербератора. Отличие заключается в том, что delay имеет одну линию задержки и больший временной интервал (не менее 50-60 мс), который позволяет отделить оригинальный звук от эффекта на слух [8].

Delay – довольно простой эффект для реализации. Входной сэмпл (от английского sample – элемент выборки, замер) сигнала сохраняется в буфер и суммируется с прошлым сэмплом, отстающим в буфере на некоторое число N сэмплов. Обычно прошлый сэмпл ослабляется. И ослабление и длина задержки (N) обычно управляются пользователем [9]. На рисунке 1.3 изображена схема простого эффекта delay.

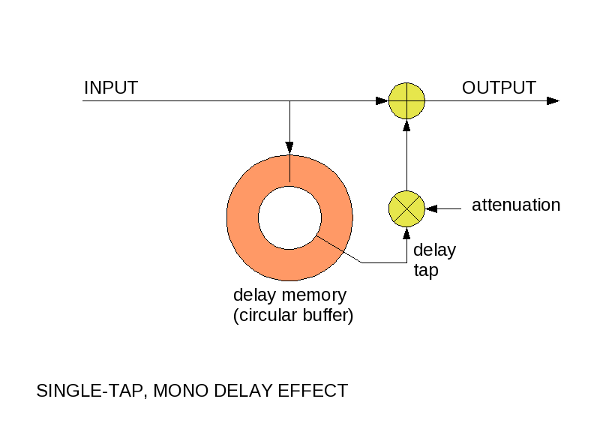


Рисунок 1.3 – Схема простого эффекта delay

Параметры эффекта:

* обратная связь (feedback, regeneration) – при отсутствии обратной связи на выходе будет одна задержка, при увеличении её значения растёт и количество сигналов на выходе;
* время задержки (delay, time) – промежуток времени между исходным сигналом и его задержкой (задержками);
* баланс (balance, mix) – соотношение исходного и задержанного сигналов.

Основные типы эффекта delay:

* slapback – одиночная задержка длительностью до 120 мс;
* echo – более длительная задержка с обратной связью;
* reverse – в цифровых эффектах возможно воспроизведение записанного в буфер сигнала в обратном порядке;
* ping pong – поочерёдное панорамирование задержанного сигнала в левый/правый канал.

В данном проекте реализуется эффект delay, с изменяемой задержкой в пределах от 20 до 1000 мс, с возможностью синхронизации с темпом музыкальной композиции.

## Reverb

Реверберация – это процесс постепенного уменьшения интенсивности звука при его многократных отражениях. Иногда под реверберацией понимается имитация данного эффекта с помощью ревербераторов [10].

Явление реверберации состоит в суперпозиции различных эхосигналов от одного источника звука. Эффект реверберации можно наблюдать в закрытых помещениях после выключения источника звука. Обычно избыточная длительность реверберации приводит к неприятной гулкости, «пустоте» помещения, а недостаточная – к резкому отрывистому звучанию, лишённому музыкальной полноты. Искусственно создаваемая реверберация в определённых пределах способствует улучшению качества звучания, создавая ощущение приятного «резонанса» помещения.

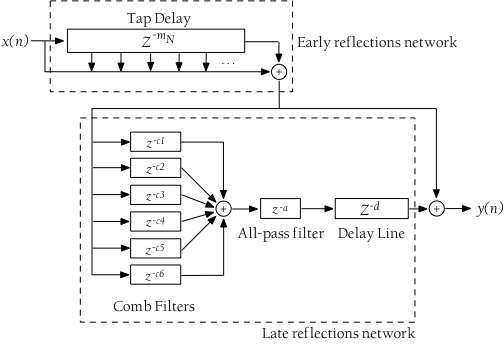


Рисунок 1.4 – Структурная схема ревербератора Мурера

Реверберацию можно разделить на два компонента, которые видны на представлении импульсного отклика в комнате:

1. Ранние отражения – первые отражения, которые мы слышим в течение приблизительно 100 мс вместе с прямым звуком от источника.
2. Поздняя реверберация – слышимый звук после 100 мс, до момента его полного затухания. Поздняя реверберация характеризуется плотной текстурой рассеянных отражений, которые достигают наших ушей несколькими путями. Эти рассеянные отражения находятся не в фазе относительно друг друга, что вызывает эффект гребенчатой фильтрации. Мы воспринимаем этот эффект как «атмосферу», характер места [11].

Структурная схема обработки сигнала ревербератором показана на рисунке 1.4. Такое решение предложил звуковой инженер, музыкант и доктор наук Стэнфордского университета Джеймс Энди Мурер, усовершенствовав алгоритм Шрёдера [12]. Исходя из Алгоритма Мурера, к исходному сигналу добавляется сигнал, прошедший через сеть ранних отражений, состоящую из множества delay -линий. Далее к нему добавляется сигнал обработанный сетью поздних отражений, состоящей из параллельных гребенчатых фильтров и фазового фильтра с delay-линией.

Гребенчатый фильтр — в обработке сигналов электронный фильтр, при прохождении сигнала через который к нему добавляется он сам с некоторой задержкой. В результате получается фазовая компенсация. АЧХ гребенчатого фильтра состоит из ряда равномерно распределённых пиков, так что она выглядит как гребёнка.

В цифровых системах, фильтр задаётся формулой 1.10:

|  |  |
| --- | --- |
| , | (1.10) |

где – коэффициенты;

– запаздывание сигнала;

– входной сигнал;

– выходной сигнал.

На данный момент существуют аналоги отдельных эффектов, реализуемых в данном дипломном проекте, как бесплатные, так и довольно дорогие. Комбинация этих эффектов создаёт новый, необычный, слабо представленный на рынке эффект. В определённых кругах этот эффект называют эффектом shimmer (от английского – мерцание). Целью данного дипломного проекта является создание программного модуля эффекта shimmer с возможностью настройки каждого из подкомпонентов в отдельности.

# СИСТЕМНОЕ ПРОЕКТИРОВАНИЕ

Разрабатываемый программный модуль разбит на отдельные логически взаимосвязанные блоки, что является необходимым условием для обеспечения гибкости его структуры. При данном подходе становится возможной выборочная модернизация отдельных частей программного кода, с минимальным влиянием на остальные части проекта, либо, в идеальном случае, вовсе без их изменения.

## Структура программного модуля

В соответствии с методологией системного подхода в разработке архитектуры, программный модуль разбивается на совокупность сущностей, представленных на структурной схеме (см. чертеж ГУИР.400201.161 С1).

Структурная схема была составлена исходя из основных стадий обработки сигнала, а также функций, которые должен предоставлять программный модуль обработки звуковой дорожки в реальном времени. Структурная схема состоит из следующих блоков:

* блок преобразования входного сигнала для обработки;
* блок эффекта «octaver»;
* блок эффекта «delay»;
* блок эффекта «reverb»;
* блок преобразования обработанных данных в выходной сигнал;
* блок настройки параметров обработки;
* блок интерфейса пользователя.

*Блок преобразования входного сигнала для обработки* принимает исходный сигнал. Перед передачей сигнала к блокам добавления эффектов в нём устраняются шумы. Это выполняется за счёт фильтра нижних частот, фильтра верхних частот и noise gate. Также корректируется уровень входного сигнала для обработки последующими блоками.

*Блок эффекта «*octaver*»* принимает уже подготовленный к обработке звуковой сигнал. Блок предназначен для добавления гармоник частотой степени двойки к исходному звуковому сигналу. Внутри блока с помощью ДПФ получается сигнал в частотном домене. Далее сигнал обрабатывается и с помощью ОДПФ переводится во временной домен. Затем обработанный сигнал складывается с исходным сигналом в задаваемой пользователем пропорции и подаётся на следующий логический блок.

*Блок эффекта «*delay*»* создаёт копию поступившего на вход сигнала и добавляет её к исходному через некоторое задаваемое пользователем время. Сигнал с выхода снова подаётся на вход этого блока уже с уменьшенным уровнем. Внутри этого блока также происходит обработка сигнала, подаваемого с выхода на вход. С выхода блока эффекта «delay» сигнал поступает на блок эффекта «reverb».

*Блок эффекта «*reverb*»* отвечает за эмуляцию постепенного уменьшения интенсивности звука при его многократных отражениях. Он включает в себя два компонента: блок ранних отражений и блок поздних отражений.

*Блок преобразования обработанных данных в выходной сигнал* стоит последним в тракте следования сигнала. Он включает в себя фильтры низких и высоких частот. Также этот блок отвечает за уровень сигнала, подаваемого на выход.

*Блок интерфейса пользователя* представляет собой совокупность средств, при помощи которых пользователь взаимодействует с программным модулем обработки звуковой дорожки в реальном времени. Для построения интерфейса используются компоненты, встроенные в фреймворк Juce (о фреймворке Juce написано в подразделе 2.2). Данный блок отвечает за получение ввода пользователя и связан лишь с блоком настройки параметров обработки двусторонней связью.

*Блок настройки параметров обработки* предназначен для преобразования ввода пользователя и хранения параметров, влияющих на работу блоков эффектов. Среди таких параметров обязательно будут присутствовать следующие:

* уровень входного сигнала (input level) для блока преобразования входного сигнала для обработки;
* соотношение обработанного сигнала к необработанному (mix) для всех эффектов;
* время задержки (delay time) для эффектов delay и reverb;
* уровень сигнала подаваемого с выход на вход (feedback) для эффекта delay;
* уровень выходного сигнала (output level) для блока преобразования обработанных данных в выходной сигнал.

Данный блок кроме двусторонней связи с блоком интерфейса пользователя имеет исходящие двусторонние связи с каждым из блоков эффектов обработки звуковой дорожки, а также блоками преобразования входного сигнала для обработки и блоком преобразования обработанных данных в выходной сигнал.

## Выбор программных средств

В качестве языка программирования задан язык C++, а в качестве среды разработки – Visual Studio 2012 Express. Для выполнения поставленной задачи необходимо выбрать фреймворк, который совместим с требованиями задания, позволяет избежать низкоуровневого программирования, включающий компоненты графического интерфейса пользователя, а также компоненты для работы со звуком. Фреймворк Juce удовлетворяет всем вышеперечисленным требованиям.

Juce – это открытый кроссплатформенный инструментарий разработки ПО (фреймворк) для языка C++, используемый для разработки GUI приложений и плагинов [13].

Цель Juce – позволить компилировать один и тот же исходный текст в программы, одинаково работающие на Windows, Mac OS X и Linux (последние версии – также iPhone и Android) платформах. Он поддерживает различные среды разработки и компиляторы, такие как GCC, Xcode и Visual Studio.

Juce впервые опубликован в 2004, держателем его кода является британская компания Raw Material Software. Имеет двойную GPL/коммерческую лицензию.

Juce содержит классы, позволяющие программе работать со звуком и графикой. За счёт этого нуждающиеся в дополнительных библиотеках программисты могут использовать только библиотеку Juce, или хотя бы сократить количество сторонних библиотек, которые они используют. На это разработчиков Juce вдохновил JDK языка Java. По их заявлению, они собирались из Juce сделать «что-то подобное для С++».

Наиболее важная особенность Juce по сравнению с другими аналогичными фреймворками — большой набор аудиофункций. Дело в том, что библиотека Juce сначала была разработана как часть аудиосеквенсора Tracktion, и лишь затем стала самостоятельным продуктом. Juce включает в себя поддержку воспроизведения звука через аудио и MIDI интерфейсы, полифонические синтезаторы, понимает файлы распространённых аудиоформатов (таких как WAV, AIFF, FLAC, и Vorbis). Он также содержит интерфейсы-оболочки для построения различных аудио плагинов, таких как эффекты и инструменты VST. Это привело к его широкому распространению в сообществе разработчиков аудио-ПО.

В поставку Juce входят классы-обёртки для создания аудиоплагинов. При сборке аудиоплагина, получается единый бинарный файл, который поддерживает несколько форматов: VST, RTAS, AU. Поскольку весь платформо- и форматозависимый код содержится в классах-обёртках, то пользователь может собирать плагины в формате VST/RTAS/AU для макинтошей и Windows из одного и того же исходного кода.

Имеется также неофициальное ответвление библиотеки, расширенное дополнительными возможностями, поддерживаемое сообществом, которое называется Juced.

На сайте этого варианта фреймворка можно найти также дополнительную документацию по Juce, которая поможет освоить библиотеку [14]. Последняя версия фреймворка на данный момент – 4.3.

Фреймворк Juce имеет встроенную хост-программу Juce Plug-In Host [15]. Она имеет только самый необходимый функционал для хост-программы и требует небольшого объёма вычислительных ресурсов компьютера, что позволяет быстро проверять работу плагина. Кроме того с её помощью можно производить отладку плагина, что очень важно для программиста. Чтобы разрабатываемый плагин обрабатывал входной сигнал необходимо соединить блок Audio Input и Midi Input с входами плагина, а выходы плагина соединить с блоком Audio output. К входам разрабатываемого плагина также подключён плагин AudioFilePlayer, который используется для проигрывания записанной звуковой дорожки. Интерфейс программы Juce Plug-In Host изображён на рисунке 2.1.

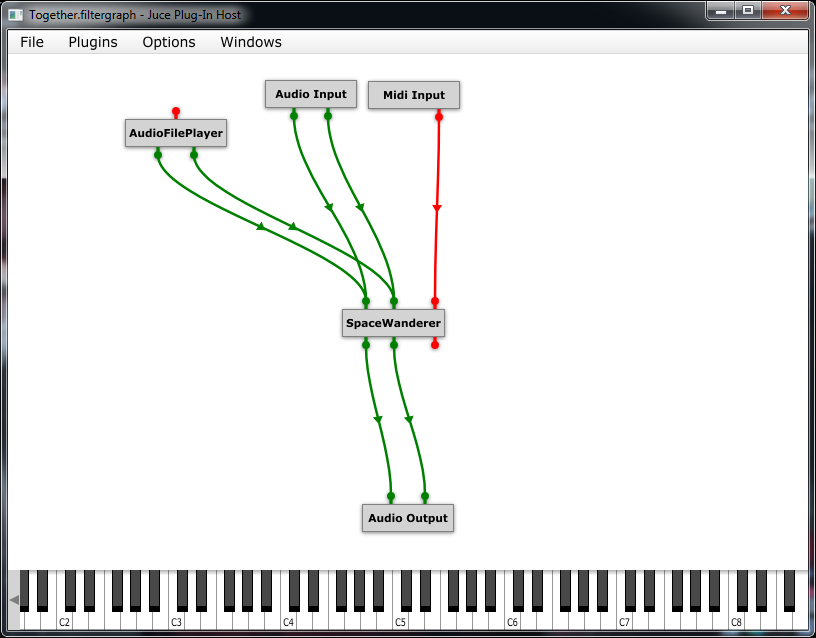


Рисунок 2.1 – Интерфейс хост-программы Juce

Для разработки VST плагинов необходимо скачать пакет VST SDK с официального сайта Steinberg – разработчика этого формата. Текущая версия этого средства разработки – 3.6.7.

Также при разработке программного модуля обработки звуковой дорожки в реальном времени будет использоваться система управления версиями – Git. Выбор сделан в пользу именно этой системы управления версиями из-за ряда причин:

1. Git использует распределённую модель вместо традиционной клиент серверной. Она не нуждается в централизованном хранилище: вся история изменения документов хранится на каждом компьютере в локальном хранилище. При необходимости локальный репозиторий может быть передан командой push на удалённый репозиторий.
2. Постоянное подключение к сети не является необходимым.
3. Git обладает высокой производительностью.
4. Продуманная и уже знакомая мне система команд.

В качестве сервиса предоставляющего хостинг для размещения git-репозиториев выбран веб-сайт GitHub [16].

Таким образом, в данном дипломном проекте при разработке программного модуля обработки звуковой дорожки в реальном времени

* используется модульное программирование;
* в качестве фреймворка выбрана библиотека Juce;
* в качестве формата ресурсозависимого плагина реального времени для DAW выбран VST;
* используется система контроля и управления версиями Git;
* удалённый репозиторий, содержащий пояснительную записку, чертежи, документы и проект программного модуля обработки звуковой дорожкой в реальном времени хранится на сайте GitHub по ссылке https://github.com/Andrewregrets/VST-Plugin.

# ФУНКЦИОНАЛЬНОЕ ПРОЕКТИРОВАНИЕ

В данном разделе подробно рассматривается функционирование программных модулей разработанного программного продукта. Раздел структурирован в соответствии с блоками, выделенными на этапе системного проектирования. В нём перечислены соответствующие классы и большинство их компонентов, состав и отношения которых показаны на диаграмме классов (см. чертёж ГУИР.400201.161 РР.1).

## Блок интерфейса пользователя

Основным классом, который представляет *блок интерфейса пользователя* является ShimmerAudioProcessorEditor (см. рисунок 3.1). Данный класс унаследован от класса AudioProcessorEditor, который предоставляет фреймворк Juce. Данный класс отвечает за обработку пользовательского ввода и отображение графической информации на экран.



Рисунок 3.1 – Организация блока интерфейса пользователя

Класс ShimmerAudioProcessorEditor имеет следующие поля:

* const int window\_width;
* const int window\_height;
* Image background\_image;fbdfnsdmn,gkj.;
* ShimmerAudioProcessor &processor;
* ScopedPointer<Label> InputGainLabel;
* ScopedPointer<Slider> InputGainKnob;
* ScopedPointer<Label> OutputGainLabel;
* ScopedPointer<Slider> OutputGainKnob;
* Octaver\_GUI\_Components octaver\_gui\_components;
* Delay\_GUI\_Components delay\_gui\_components;
* Reverb\_GUI\_Components reverb\_gui\_components.

Константные поля window\_width и window\_height инициализируются в конструкторе данного класса и содержат соответственно ширину и высоту окна.

Поле background\_image необходимо для отображения фонового изображения программного модуля.

Поле processor содержит ссылку на объект типа ShimmerAudioProcessor (см. рисунок 3.2). Это поле инициализируется при создании объектом типа ShimmerAudioProcessor объекта типа ShimmerAudioProcessorEditor и используется для доступа к параметрам обработки звука программным модуля.



Рисунок 3.2 – Агрегирование по ссылке класса ShimmerAudioProcessor классом ShimmerAudioProcessorEditor

Поля InputGainLabel, InputGainKnob, OutputGainLabel, OutputGainKnob – указатели на элементы графического интерфейса, не относящиеся ни к одному из компонентов программного модуля обработки звуковой дорожки в реальном времени. InputGainKnob и OutputGainKnob – ручки, отвечающие соответственно за регулировку входного и выходного уровней сигнала. InputGainLabel и OutputGainLabel – метки для отражения смысла этих ручек.

Класс ScopedPointer предоставлен фреймворком Juce. Данный класс содержит указатель, который автоматически удаляется, когда объект этого класса выходит за свою зону видимости, то есть данный класс следует идиоме RAII (от английского Resource Acquisition Is Initialization – получение ресурса есть инициализация), согласно которой получение ресурса неразрывно совмещается с инициализацией, а освобождение — с уничтожением объекта. RAII является ключевой концепцией для написания безопасного при исключениях кода в языках программирования, где конструкторы и деструкторы автоматических объектов вызываются автоматически, прежде всего – в C++.



Рисунок 3.3 – Компоненты графического интерфейса

Поля octaver\_gui\_components, delay\_gui\_components и reverb\_gui\_components содержат элементы графического интерфейса, относящиеся соответственно к эффектам Octaver, Delay и Reverb (см. рисунок 3.3). Каждый из классов этих объектов имеет конструктор, имеющий три параметра. Первый параметр – указатель на объект типа AudioProcessorEditor, необходим для вызова метода addAndMakeVisible(Component\*), который необходимо вызывать для всех компонентов графического интерфейса для их отображения. Второй параметр – указатель на объект типа ButtonListener. Он необходим, так как используется в качестве параметра функции addListener(ButtonListener\*), которая является членом класса TextButton и предназначена для задания объекта, ответственного за обработку событий связанных с объектом класса TextButton. Такими событиями могут быть, например, нажатие кнопки, наведение на кнопку указателя мыши. Третий параметр – указатель на объект типа SliderListener. Он необходим, так как используется в качестве аргумента функции addListener(SliderListener \*), которая является членом класса Slider и предназначена для задания объекта, ответственного за обработку событий связанных с объектом класса Slider. Примером таких событий является изменение значения путём кручения ручки.

Конструктор выделяет память для компонентов графического интерфейса, делает их видимыми, задаёт их параметры визуализации, то есть определяет то, как эти компоненты будут отображаться. В конструкторе также задаются граничные значения параметров, задаваемых ручками.

Кроме однотипного конструктора, классы Octaver\_GUI\_Components, Delay\_GUI\_Components и Reverb\_GUI\_Components имеют метод resized(Rectangle<int> r). Данный метод принимает в качестве параметра объект типа Rectangle<int>. Данный объект описывает прямоугольную область и в данном случае определяет область экрана, в которой будут отображаться компоненты графического интерфейса конкретного эффекта. Подробнее поля и методы этих классов представлены ниже.

Octaver\_GUI\_Components(AudioProcessorEditor\* ae, ButtonListener\* bl, SliderListener\* sl);

void resized(Rectangle<int> r);

ScopedPointer<Label> MixLabel;

ScopedPointer<Label> OctaveLowLabel;

ScopedPointer<Label> OctaveHighLabel;

ScopedPointer<Slider> MixKnob;

ScopedPointer<Slider> OctaveLowKnob;

ScopedPointer<Slider> OctaveHighKnob;

ScopedPointer<TextButton> BypassButton;

ScopedPointer<GroupComponent> groupComponent;

Компоненты графического интерфейса пользователя для эффекта Octaver содержат ручки типа Slider для изменения соотношения исходного сигнала к обработанному, уровня звука на октаву ниже основного и уровня звука на октаву выше основного. Также присутствует названия типа Label для соответствующих ручек, поле groupComponent типа GroupComponent для очерчивания границы компонента Octaver, а также кнопка BypassButton, предназначенная для перевода данного эффекта в пассивное состояние, в котором он не влияет на звук.

Delay\_GUI\_Components(AudioProcessorEditor\* ae, ButtonListener\* bl, SliderListener\* sl);

void resized(Rectangle<int> r);

ScopedPointer<GroupComponent> groupComponent;

ScopedPointer<Label> timecodeDisplayLabel;

ScopedPointer<Label> DelayLabel;

ScopedPointer<Label> MixLabel;

ScopedPointer<Label> FeedbackLabel;

ScopedPointer<Slider> MixKnob;

ScopedPointer<Slider> FeedbackKnob;

ScopedPointer<Slider> DelayKnob;

ScopedPointer<TextButton> BypassButton;

ScopedPointer<TextButton> SynchButton;

ScopedPointer<TextButton> DottedButton;

ScopedPointer<TextButton> SecondDottedButton;

Компоненты графического интерфейса пользователя для эффекта Delay содержат ручки и метки, поясняющие их функции. Поле timecodeDisplayLabel предназначено для отображения текущей позиции воспроизведения дорожки, а также темпа, которые принимаются из host-программы. Поле FeedbackKnob – ручка, задающее параметр обратной связи в эффекте Delay, то есть она определяет количество копий исходного сигнала, которые будут воспроизведены, с интервалом времени между повторами, задаваемым ручкой DelayKnob. Кнопка BypassButton переводит компонент эффекта Delay в пассивное состояние. Кнопка SynchButton предназначена для перевода компонента эффекта Delay в синхронный режим. в котором время между повторами основывается на темпе музыкальной композиции, задаваемом host-программой. Кнопка DottedButton будет добавлять половину ко времени между повторами, а кнопка SecondDotted – четверть.

Reverb\_GUI\_Components(AudioProcessorEditor\* ae, ButtonListener\* bl, SliderListener\* sl);

void resized(Rectangle<int> r);

ScopedPointer<Label> DecayLabel;

ScopedPointer<Label> MixLabel;

ScopedPointer<Slider> DecayKnob;

ScopedPointer<Slider> MixKnob;

ScopedPointer<TextButton> BypassButton;

ScopedPointer<GroupComponent> groupComponent;

Данный класс имеет поля BypassButton, groupComponent, MixKnob, назначение которых описано выше для классов компонентов графического интерфейса других блоков эффектов. Поле DecayKnob представляет собой ручку, регулирующую затухание эмулируемых блоком эффекта Reverb отражений.

Класс ShimmerAudioProcessorEditor имеет следующие основные методы:

* + - ShimmerAudioProcessorEditor (ShimmerAudioProcessor&);
    - ~ShimmerAudioProcessorEditor();
    - void paint (Graphics&) override;
    - void resized() override;
    - void sliderValueChanged (Slider\* sliderThatWasMoved) override;
    - void buttonClicked (Button\* buttonThatWasClicked) override;
    - void timerCallback();
    - const String setSyncLabel(int sl);
    - void updateTimecodeDisplay (AudioPlayHead::CurrentPositionInfo pos);

В конструкторе ShimmerAudioProcessorEditor()вызываются конструкторы компонентов графического интерфейса, задаются их параметры визуализации.

В конструкторе также вызываются следующие функции:

* + - setSize (window\_width,window\_height);
    - startTimer(200);
    - background\_imageImageCache::getFromMemory( Images::background\_jpg, Images::background\_jpgSize).

Функция setSize() устанавливает размер окна в соответствии с константными значениями полей window\_width и window\_height.

Функция startTimer() начинает отсчёт таймера. В качестве параметра данная функция принимает время в миллисекундах. По истечении этого времени будет вызвана функция timerCallback().

Функция getFromMemory() класса ImageCache загружает фоновое изображение из файла в памяти, или возвращает его, если изображение уже кэшировано. Фоновое изображение программного модуля обработки звуковой дорожки в реальном времени определяется в классе Images и задаётся статическим константным массивом типа unsigned char (cм. рисунок 3.4).

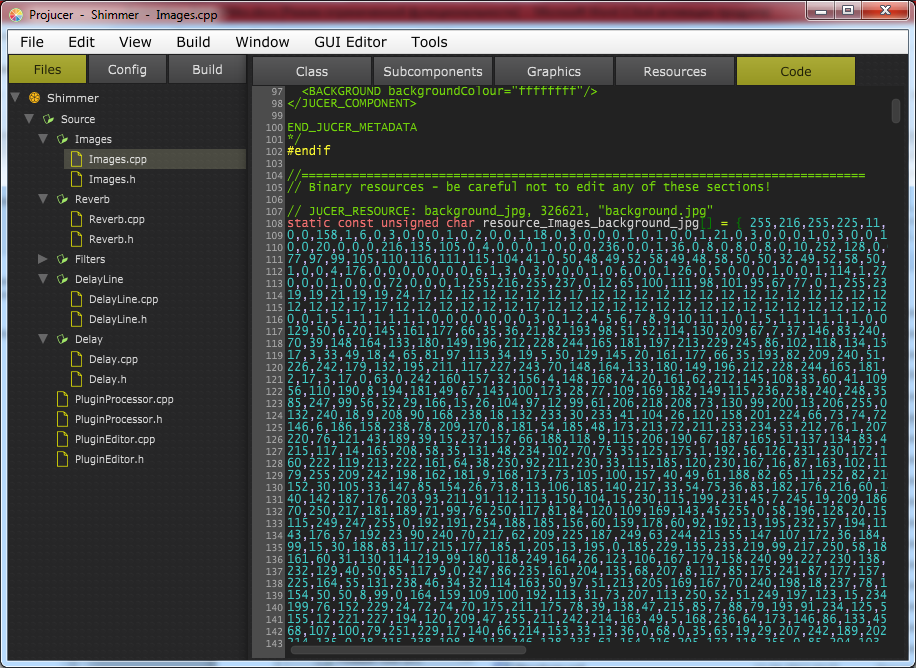


Рисунок 3.4 – Представление background\_image в файле Images.cpp

В конструкторе также задаются значения ручек в соответствии с параметрами блоков эффектов, которые находятся в объекте типа ShimmerAudioProcessor. Это осуществляется вызовом метода getParameter(), определённого в классе ShimmerAudioProcessor используя ссылку на объект processor данного типа.

Класс ShimmerAudioProcessor унаследован от следующих абстрактных классов (см. рисунок 3.5):

* AudioProcessorEditor;
* SliderListener;
* Timer;
* ButtonListener.



Рисунок 3.5 – Наследование класса ShimmerAudioProcessorEditor

Класс ShimmerAudioProcessorEditor является основным, как и класс ShimmerAudioProcessor, унаследованный от AudioProcessor. Два этих класса создаются автоматически при создании проекта и являются неотъемлемой его частью.

Основные виртуальные методы, которые включены в класс ShimmerAudioProcessor при создании проекта, и которые необходимо переопределить:

* void paint (Graphics&);
* void resized().

Метод paint(Graphics&) вызывается, когда область некоторого компонента графического интерфейса нуждается в перерисовке.

g.drawImage (background\_image, 0, 0, getWidth(), getHeight(), 200, 200, getWidth(), getHeight());

В данном методе вызывается функция drawImage(), прорисовывающая фоновое изображение программного модуля. В качестве параметров в неё передаётся ссылка на объект background\_image типа Image, в котором хранится информация об изображении, координаты левого верхнего уровня прямоугольника, в котором будет отображаться изображение, ширина и высота этого прямоугольника, координаты прямоугольника исходного изображения, его ширина и высота.

void ShimmerAudioProcessorEditor::resized()

{

int text\_height = 30;

int knob\_size = 80;

int gain\_area\_width = 100;

int octaver\_height = 175;

int delay\_height = 200;

int reverb\_height = 175;

Rectangle<int> window\_rectangle = getLocalBounds();

Rectangle<int> gain\_area =

window\_rectangle.removeFromLeft(gain\_area\_width);

Rectangle<int> octaver\_rectangle =

window\_rectangle.removeFromTop(octaver\_height);

Rectangle<int> delay\_rectangle =

window\_rectangle.removeFromTop(delay\_height);

Rectangle<int> reverb\_rectangle =

window\_rectangle.removeFromTop(reverb\_height);

octaver\_gui\_components.resized(octaver\_rectangle);

delay\_gui\_components.resized(delay\_rectangle);

reverb\_gui\_components.resized(reverb\_rectangle);

InputGainLabel->setBounds

(gain\_area.removeFromTop(text\_height));

InputGainKnob->setBounds

(gain\_area.removeFromTop(knob\_size));

OutputGainKnob->setBounds

(gain\_area.removeFromBottom(knob\_size));

OutputGainLabel->setBounds

(gain\_area.removeFromBottom(text\_height));

}

В методе resized() задаются используемые значения размеров компонентов графического интерфейса, вызываются методы resized() объектов компонентов графического интерфейса каждого эффекта. Также в данном методе задаются границы меток и ручек входного и выходного усилителей сигнала.

В фреймворке Juce обработка событий компонентов графического интерфейса реализована системой broadcaster-listener, которая соответствует поведенческому шаблону проектирования наблюдатель (на английском Observer). Данный шаблон создает механизм у класса, который позволяет получать экземпляру объекта этого класса оповещения от других объектов об изменении их состояния, тем самым наблюдая за ними. В конкретном случае компоненты графического интерфейса добавляют слушателей или наблюдателей с помощью функции AddListener(). Когда происходит некоторое событие, связанное с некоторым компонентом графического интерфейса, объект оповещает об этом своего слушателя. Слушатель, в свою очередь, ответственен за обработку события.

Данный класс также унаследован от абстрактных классов SliderListener и ButtonListener. Соответственно перегружены их методы sliderValueChanged (Slider\* sliderThatWasMoved) и buttonClicked (Button\* buttonThatWasClicked).

Функция sliderValueChanged (Slider\* sliderThatWasMoved) вызывается, когда пользователь изменяет положение ручки, а значит, и значение. Функция обработчик вызывается лишь в том случае, если до изменения пользователем положения ручки у соответствующего графического компонента была вызвана функция addListener(SliderListener \*), в которую был передан указатель на объект класса, унаследованного от SliderListener.

Функция buttonClicked (Button\* buttonThatWasClicked) вызывается когда пользователь нажимает на кнопку. Аналогично функция обработчик вызывается лишь в том случае, если до нажатия кнопки у соответствующего графического компонента была вызвана функция addListener(ButtonListener\*), в которую был передан указатель на объект класса, унаследованного от ButtonListener.

Класс ShimmerAudioProcessorEditor также унаследован от абстрактного класса Timer. Соответственно переопределена его виртуальная функция timerCallback(). Внутри вызывается функция updateTimecodeDisplay(), предназначенная для отображения на графическом интерфейсе обновлённой информации, полученной от host-программы.

## Блок настройки параметров обработки

*Блок настройки параметров обработки* реализован в классе ShimmerAudioProcessor (см. рисунок 3.6). Данный блок предназначен для преобразования ввода пользователя и хранения параметров, влияющих на работу блоков эффектов.

Пользовательские параметры хранятся в массиве UserParams[NumParams] типа float. Для понимания того, за что отвечает параметр, было определено перечисление Parameters. Перечисление (enum) – тип который может содержать значения указанные программистом.



Рисунок 3.6 – Организация блока настройки параметров обработки

NumParams – размер массива параметров обработки. В структуре Parameters это значение следует за названием номера последнего элементов, а значит равно количеству параметров.

enum Parameters{

DelayBypass = 0,

DelayTime,

DelayFeedback,

DelayMix,

DelaySynch,

DelayDot,

DelaySecondDot,

ReverbBypass,

ReverbMix,

ReverbDecay,

InputGain,

OutputGain,

NumParams

};

В массиве хранятся как положения ручек, которые представляют собой значения типа float, так и информация о состоянии кнопок, которая представляет собой значения типа bool. Принято решение, что если кнопка нажата, то значение true типа bool сохраняется в массиве параметров как значение 1.0f типа float. Значение false типа bool сохраняется в массиве как значение 0.0f типа float. Ещё одним параметром является поле hostBPM типа double, представляющее собой темп композиции. Темп задаётся host-программой. Поле lastPosInfo типа CurrentPositionInfo содержит информацию головки воспроизведения. С помощью этого объекта извлекается и темп композиции.

Массив параметров UserParams[NumParams] инициализируется значениями в конструкторе ShimmerAudioProcessor() после инициализации объектов классов, представляющих работу звуковых эффектов.

UserParams[DelayBypass] = delay.getByPass();

UserParams[DelayTime] = delay.getDelayTimeMS();

UserParams[DelayFeedback] = delay.getFeedback();

UserParams[DelayMix] = delay.getMix();

UserParams[DelaySynch] = 0.0f;

UserParams[DelayDot] = 0.0f;

UserParams[DelaySecondDot] = 0.0f;

UserParams[ReverbBypass] = reverb.getBypass();

UserParams[ReverbMix] = reverb.getMix();

UserParams[ReverbDecay] = reverb.getDecayFactor();

UserParams[InputGain] = 1.0f;

UserParams[OutputGain] = 1.0f;

//default host BPM to 120, default in most DAWs

hostBPM = 120;

AudioPlayHead::CurrentPositionInfo lastPosInfo;

Методы класса ShimmerAudioProcessor, работают как интерфейс между блоком интерфейса пользователя и блоками.

float getParameter (int index);

void setParameter (int index, float newValue);

float calculateDelayTap(float tap);

Функция getParameter() в качестве параметра принимает значение типа int, которое определяет индекс параметра, который будет возвращён вызывающей функции.

Функция setParameter() устанавливает новое значение элемента с номером index массива параметров UserParams[], выполняет обработку и преобразование входных данных во внутренние параметры программного модуля. Данная функция вызывается при обработке пользовательского ввода. Логика обработки входных данных зависит от устанавливаемого параметра. Например, пользователь устанавливает ручку Delay в новое значение. Вызывается обработчик события изменения значения ручки, в котором, в свою очередь, вызывается метод setParameter(). В ячейку памяти UserParams[DelayTime] заносится новое значение newValue и вызывается функция SetDelay(float d\_ms) объекта класса Delay, который представляет собой модуль эффекта Delay. Данная функция устанавливает значение задержки равным передаваемому параметру. Типичное взаимодействие пользователя и объектов программного модуля обработки звуковой дорожки с течением времени представлено на диаграмме последовательности (см. чертёж ГУИР.400201.161 РР.2).

Класс ShimmerAudioProcessor содержит в качестве полей объекты классов Delay и SReverb, которые соответственно описывают работу блоков эффектов Delay и Reverb.

## Класс DelayLine

Перед рассмотрением классов Delay и Reverb, необходимо описать класс DelayLine, так как он является их неотъемлемой частью: цифровые эффекты Delay и Reverb основаны на простой DelayLine (см. рисунок 3.7). Класс DelayLine представляет собой обёртку буфера значений типа float.

Поле buffer типа float\* – указатель на область памяти, где будут храниться задержанный сигнал. Как известно, оцифрованный звук представляет собой массив сэмплов. Для создания буфера задержки сигнала определённой длительности в миллисекундах, необходимо знать частоту цифрового сигнала. Поле sample\_rate типа int хранит значение частоты дискретизации сигнала. Типичными значениями данного поля являются числа 44100, 48000, 96000, 192000.

Поля MAX\_DELAY\_SAMPLES и MAX\_DELAY\_MS задают максимальную длину буфера задержки в сэмплах и миллисекундах соответственно. Поля delay\_ms и delay\_samples задают текущую длину буфера задержки в миллисекундах и сэмплах соответственно.



Рисунок 3.7 – Класс DelayLine и его непосредственные связи с другими классами

Позиция, с которой будет производиться следующая запись в буфер, задаётся полем writePos, а позиция, с которой будет производиться следующее чтение, задаётся полем readPos.

class DelayLine

{

public:

//constructor / destructor

DelayLine(float d\_ms\_max = 0.0f, int sr = 44100,

float d\_ms = 0.0f);

~DelayLine();

//getters

float getDelayTimeMS();

float getMaxDelayTimeMS();

//setters

void setDelayTimeMS(float time\_ms);

void setDelay(float d\_ms);

//business functions

float readDelay();

void writeDelay(float in);

float next(const float in);

void resetDelayLine();

void clearBuffer();

private:

int sample\_rate;

int writePos, readPos, MAX\_DELAY\_SAMPLES;

float delay\_ms, delay\_samples, fraction,

MAX\_DELAY\_MS;

float \*buffer;

};

## Блок эффекта «delay»

*Блок эффекта «delay»* представлен классом Delay. Данный класс представляет собой обёртку для класса DelayLine.

class Delay

{

public:

Delay(float d\_ms\_max = 0.0f, int sr = 44100, float

d\_ms = 800.0f, float feedback = 0.0f, float

mixLevel = 0.3f, bool bp = false);

~Delay();

float next(float in);

float getDelayTimeMS();

float getFeedback();

float getMix();

bool getByPass();

void setFeedback(float f\_pct);

void setMix(float m\_pct);

void setByPass(bool bp);

void setDelay(float d);

private:

DelayLine delay\_line;

float feedback, mixLevel;

bool delay\_bypass;

}

Класс Delay кроме поля delay\_line класса DelayLine содержит параметры, которые задаются и изменяются пользователем. Поле bypass типа bool определяет, в каком состоянии находится блок эффекта: пассивном или активном. Поле Feedback типа float отвечает за обратную связь эффекта Delay, то есть он определяет коэффициент ослабления каждой последующей копии сигнала. Поле mixLevel типа float отвечает за соотношение между исходным или «сухим» сигналом и обработанным эффектом Delay или «влажным» сигналом.

## Блок эффекта «reverb»

*Блок эффекта «reverb»* задан классом SReverb. Его структура отражена на рисунке 3.8. Цифровой эффект Reverb содержит следующие фильтры:

* + - всепропускающий фильтр;
    - гребенчатый фильтр;
    - фильтр нижних частот.

Соответственно эти фильтры реализованы в классах:

* + - AllPass;
    - Comb;
    - LowPass.

Класс SReverb кроме полей вышеуказанных классов содержит параметры, которые задаются и управляются пользователем. Поле decayFactor типа float определяет коэффициент затухания звуковых отражений. Поле mix типа float определяет соотношение необработанного сигнала к обработанному. Поле bypass типа bool определяет активен ли в данный момент времени эффект Reverb. Активным называется состояние эффекта, при котором он обрабатывает входной сигнал и подаёт его на выход. При значении bypass равном true, модуль эффекта Reverb не вносит изменений в сигнал.



Рисунок 3.8 – Класс SReverb и его компоненты

Класс SReverb имеет единственную бизнес-функцию, которая принимает очередной сэмпл цифрового сигнала и возвращает сэмпл обработанного сигнала.

class SReverb

{

public:

//constructor / destructor

SReverb(const int sr = 44100, const float rt60

= 3.0f, const float cDelay1 = 29.7, const float cDelay2 = 37.1, const float cDelay3 = 41.1, const float cDelay4 = 43.7, const float aDelay1 = 5.0, const float aDelay2 = 1.7, const float aGain1 = 0.707, const float aGain2 = 0.5, const float lCutoff1 = 2300.0f, const float mx = 0.3f); //const float lCutoff1 = 2300.0f

~SReverb();

//getters

float getDecayFactor();

float getCombDelay(const int id);

float getAllpassDelay(const int id);

float getAllpassGain(const int id);

float getLowpassCutoff(const int id);

bool getBypass();

float getMix();

//setters

void setDecayFactor(const float df);

void setCombDelay(const int id, const float sr,

const float d\_ms);

void setAllpassGain(const int id, const float

g);

void setAllpassDelay(const int id, const int

sr, const float d\_ms);

void setLowpassCutoff(const int id, const int

sr, const float cf\_hz);

void setBypass(bool bp);

void setMix(float value);

//business methods

float next(const float in);

private:

float decayFactor, ALLPASS\_GAIN\_LIMIT;/

float mix;

bool bypass;

Comb \*combs[NUM\_COMBS];

Allpass \*allpasses[NUM\_ALLPASSES];

Lowpass \*lowpasses[NUM\_LOWPASSES];

};

В конструкторе класса SReverb, инициализируются многочисленные буферы DelayLine, содержащиеся в фильтрах Comb и AllPass. Параметры гребенчатых и всепропускающих фильтров установлены экспериментальным путём, отталкиваясь от значений, указанных на рисунке 3.9. Переменная rt60 является стандартным измерением времени реверберации, и определяется как время, необходимое для уменьшения звукового давления на 60 dB, с момента внезапного окончания генерируемого тестового сигнала.

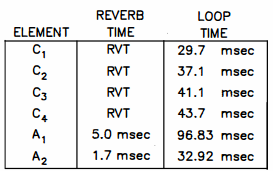


Рисунок 3.9 – Значения параметров реверберации для симуляции концертного зала среднего размера

## Фильтры

Всепропускающий фильтр представлен классом AllPass.

class Allpass{

public:

//constructor

Allpass(const int sr, const float d\_ms, const

float d\_ms\_max, const float g);

~Allpass();

//getters

float getGain();

float getDelayTimeMS();

//setters

void setGain(const float g);

void setDelayTimeMS(float time\_ms);

//business methods

float next(const float in);

private:

float gain;

DelayLine \*delay;

};

Работа данного фильтра основывается на DelayLine. Данный фильтр имеет единственную бизнес-функцию next(float in), которая принимает на свой вход очередной сэмпл сигнала и возвращает на выход сэмпл типа float обработанного сигнала. Кроме приватного поля delay\*, класс содержит поле gain типа float. Поле gain отвечает за усиления сигнала внутри цепочки фильтра (см. рисунок 3.10). Данный параметр рассчитывается в классе Reverb на основе времени реверберации и времени циклической задержки.

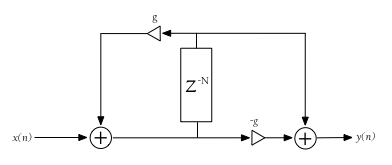


Рисунок 3.10 – Структура всепропускающего фильтра

class Comb{

public:

//constructor

Comb(const int sr, const float d\_ms, const float d\_ms\_max, const float g);

//destructor

~Comb();

//getters

float getGain();

float getDelayTimeMS();

//setters

void setGain(const float g);

void setDelayTimeMS(float time\_ms);

//business methods

float next(const float in);

private:

float gain;

DelayLine \*delay;

Lowpass \*lpFilter;

};

Гребенчатый фильтр имеет такой же интерфейс, как и всепропускающий фильтр: конструктор с таким же составом параметров, функции для получения и установки значений private полей, бизнесс-функцию next(const float in). Однако внутреннее устройство фильтров отличается и соответствует структуре, указанной на рисунке 3.11.

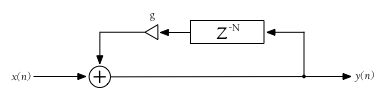


Рисунок 3.11 – Структура гребенчатого фильтра

Фильтр нижних частот реализован довольно простым классом Lowpass. В качестве параметров конструктор этого класса принимает частоту дискретизации и частоту среза в герцах. Как и вышеописанные модули имеет одну бизнес-функцию next(float in),которая возвращает сэмпл отфильтрованного сигнала.

class Lowpass{

public:

//constructor

Lowpass(const int sr, const float cf\_hz);

//getters

float getCutoff();

//setters

void setCutoff(const int sr, const float cf\_hz);

//business methods

float next(const float in);

private:

float cutoff, coef, prev;

};

## Блоки преобразования входного и выходного сигналов

Блоки преобразования входного и выходного сигналов представляют собой ручки усиления уровня соответствующих сигналов. Данные блоки довольно просты и реализуется в рамках функции processBlock() класса ShimmerAudioProcessor. Входной сигнал канала умножается на параметр в ячейке InputGain, а выходной сигнал с блока эффекта Reverb умножается на параметр в ячейке OutputGain.

channelDataLeft[i] = inChannelDataLeft[i] \* UserParams[InputGain];

channelDataLeft[i] = out\_l \* UserParams[OutputGain];

# РАЗРАБОТКА ПРОГРАММНЫХ МОДУЛЕЙ

## Функции класса ShimmerAudioProcessorEditor

В данном разделе более детально рассмотрена реализация функций разрабатываемого программного модуля обработки звуковой дорожки в реальном времени. Функции рассматриваются в порядке их вызова после взаимодействия пользователя с интерфейсом.



Рисунок 4.1 – Обработка нажатия кнопки SynchButton

Функция sliderValueChanged(Slider\* sliderThatWasMoved) вызывается, когда пользователь ввёл значение параметра с помощью ручки. В данной функции параметр sliderThatWasMoved сравнивается с указателями компонентов графического модуля. Когда стало известно, значение какой ручки было изменено пользователем, с помощью ссылки на объект processor класса ShimmerAudioProcessor вызывается функция setParameterNotifyingHost() с параметрами индекса изменённого параметра и его новым значением. Данная функция отправляет сообщение host-программе об изменении значения внутреннего параметра и вызывает функцию setParaneter() с теми же параметрами.

Функция buttonClicked (Button\* buttonThatWasClicked)вызывается, когда пользователь нажал на некоторую кнопку, отображённую на экране. В данной функции параметр buttonThatWasClicked сравнивается с указателями компонентов графического модуля и вызывается функция setParameterNotifyingHost(), в которую передаются в качестве аргументов индекс параметра и его новое значение. Обработка кнопок BypassButton каждого модуля эффектов, а также кнопок DottedButton и SecondDottedButton ограничивается этими действиями.

Обработка нажатия кнопки SynchButton является более сложной (см. рисунок 4.1). Если пользователь нажал на кнопку SynchButton и её состояние теперь «нажата», то модуль эффекта Delay переходит в синхронизированный с темпом режим. На экран отображаются невидимые в несинхронизированном режиме кнопки DottedButton и SecondDottedButton. Если состояние кнопки SynchButton «отжата», то вышеуказанные кнопки DottedButton и SecondDottedButton становятся невидимыми для пользователя, а значит, он не может изменить их состояние.

Отличие интерфейса в двух режимах не ограничивается только присутствием или отсутствием кнопок DottedButton и SecondDottedButton. Если устанавливается синхронный режим, то название ручки Delay меняется на Denominator. Также меняется её назначение и диапазон задаваемых значений. В синхронном режиме ручкой с названием Denominator задаётся делитель темпа.

## Функции класса ShimmerAudioProcessor

Функция setParameter (int index, float newValue) предоставляется фреймворком Juce и призвана устанавливать параметры обработки программного модуля обработки звука в новые значения. Подразумевается, что параметром index указывается, какой именно параметр необходимо обновить, то есть заменить установить его в значение, переданное параметром newValue.

Реализация этой функции в типичной ситуации записывает значение newValue в ячейку массива UserParams[NumParams] с индексом index и устанавливает новое значение параметра соответствующего модуля эффекта с помощью вызова функции рода setter. (см. рисунок 4.2).



Рисунок 4.2 – Типичная схема установки параметра эффекта

Установка нового значения параметра модуля эффекта Delay предполагает обработку входного параметра. Внутри тела функции setParameter (int index, float newValue)объявлена статическая переменная delay, которая предназначена для хранения заданного пользователем делителя темпа в несинхронизированном режиме или времени задержки между копиями сигнала в миллисекундах в синхронном режиме.

При установке параметра DelayTime происходит проверка, в каком режиме находится модуль эффекта Delay в данный момент времени (см. рисунок 4.3).

case DelayTime:

UserParams[DelayTime] = newValue;

if(UserParams[DelaySynch] == 1.0f)

{

delay.setDelay( calculateDelayTap(UserParams[DelayTime]));

}

else

{

delay.setDelay(UserParams[DelayTime]);

}

break;

В синхронном режиме в функцию setDelay() передаётся не значение UserParams[DelayTime], а значение, возвращённое функцией calculateDelayTap(UserParams[DelayTime]). Функция calculateDelayTap()описана ниже.



Рисунок 4.3 – Изменение параметра DelayTime

При установке параметра DelaySynch новое значение заносится в массив параметров в ячейку UserParams [DelaySynch]. Далее с помощью функции swap(UserParams[DelayTime], old\_delay) соответствующее значение, хранящееся в массиве UserParams[DelayTime], меняется местами со значением статической переменной old\_delay, определённой в теле функции (см. рисунок 4.4).



Рисунок 4.4 – Изменение параметра DelaySynch

Функция-шаблон swap меняет значения параметров одного и того же типа между собой.

template <class T>

void swap(T& a, T& b)

{

T temp;

temp = a;

a = b;

b = temp;

}

Если текущий режим синхронизированный, вызывается функция calculateDelayTap(UserParams[DelayTime]) для подсчёта времени задержки в миллисекундах.

При установке значений массива параметров с индексами DelayDot и DelaySecondDot также подсчитывается время задержки в миллисекундах с помощью функции calculateDelayTap(UserParams[DelayTime]) и передаётся модулю эффекта Delay функцией setDelay().

float ShimmerAudioProcessor::calculateDelayTap( float tp)

{

float result = .0f;

float temp = .0f;

result = temp = 60\*4\*1000/(hostBPM\*(tp));

if(UserParams[DelayDot])

result += temp/2;

if(UserParams[DelaySecondDot])

result += temp/4;

return result;

}

Функция calculateDelayTap(float tp) возвращает значение в миллисекундах времени задержки между повторами сигнала. Преобразование входного параметра, который представляет собой делитель темпа, осуществляется с использованием текущего значения темпа с использованием формулы 4.1:

|  |  |  |
| --- | --- | --- |
|  |  | (4.1) |

где ** – время между повторами, мс;

 – количество миллисекунд в минуте, равное 60000;

4 – количество четвёртых нот в целой ноте;

 – количество четвёртых нот в минуте;

 – заданный пользователем делитель времени.

В числителе дроби присутствует множитель 4, так как темп обычно задаётся в четвертях.

Алгоритм функции включает следующие шаги:

1. Создать переменную result типа float.
2. Создать переменную temp типа float.
3. Присвоить переменным result и temp значение, вычисленное по формуле 60\*4\*1000/(hostBPM\*tp).
4. Если задан параметр DelayDot, прибавить к значению переменной result половину значения переменной temp.
5. Если задан параметр DelaySecondDot, прибавить значению переменной result четверть значения переменной temp.
6. Вернуть значение переменной result.

Обработка звука изначально закреплена за классом ShimmerAudioProcessor. Данный класс имеет две функции, в которых происходит непосредственная обработка аудиоинформации:

* processBlock ();
* processBlockBypassed ().

В одно и то же время функционирует лишь один из указанных выше методов. Функция processBlockBypassed () вызывается тогда, когда весь плагин находится в режиме Bypass, а функция processBlock (), когда плагин активен.

void ShimmerAudioProcessor::processBlock

(AudioSampleBuffer& buffer, MidiBuffer& midiMessages)

{

int numSamples = buffer.getNumSamples(); //THIS IS NUM SAMPLES PER CHANNEL

float\* channelDataLeft = buffer.getWritePointer(0);

float\* channelDataRight =

buffer.getWritePointer(1);

const float\* inChannelDataLeft =

buffer.getReadPointer(0);

const float\* inChannelDataRight =

buffer.getReadPointer(1);

float out\_l = 0.0f, out\_r = 0.0f;

for(int i = 0; i < numSamples; i++)

{

out\_l = delay.next(channelDataLeft[i]);

out\_l = reverb.next(out\_l);

channelDataLeft[i] = out\_l \*

UserParams[OutputGain];

channelDataRight[i] = channelDataLeft[i];

}

for (int i = getNumInputChannels(); i <

getNumOutputChannels(); ++i)

{

buffer.clear (i, 0,

buffer.getNumSamples());

}

//get the host BPM and sync playhead to it

updateCurrentTimeInfoFromHost();

}

Параметрами функции processBlock() являются ссылка на буфер типа AudioSampleBuffer и ссылка на midi-буфер типа MidiBuffer.

Вызовом функции buffer.getNumSamples() в переменную numSamples сохраняется количество сэмплов в канале. Функция buffer.getWritePointer(0) возвращает указатель типа float содержащий сэмплы сигнала левого канала. Соответственно buffer.getWritePointer(1) возвращает указатель типа float содержащий сэмплы сигнала правого канала. Функция buffer.getReadPointer() возвращает указатель const float\*, и используется для чтения сэмплов входного сигнала. В цикле каждый сэмпл входного сигнала подаётся на блок эффекта Delay вызовом функции next().Возвращаемое значение подаётся на следующий блок – блок эффекта Reverb с помощью вызова одноимённой функции.

Далее, возвращённое функцией next() объекта reverb значение записывается в соответствующие позиции буферов левого и правого выходных каналов. Если количество выходных каналов оказалось больше числа входных, в цикле очищаются избыточные буферы. Такая ситуация может возникнуть, например, если в host-программе была создана звуковая дорожка конфигурации mono. Тогда количество входных каналов будет равным 1, а выходных – 2. Это значит, что второй буфер необходимо очистить.

В конце метода вызывается функция updateCurrentTimeInfoFromHost(), которая обновляет информацию о текущем времени, получая её от host-программы.

void ShimmerAudioProcessor::

updateCurrentTimeInfoFromHost()

{

if (AudioPlayHead\* ph = getPlayHead())

{

AudioPlayHead::CurrentPositionInfo newTime;

if (ph->getCurrentPosition (newTime))

{

lastPosInfo = newTime;

/\* Successfully got the current time from the host..\*/

}

}

/\* If the host fails to provide the current time, we'll just reset our copy to a default..\*/

lastPosInfo.resetToDefault();

}

Функция getPlayHead() класса ShimmerAudioProcessor возвращает объект класса AudioPlayHead, который используется для выяснения состояния и позиции головки для считывания. Эта функция может быть вызвана только из метода processBlock(). В иных случаях вызов этого метода может привести к неопределённому поведению. Полученный объект может быть использован для получения информации с начала обработки блока сэмплов. Если host-программа не может обеспечить запрашиваемой информацией, то функция getPlayHead()вернёт nullptr. В этом случае объект lastPosInfo инициализируется стандартными величинами, указанными ниже.

void AudioPlayHead::

CurrentPositionInfo::resetToDefault()

{

zerostruct (\*this);

timeSigNumerator = 4;

timeSigDenominator = 4;

bpm = 120;

}

В следующих разделах подробно описывается непосредственная обработка звука модулями эффектов Delay и Reverb. Алгоритм обработки звука отражён на плакате ГУИР.400201.161 ПЛ.2.

## Функции класса Delay

Класс Delay включает в себя конструктор, деструктор, сеттеры и геттеры. Но основной является бизнес функция next(const float in).

float Delay::next(const float in)

{

if(delay\_bypass)

return in;

float value\_to\_read = delay\_line.readDelay();

delay\_line.writeDelay(in + feedback\*value\_to\_read);

return mixLevel \* value\_to\_read + (1-mixLevel) \* in;

}

Данная функция получает сэмпл сигнала и возвращает обработанный сэмпл. Если модуль эффекта находится в состоянии Bypass, то функция возвращает исходный сэмпл сигнала. Иначе производится вызов функции readDelay() объекта delay\_line класса DelayLine, которая возвращает значение типа float. Фактически, эта функция производит чтение сэмпла из буфера задержки и инкрементирует позицию следующего чтения.

float DelayLine::readDelay(){

float yn = buffer[readPos];

int readPos\_minus1 = readPos - 1;

if(readPos\_minus1 < 0)

readPos\_minus1 = MAX\_DELAY\_SAMPLES - 1; //MAX\_DELAY\_SAMPLES - 1 is the last location of the buffer

//get y(n-1)

float yn\_minus1 = buffer[readPos\_minus1];

readPos++;

if(readPos >= MAX\_DELAY\_SAMPLES)

readPos = 0;

//perform linear interpolation of : (0,yn) and (1,yn\_minus1) by the ammount of fractional delay(fraction)

return linInterp(0, 1, yn, yn\_minus1, fraction);

}

Далее с помощью функции writeDelay(float in) в буфер, находящийся в классе DelayLine производится запись суммы сэмпла входного сигнала и произведения параметра feedback на значение, полученное с помощью функции readDelay(). Возвращаемое значение зависит от поля mix класса Delay. Поле mix определяет отношение в возвращаемом значении исходного сэмпла и сэмпла, прочитанного из буфера задержки.

void DelayLine::writeDelay(float in)

{

buffer[writePos] = in;

writePos++;

if(writePos >= MAX\_DELAY\_SAMPLES)

writePos = 0;

}

## Функции класса SReverb

В конструкторе SReverb инициализируются, фильтров, из которых состоит модуль эффекта Reverb табличными значениями. Среди таких параметров – время задержки дилэй-линии, частота фильтра нижних частот, параметр gain, отвечающий за постепенное затухание сигнала. Также инициализируются параметры эффекта decayFactor, mix, непосредственно управляемые пользователем.

SReverb::SReverb(const int sr, const float rt60,

const float cDelay1, const float cDelay2, const float cDelay3, const float cDelay4,

const float aDelay1, const float aDelay2, const float aGain1, const float aGain2, const float lCutoff1, const float mx){

ALLPASS\_GAIN\_LIMIT = 0.107f; //0.707f;

decayFactor = rt60;

float d\_ms, d\_ms\_max = 100.0f, gain;

d\_ms = cDelay1;

bypass = false;

mix = mx;

gain = calcCombGain(d\_ms, decayFactor);

combs[0] = new Comb(sr, d\_ms, d\_ms\_max, gain);

setCombDelay(0,sr,d\_ms);

d\_ms = cDelay2;

gain = calcCombGain(d\_ms, decayFactor);

combs[1] = new Comb(sr, d\_ms, d\_ms\_max, gain);

setCombDelay(1,sr,d\_ms);

d\_ms = cDelay3;

gain = calcCombGain(d\_ms, decayFactor);

combs[2] = new Comb(sr, d\_ms, d\_ms\_max, gain);

setCombDelay(2,sr,d\_ms);

d\_ms = cDelay4;

gain = calcCombGain(d\_ms, decayFactor);

combs[3] = new Comb(sr, d\_ms, d\_ms\_max, gain);

setCombDelay(3,sr,d\_ms);

d\_ms\_max = 20.0f;

allpasses[0] = new Allpass(sr, aDelay1,

d\_ms\_max, aGain1);

allpasses[1] = new Allpass(sr, aDelay2,

d\_ms\_max, aGain2);

lowpasses[0] = new Lowpass(sr, lCutoff1);

}

Функция calcCombGain() рассчитывает на основе времени повторов и времени затухания реверберации значение gain, которое потом передаётся в качестве параметра в конструктор гребенчатого фильтра вместе с максимальным и действующим временем задержи. Так как параметр rt60 определяется как время, необходимое для уменьшения звукового давления на 60 dB, с момента внезапного окончания генерируемого тестового сигнала, то справедлива формула 4.2:

|  |  |  |
| --- | --- | --- |
|  |  | (4.2) |

где ** – изменение уровня сигнала, dB;

 – множитель десятичного логарифма;

 – звуковое давление в начальный момент времени;

– звуковое давление в конечный момент времени.

inline float calcCombGain(const float d\_ms, const float rt60){

return pow(10.0, ((-3.0 \* d\_ms) / (rt60 \* 1000.0)));

}

Основным методом класса SReverb является бизнес-функция next(const float in), которая принимает сэмпл входного сигнала. Если модуль эффекта Reverb находится в состоянии Bypass, то функция возвращает исходный сэмпл сигнала без изменений.

float SReverb::next(const float in){

if(bypass)

return in;

float out = 0.0f;

for(int i = 0; i < NUM\_COMBS; i++){

out += combs[i]->next(in \* 0.125f); //scale down to avoid clipping

}

float passOut = 0.0f;

float passOut2 = 0.0f;

passOut = allpasses[0]->next(out);

passOut2 = allpasses[1]->next(passOut);

return mix \* passOut2 \* NUM\_COMBS + (1.0f - mix) \* in;

//return passOut2;

}

В цикле для каждого гребенчатого фильтра вызывается его бизнес-функция next(), принимающая в качестве параметра сэмпл входного сигнала умноженный на коэффициент масштабирования, равный 0.125f . Возвращённые функцией значения суммируются в переменную out и подаются на первый всепропускающий фильтр с помощью вызова аналогичной функции. Значение, полученное с выхода первого всепропускающего фильтра, подаётся на второй.

float Comb::next(const float in){

float dL = delay->readDelay();

float lpRetVal = lpFilter->next(dL);

float dLW = in + lpRetVal\*gain;

delay->writeDelay(dLW);

return dL;

}

Алгоритм метода next(float in) класса Comb:

1. Прочитать содержимое буфера задержки.
2. Подать полученное значение на фильтр низких частот.
3. Полученное значение умножить на параметр gain, определяющий коэффициент затухания, и сложить с входным параметром метода.
4. Записать в буфер задержки полученное значение.
5. Вернуть значение, полученное в пункте 1.

float Lowpass::next(const float in){

prev = in \* (1 + coef) - (prev \* coef);

return prev;

}

Алгоритм метода next(float in) класса Allpass:

1. Прочитать содержимое буфера задержки.
2. Полученное значение умножить на параметр gain и сложить с входным параметром метода.
3. Записать полученное значение в буфер задержки.
4. Умножить значение на противоположное параметру gain значение.
5. Вернуть сумму произведения и полученного в пункте 1 значения.

float Allpass::next(const float in){

float dL = delay->readDelay();

float fW = in + (gain\*dL);

delay->writeDelay(fW);

float out = -gain\*fW + dL;

return out;

}

В результате, значение сэмпла с выхода последнего фильтра цепочки смешивается со значением исходного сигнала в пропорции, задаваемой параметром mix, управляемым пользователем, и возвращается в функцию processBlock() класса ShimmerAudioProcessor, где записывается в выходные буферы левого и правого каналов.

# ПРОГРАММА И МЕТОДИКА ИСПЫТАНИЙ

Тестирование является одним из важнейших этапов разработки программного обеспечения. Исключением не являются и программные модули обработки звука. Тестирование позволяет выявить проблемы и ситуации, в которых поведение программного модуля является неправильным ещё на стадии разработки. Таким образом, проводя тестирование, разработчики убеждаются в исправности написанного кода, а также в соответствии поведения программного продукта требованиям заказчика.

Также тестирование позволяет продемонстрировать заказчикам, что программный продукт завершён и соответствует требованиям, а значит, готов к реализации на рынке.

Тестирование программного модуля обработки звуковой дорожки в реальном времени проводилось в операционных системах Windows и Mac в следующих цифровых звуковых рабочих станциях:

* Cubase 7 Elements;
* Pro Tools;
* Apple Logic Pro X;
* Ableton Live 7.

При тестировании программного модуля обработки звуковой дорожки в реальном времени использовался метод чёрного ящика. Данный метод предполагает тестирование поведения объекта с точки зрения внешнего мира, то есть подразумевается отсутствие знаний о внутреннем устройстве программного модуля.

При разработке программного модуля обработки звуковой дорожки в реальном времени был использованы приёмы тестирования, называемые «эквивалентное разбиение», «анализ граничных значений» и «предположение об ошибке».

Метод «предположение об ошибке» в значительной степени основан на интуиции опытного разработчика. Идея заключается в том, что составляется список, включающий все ошибки и исключительные ситуации, которые могут возникнуть. Далее на основе этого списка создаются тесты.

Граничные условия – это ситуации, возникающие на высших и нижних границах входных классов эквивалентности. В файле исходных текстов DelayLine.cpp присутствует следующий код:

assert(time\_ms <= MAX\_DELAY\_MS);//check bound on delay time

Функция assert() подключается заголовочным файлом <cassert>. Она оценивает выражение, которое передается ей в качестве аргумента. Если аргумент-выражение этого макроса в функциональной форме равно нулю (т.е. выражение ложно), сообщение записывается на стандартное устройство вывода ошибок и вызывается функция abort(), работа программы прекращается. При отладке программы, мы можем неограниченное количество раз вызывать assert(). После отладки программы, не будет надобности в вызванных нами функциях assert(). Чтобы отключить все функции assert() в языке С++ определён макрос NDEBUG. То есть, необходимо включить строку #define NDEBUG в код программы.

В случае, указанном выше, оценивается верхнее граничное значение размера буфера задержки в методе setDelayTimeMS() и конструкторе класса DelayLine. В ходе разработки модуля эффекта Delay программного модуля обработки звуковой дорожки в реальном времени данное выражение позволило обнаружить ошибку, которая возникла при добавлении кода, обрабатывающего нажатие кнопок Dotted и Second Dotted, которые увеличивают время между повторами в 1,5 и в 1,25 раз соответственно. Отсутствие данной проверки привело бы к обращению к области памяти, не выделенной программой. Это бы могло привести к непредсказуемым последствиям.

Use Case (от английского – вариант использования) — это сценарная техника описания взаимодействия. С помощью Use Case может быть описано и пользовательское требование, и требование к взаимодействию систем, и описание взаимодействия людей и компаний в реальной жизни.

В разработке ПО эту технику часто применяют для проектирования и описания взаимодействия пользователя и системы, поэтому название Use Case часто воспринимает как синоним требования пользователя к решению определенной задачи в системе.

Тестовый случай (Test Case) – это артефакт, описывающий совокупность шагов, конкретных условий и параметров, необходимых для проверки реализации тестируемой функции или её части. Тестовый случай имеет следующую структуру:

* действие;
* ожидаемый результат;
* результат теста.

Каждый тест кейс должен иметь 3 части:

* + - cписок действий, которые приводят систему к состоянию, пригодному для проведения основной проверки, или условий, выполнение которых говорит о том, что система находится в пригодном для проведения основного теста состояния;
    - cписок действий, переводящих систему из одного состояния в другое, для получения результата, на основании которого можно сделать вывод о удовлетворении реализации, поставленным требованиям;
    - cписок действий, переводящих систему в первоначальное состояние (состояние до проведения теста - initial state).

Таблица 5.1 – Основные тесты программного модуля

|  |  |  |
| --- | --- | --- |
| Требование | Тест | Тест пройден |
| 1 | 2 | 3 |
| Настраиваемое время задержки должно соответство-вать отобра-жаемому на экране. | 1. Убедиться, что компонент находится в активном состоянии в режиме, не синхронизированным с темпом host-программы (отсутствуют кнопки Dotted и Second Dotted). 2. Установить значение ручки Delay равным 400, ручки Feedback – 1, ручки Mix – 0,5. 3. Воспроизвести звук щелчка. 4. Измерить время от оригинального звука до 10 повтора. 5. Разделить время на количество повторов и убедиться, что получившееся значение равно установленному ручкой. | да |
| При нажатии на кнопку Synchronize интерфейс должен измениться. | 1. Убедиться, что компонент находится в активном состоянии в режиме, не синхронизированным с темпом host-программы (отсутствуют кнопки Dotted и Second Dotted). 2. Нажать на кнопку Synchronize; 3. Убедиться, что на экране появились дополнительные кнопки Dotted и Second Dotted,название ручки Delay изменилось на Denominator. 4. Убедиться, что диапазон задаваемых значений этой ручкой изменился с 0..2000 на 0..5. | да |
| Кнопки Bypass должны отключать соответствующие им модули обработки. | 1. Убедиться, что компоненты находятся в активном режиме, ручки Mix, Delay и Decay установлены в ненулевые значения. 2. Воспроизвести звук щелчка. 3. Нажать на кнопку Bypass компонента Delay. 4. Убедиться, что после нажатия кнопки Bypass компонент необрабатывает звук. 5. Снова нажать на кнопку Bypass компонента. 6. Повторить все вышеперечисленные действия для компонента Reverb. | да |

*Продолжение таблицы 5.1*

|  |  |  |
| --- | --- | --- |
| 1 | 2 | 3 |
| Ручки Mix должны изменять соотношение между необработанным сигналом и обработанным. | 1. Установить значения всех ручек Mix в ноль. 2. Воспроизвести звук щелчка. 3. Подождать 5 секунд и убедиться, что уровень выходного сигнала равен нулю. 4. Установить значения всех ручек Mix в значения, отличные от нуля. 5. Убедиться, что звук обрабатывается модулями эффектов. | да |
| Кнопка Dotted должна добавлять 1/2 от времени между повторами, а кнопка Second Dotted должна добавлять 1/4 от времени между повторами. | 1. Убедиться, что компонент находится в активном состоянии в режиме, синхронизированным с темпом host-программы (присутствуют кнопки Dotted и Second Dotted). 2. Нажать на кнопку Dotted. 3. Установить значение ручки Delay равным 4, а ручки Feedback – 100. 4. Воспроизвести звук щелчка. 5. Засечь время от оригинального звука до 10 повтора. 6. Разделить время на количество повторов. 7. Убедиться, что полученное значение равно 2/3 от темпа host-программы. 8. Нажать на кнопку Dotted, а затем на кнопку Second Dotted. 9. Воспроизвести звук щелчка. 10. Засечь время от оригинального звука до 20 повтора. 11. Разделить время на количество повторов. 12. Убедиться, что полученное значение равно 4/5 от темпа host-программы. | да |

В таблице 5.1 представлены некоторые тесты, основанные на пользовательских требованиях, которые были пройдены программным модулем обработки звуковой дорожки в реальном времени. Для начала тестирования программного модуля обработки звуковой дорожки в реальном времени необходимо в цифровой рабочей звуковой станции создать проект с одной моно-дорожкой, на которой будет записан кратковременный звук щелчка. Во вкладке Inserts дорожки нажать на пустой слот, выбрать плагин Shimmer и убедиться, что отобразилось его окно. Если окно не появилось, необходимо щёлкнуть мышкой по появившемуся названию Shimmer во вкладке Inserts.

В таблице 5.1 указана лишь доля реально проводимых тестов. Более детально проверялось функционирование ручек Input Gain и Output Gain, Mix, Decay, а также работа неявных для пользователя фильтров. Для тестирования их работы был использован программный модуль компании Spectrum Analysis Voxengo SPAN PLUS. Данный модуль имеет спектральный анализатор, измеритель пикового и действующих значений уровня сигнала, детектор превышения сигналом уровня 0 dBFs.

Таким образом, проанализировав результаты испытаний и тестирования, можно сделать вывод, что модуль обработки звуковой дорожки в реальном времени является надёжным и работоспособным. Программный модуль точно выполняет свою функцию в соответствии с параметрами, задаваемыми пользователем.

# РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

## Системные требования

Требования к аппаратному обеспечению в большей степени диктуются используемой DAW, нежели разработанным программным модулем. Требования к программному обеспечению зависят от используемой DAW. В качестве примера в таблице 6.1 приведены системные требования новой цифровой звуковой рабочей станции фирмы Steinberg Cubase Pro 9.

Формат программного модуля – VST. Данный формат широко распространён и поддерживается всеми известными DAW: от популярных и функциональных Cubase и Pro Tools, до малоизвестной и легковесной TunaFish.

Таблица 6.1 – Системные требования Cubase Pro 9

|  |  |
| --- | --- |
| Mac OS X | Windows |
| OS X 10.11 / macOS Sierra | 64-Bit Windows 7 / 8.x / 10 |
| Многоядерный процессор 64-bit Intel или AMD (рекомендован Intel i5 или более быстрый) | |
| 4 GB ОЗУ (рекомендовано 8 GB или больше) | |
| 18 GB свободного места на жёстком диске | |
| разрешение экрана 1366 x 768 (рекомендовано 1920 x 1080) | |
| Графическая карта с DirectX 10 и поддержкой WDDM 1.1 (для ОС Windows) | |
| USB порт для USB-eLicenser (управление лицензиями) | |
| Аудиоинтерфейс, поддерживаемый ОС (для высокой производительности, низких задержек в звуке рекомендуется поддержка ASIO) | |
| Соединение с интернетом (для активации, регистрации аккаунта и продукта, дополнительных загрузок в ходе инсталляции) | |

Файл разработанного модуля обработки звука Shimmer.dll имеет размер всего 5 MiB.

## Процесс инсталляции

Процесс инсталляции программного модуля обработки звуковой дорожки в реальном времени указан для DAW Cubase 7 Elements на операционной системе Windows 7 Service Pack 1. Для других операционных систем и цифровых звуковых рабочих станций процесс инсталляции может отличаться, но основные этапы будут такими же.

Алгоритм добавления и использования разработанного плагина в Cubase следующий:



Рисунок 6.1 – Создание проекта в Cubase 7 Elements

1. Скопировать с диска файл Shimmer.dll в папку с VST плагинами на вашем компьютере. В данном случае была выбрана папка D:\vst\_plugins\_folder.
2. Открыть цифровую звуковую рабочую станцию (2 раза кликнуть на исполняемый файл Cubase LE AI Elements 7.exe) и создать в ней проект. Для этого в меню More (см. рисунок 6.1) выбрать пункт Empty, указать папку, в которой будет создан проект, нажать Create.



Рисунок 6.2 – Открытие меню Plug-in Information

1. Открыть меню с информацией о плагинах. В строке меню выбрать пункт Devices, в выпадающем списке выбрать Plug-in Information (см. рисунок 6.2).
2. Указать путь к папке, в которой находится файл Shimmer.dll. Во вкладке VST Plug-ins нажать кнопку VST 2.x Plug-in Path. В появившемся окне нажать кнопку Add (см. рисунок 6.3). В появившемся окне выбрать местоположение папки и нажать OK. Снова нажать OK. Нажать кнопку Update. Как видно на рисунке 6.4, в списке доступных плагинов появился файл с именем Shimmer.dll.

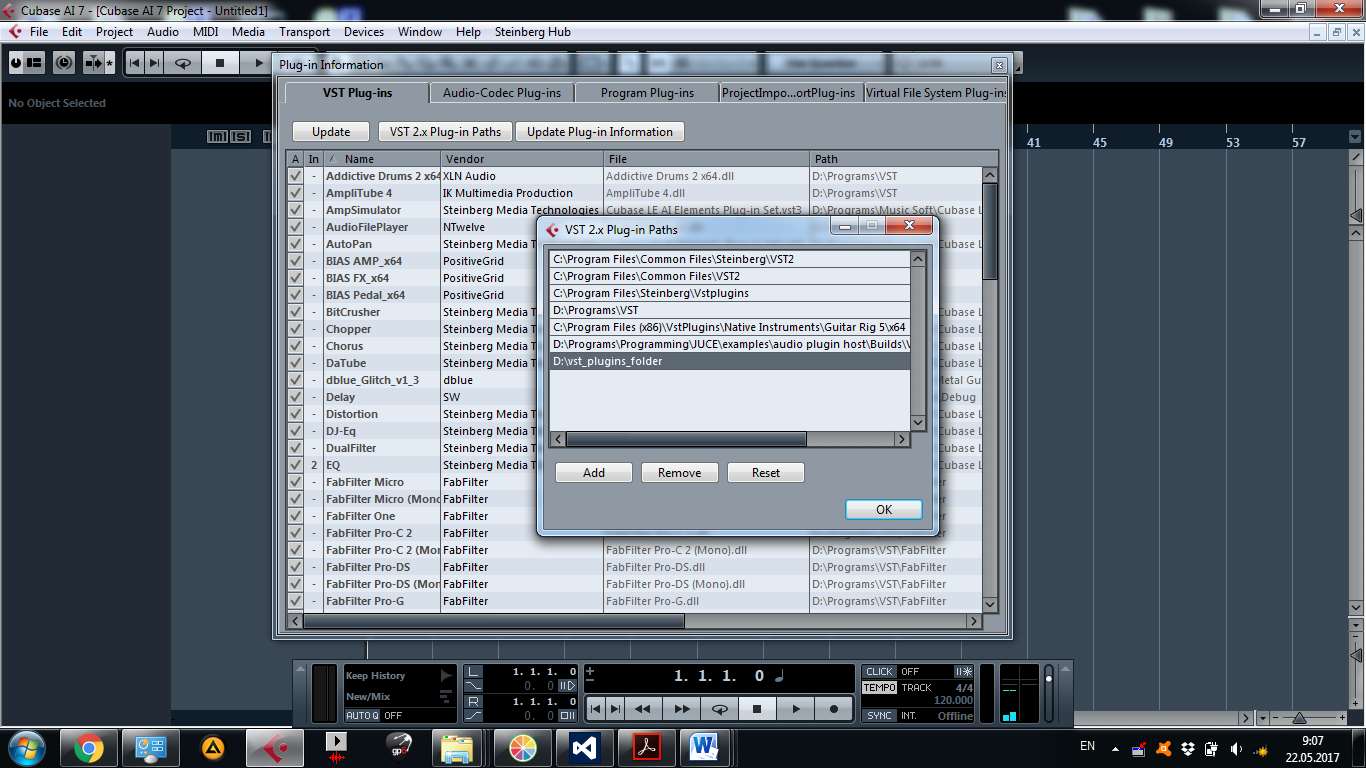


Рисунок 6.3 – Окно VST 2.x Plug-in Paths

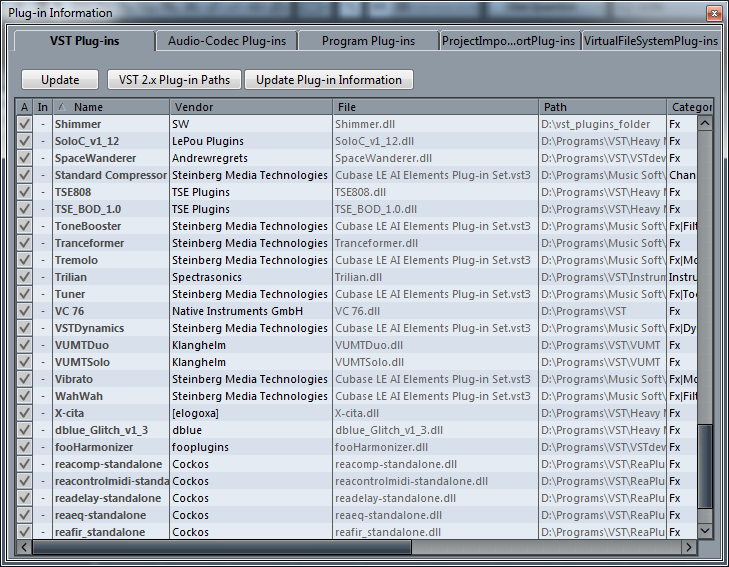


Рисунок 6.4 – Окно Plug-in Information

1. Чтобы использовать плагин на аудиодорожке нужно её создать. Нужно нажать правой кнопкой по области дорожек и выбрать Add Audio Track … (см. рисунок 6.5). В появившемся окне выбрать конфигурацию дорожки Mono, указать любое имя и нажать Add Track (см. рисунок 6.6).

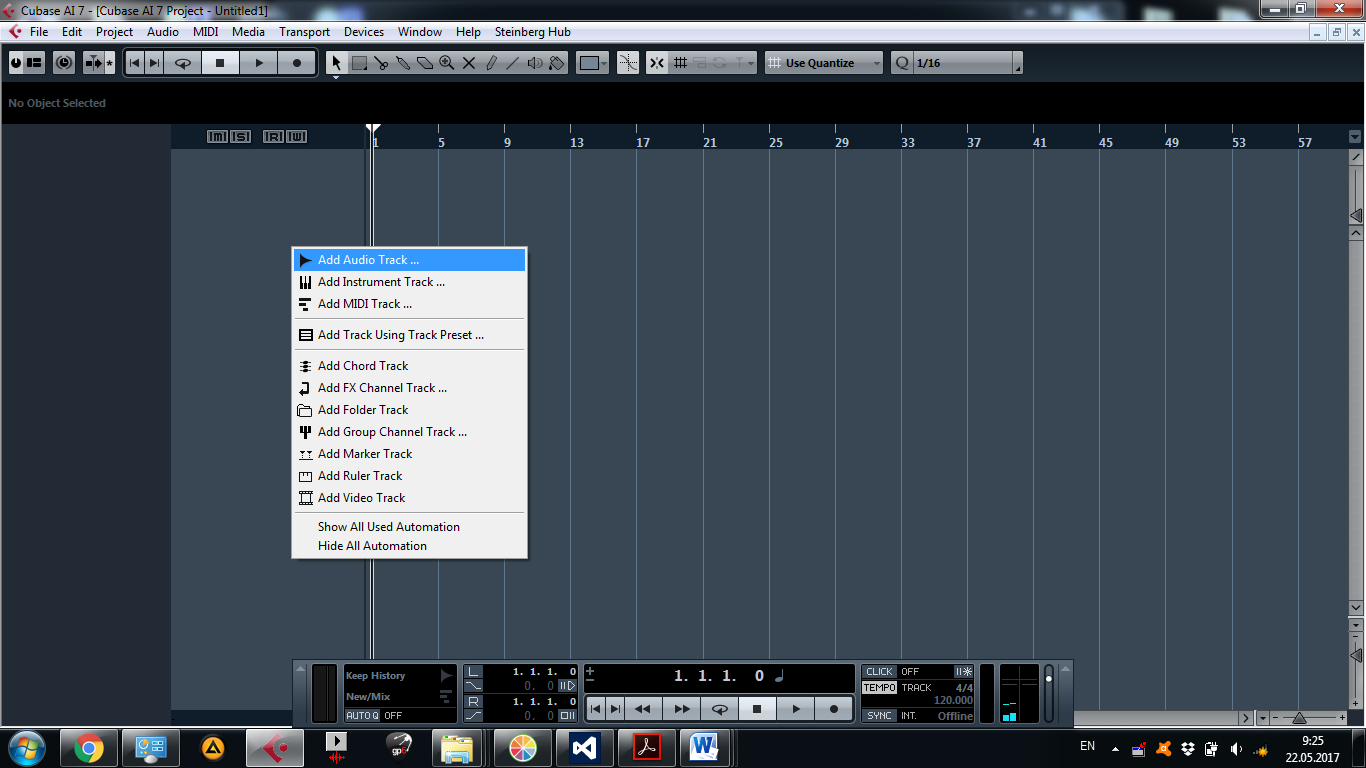


Рисунок 6.5 – Добавление звуковой дорожки 1

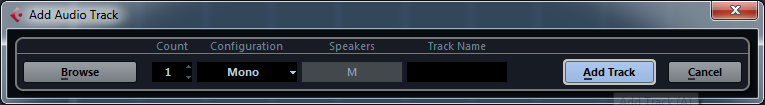


Рисунок 6.6 – Добавление звуковой дорожки 2



Рисунок 6.7 – Применение плагина к звуковой дорожке

1. Применить плагин к звуковой дорожке. Для этого необходимо выбрать дорожку, в меню Inserts нажать на пустой слот, выбрать необходимый плагин (см. рисунок 6.7). Теперь звуковая дорожка обрабатывается плагином в реальном времени.

## Пользовательский интерфейс



Рисунок 6.8 – Графический интерфейс программного модуля обработки звуковой дорожки в реальном времени в несинхронизированном режиме.

* + 1. Элементы управления эффекта Octaver

Компонент Octaver имеет следующие элементы графического интерфейса:

* кнопка Bypass;
* ручка Mix;
* ручка Octave low;
* ручка Octave high;

При нажатой кнопке Bypass, исходный звуковой сигнал не будет претерпевать обработку эффектом Delay и поступит на вход следующего компонента без изменений.

Ручка Mix регулирует соотношение обработанного сигнала эффектом Octaver к необработанному.

Ручка Octave low регулирует громкость добавляемого сигнала, который на октаву ниже основного.

Ручка Octave high регулирует громкость добавляемого сигнала, который на октаву выше основного.

* + 1. Элементы управления эффекта Delay

Эффект Delay управляется следующими элементами графического интерфейса:

* кнопка Bypass;
* ручка Mix;
* ручка Delay;
* ручка Feedback
* кнопка Synchronize
* кнопка Dotted;
* кнопка Second dotted.

При нажатой кнопке Bypass, звуковой сигнал, поступающий от блока Octaver, не будет претерпевать обработку эффектом Delay и поступит на вход следующего компонента Reverb без изменений.

Ручка Mix определяет количество эффекта в процентах, то есть громкость задержанного сигнала. Пределы изменения: от нуля до ста.

Ручкой Delay задаётся время, спустя которое сигнал будет повторяться. Пределы изменения этого параметра: от нуля до двух тысяч миллисекунд.

Ручка Feedback отвечает за обратную связь эффекта Delay. При минимальном значении равном нулю, компонент будет добавлять одну копию исходного сигнала. При максимальном – бесконечное количество копий. Если значение этого параметра находится в пределах от минимального, до максимального, громкость периодических копий сигнала будет постепенно уменьшаться.

Кнопка Synched меняет режим работы эффекта Delay на синхронный и обратно (см. рисунок 6.9). В синхронном режиме программный модуль обработки звуковой дорожки в реальном времени получает темп от хост-программы, то есть от цифровой звуковой рабочей станции, и синхронизирует копии сигнала с полученным темпом. В этом режиме ручка Delay изменяет своё название на Denominator. Вводимое значение при этом находится в пределах от 1 до 8. Также меняется предназначение ручки: в этом режиме с её помощью задаётся делитель времени между копиями сигнала.



Рисунок 6.9 – Графический интерфейс программного модуля обработки звуковой дорожки в реальном времени в синхронизированном с тэмпом режиме.

Кнопка Dotted добавляет одну вторую от времени между долями к синхронизированным с темпом повторам.

Кнопка Second dotted добавляет одну четверть от времени между долями к синхронизированным с темпом повторам.

При одновременно нажатых кнопках Dotted и Second dotted время между повторами увеличивается на три четверти от времени между долями.

* + 1. Элементы управления эффекта Reverb

Эффект Reverb управляется следующими элементами графического интерфейса:

* кнопка Bypass;
* ручка Decay;
* ручка Mix.

При нажатой кнопке Bypass, звуковой сигнал не будет претерпевать обработку эффектом Reverb и поступит на выход.

Ручка Mix определяет соотношение необработанного сигнала к обработанному эффектом сигналу. Этот параметр изменяется в пределах от нуля до единицы.

Ручкой Decay задаётся время затухания отражений. Оно равно времени, за которое звуковое давление уменьшается на 60 dB с момента внезапного окончания генерируемого тестового сигнала.

# ТЕХНИКО-ЭКОНОМИЧЕСКОЕ ОБОСНОВАНИЕ ЭФФЕКТИВНОСТИ РАЗРАБОТКИ И РЕАЛИЗАЦИИ ПРОГРАММНОГО ПРОДУКТА

## Характеристика программного продукта

Целью дипломного проекта является создание программного модуля обработки звуковой дорожки в реальном времени. Программный продукт может быть использован на обычных настольных компьютерах и ноутбуках во всех популярных цифровых звуковых рабочих станциях. Разработка ПО осуществляется IT-компанией.

Потенциальные пользователи программного модуля обработки звуковой дорожки в реальном времени:

* + - музыканты, имеющие возможность подключить свой инструмент к компьютеру для получения цифрового представления звука;
    - диджеи, осуществляющие воспроизведение записанных на звуковые носители музыкальных произведений с изменением материала техническими средствами в реальном времени;
    - звукорежиссёры, обрабатывающие записанные звуки или музыкальные произведения.

В связи с распространением цифровой обработки звука с помощью персонального компьютера на рынке есть достаточный спрос на данный программный продукт. Программный модуль обработки звуковой дорожки в реальном времени разрабатывается для свободной реализации на рынке информационных технологий копиями.

Исходя из тенденций на рынке модулей обработки звука, а также дополнительного маркетингового исследования, приложение будет востребовано на рынке в течение четырех лет: в 2017 году планируется реализовать 60 копий, в 2018 году планируется реализовать 70 копий, в 2019 планируется реализовать 70 копий, в 2020 планируется реализовать 50 копий. Итого 250 копий.

Экономическая целесообразность инвестиций в разработку и реализацию представленного программного продукта определяется на основе расчета таких показателей, как:

* чистая дисконтированная стоимость (ЧДД);
* срок окупаемости инвестиций (Ток);
* рентабельность инвестиций (Ри).

## Расчет сметы затрат на разработку и отпускной цены программного продукта

Расчет величины основной заработной платы участников команды осуществляется по формуле 7.1:

|  |  |
| --- | --- |
|  | (7.1) |
|  |  |

где *n* – количество исполнителей занятых в разработке;

Т1ч*i* – часовая тарифная ставка i-го исполнителя, руб.;

Кт*i* – тарифный коэффициент разряда i-го исполнителя;

Тч – количество часов работы в день, ч;

Фэ*i* – эффективный фонд рабочего времени i-го исполнителя;

К – коэффициент премирования, принятый равным 1,3.

Тарифная ставка первого разряда (на 01.03.2017) на предприятии составляет 120 рублей. Среднемесячная расчетная норма рабочего времени на 2017 год составляет 22 дня. Коэффициент премирования равен 1,3.

В процессе разработки приложения принимала участие команда из двух исполнителей: инженера-программиста 10 разряда с тарифным коэффициентом 2,48, и руководителя проекта 12 разряда с тарифным коэффициентом 3,25. Руководитель проекта ставит задачи перед инженером-программистом, среди которых следующие работы: проектирование интерфейса, написание кода, отладка и тестирование программного продукта.

Результаты расчёта основной заработной платы команды разработчиков представлены в таблице 7.1.

Таблица 7.1 – Расчет затрат на основную заработную плату команды разработчиков

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Участник  команды | Раз-  ряд | Тариф-  ный коэффи-  циент | Месячная  тарифная  ставка, руб. | Дневная тарифная  ставка, руб. | Плано-вый фонд рабочего времени, дн. | Заработная  плата, руб. |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| Руково-дитель проекта | 12 | 3,25 | 390 | 17,73 | 25 | 443,18 |
| Инженер-програм-  мист | 10 | 2,48 | 297,60 | 13,53 | 50 | 676,36 |
| Итого |  |  |  |  |  | 1119,55 |
| Премия (30%) |  |  |  |  |  | 335,86 |

*Продолжение таблицы 7.1*

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| Основная заработная плата (Зо) |  |  |  |  |  | 1455,41 |

Затраты на дополнительную заработную плату команды разработчиков включает выплаты, предусмотренные законодательством о труде (оплата отпусков, льготных часов, времени выполнения государственных обязанностей и других выплат, не связанных с основной деятельностью исполнителей) и определяется по формуле 7.2:

|  |  |
| --- | --- |
|  | (7.2) |

где – норматив дополнительной заработной платы, 10%.

Размер дополнительной заработной платы исполнителей составит:

Отчисления на социальные нужды (в фонд социальной защиты населения и на обязательное страхование) определяются в соответствии с действующими законодательными актами по формуле 7.3:

|  |  |
| --- | --- |
|  | (7.3) |

где – норматив отчислений на социальные нужды, 34,6%.

Размер отчислений в фонд социальной защиты населения и на обязательное страхование составляет:

Расходы по статье «Машинное время» включают оплату машинного времени, необходимого для разработки и отладки ПО и определяются по формуле 7.4:

|  |  |  |
| --- | --- | --- |
|  |  | (7.4) |

где Цм – цена одного машино-часа, руб.;

Тч – количество часов работы в день;

Тдн – время, затраченное на разработку проекта, дн.

Расходы по статье «Машинное время», на разработку программного продукта составляют:



Размер прочих затрат, включающий затраты на аренду помещения, освещение и отопление, вычисляется по формуле 7.5:

|  |  |
| --- | --- |
|  | (7.5) |

Норматив прочих затрат принят равным 10%. Тогда прочие затраты на разработку программного продукта составляют:

Затраты по статье «Накладные расходы» связаны с необходимостью содержания аппарата управления, вспомогательных хозяйств и опытных производств. Определяются по формуле:

|  |  |
| --- | --- |
|  | (7.6) |

где Нн – норматив накладных расходов, рассматриваемых в целом по организации (50%).

Общая сумма расходов по смете (Ср) определяется как сумма выше рассчитанных показателей:

|  |  |  |
| --- | --- | --- |
|  |  | (7.7) |

Подставив значения в формулу расчета расходов по смете 7.7, получим:

В дополнение к выше рассчитанным параметрам, определяются расходы на сопровождение и адаптацию программного продукта (Зса):

|  |  |  |
| --- | --- | --- |
|  |  | (7.8) |

где Нрса – это норматив расходов обусловленных сопровождением и адаптацией ПО (5%).

Используя формулу 7.8, определим значение расходов:

Общая сумма расходов на разработку приложения, как полная себестоимость программного продукта (Сп) определяется по формуле 7.9:

|  |  |  |
| --- | --- | --- |
|  |  | (7.9) |

Общая сумма расходов на разработку:

В таблице 7.2 указаны рассчитанные значения затрат на разработку программного продукта.

Таблица 7.2 – Смета затрат на разработку программного обеспечения

|  |  |  |
| --- | --- | --- |
| Статья затрат | Условное обозначение | Сумма, руб. |
| Основная заработная плата |  | 1455,41 |
| Дополнительная заработная плата |  | 145,54 |
| Отчисления на социальные нужды |  | 553,93 |
| Затраты по статье «Машинное время» |  | 720,00 |
| Прочие затраты |  | 145,54 |
| Затраты по статье «Накладные расходы» |  | 727,70 |
| Сумма расходов по смете |  | 3748,12 |
| Затраты на сопровождение и адаптацию |  | 187,41 |
| Полная себестоимость |  | 3935,53 |

Прогнозируемая прибыль программного продукта рассчитывается по формуле 7.10:

|  |  |  |
| --- | --- | --- |
|  |  | (7.10) |

где Пр – прибыль от реализации программного продукта;

Ур – планируемый уровень рентабельности (15%);

Сп – себестоимость программного продукта.

Получаем:

Прогнозируемая отпускная цена программного продукта будет определяться по формуле 7.11:

|  |  |  |
| --- | --- | --- |
|  |  | (7.11) |

Подставляя значения в формулу (7.10), получаем прогнозируемую отпускную цену программного продукта:

## Расчёт экономического эффекта от реализации ПО

Экономический эффект организации-разработчика программного обеспечения в данном случае заключается в получении прибыли от его продажи множеству потребителей. Прибыль от реализации в данном случае напрямую зависит от объемов продаж, цены реализации и затрат на разработку ПО.

На основании анализа цен на аналогичные программные продукты цена программного модуля обработки звуковой дорожки в реальном времени будет составлять 50 руб.

Запланированное количество проданных копий – 250. Прибыль от продажи одной копии рассчитывается по формуле 7.12:

|  |  |  |
| --- | --- | --- |
|  |  | (7.12) |

где Ц – цена реализации одной копии;

НДС – сумма налога на добавленную стоимость;

Зр – сумма расходов на разработку и реализацию;

N – количество реализованных копий.

В связи с тем, что компания-исполнитель является резидентом ПВТ и по текущему законодательству РБ такие предприятия освобождаются от уплаты налога на прибыль и на добавленную стоимость, в подсчёте прибыли от продажи одной копии программного продукта НДС участвовать не будет.

Итого, прибыль за продажу одной копии программного продукта:

Суммарная прибыль за год рассчитывается, как прибыль от продажи одной копии на количество копий:

|  |  |  |
| --- | --- | --- |
|  |  | (7.13) |

где Пед – прибыль от продажи одной копии, руб;

N – количество копий за год.

Прибыль от продажи программного продукта по годам, рассчитанная по формуле 7.13 составляет:

Так как компания-исполнитель по текущему законодательству освобождается от уплаты налога на прибыль, чистая прибыль по годам составит:

## Расчет показателей экономической эффективности инвестиций в разработку и реализацию программного продукта

В ходе реализации программного продукта чистая прибыль в конечном итоге возмещает расходы на разработку. Однако полученные при этом суммы результатов (прибыли) и затрат по годам следует привести к единому времени, а именно началу расчётного года. Для этого необходимо использовать дисконтирование путем умножения соответствующих результатов и затрат на коэффициент дисконтирования соответствующего года t, который определяется по формуле 7.14:

|  |  |
| --- | --- |
|  | (7.14) |

где – норма дисконта (в долях единиц), равная или больше средней процентной ставки по банковским депозитам действующей на момент осуществления расчетов, равная 0.14;

– порядковый номер года периода реализации инвестиционного проекта (предполагаемый период использования разрабатываемого ПО пользователем и время на разработку).

Чистый дисконтированный доход рассчитывается по формуле 7.15:

|  |  |
| --- | --- |
| , | (7.15) |

где – расчетный период, лет;

– результат (экономический эффект), полученный в году t, руб.;

– затраты (инвестиции в разработку программного обеспечения) в году t, руб.

Сведем данные расчета показателей экономической эффективности в таблицу 7.3.

Таблица 7.3 – Расчет показателей экономической эффективности разработки и реализации программного продукта

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Показатели | Ед. изм. | Годы | | | |
| 2017 | 2018 | 2019 | 2020 |
| 1 | 2 | 3 | 4 | 5 | 6 |
| Результаты: | | | | | |
| Экономический эффект | руб. | 1913,79 | 2232,76 | 2232,76 | 1594,83 |

*Продолжение таблицы 7.3*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 | 6 |
| Дисконтированный результат | руб. | 1913,79 | 1958,56 | 1718,04 | 1076,46 |
| Затраты: | | | | | |
| Затраты на разработку | руб. | 3935,53 |  |  |  |
| Дисконтированные вложения | руб. | 3935,53 |  |  |  |
| Экономический эффект: | | | | | |
| Чистый дисконтированный доход по годам | руб. | -2021,74 | 1958,56 | 1718,04 | 1076,46 |
| Чистый дисконтированный доход нарастающим итогом | руб. | -2021,74 | -63,18 | 1654,86 | 2731,32 |

Рентабельность инвестиций вычисляется по формуле 7.16:

|  |  |  |
| --- | --- | --- |
|  |  | (7.16) |

где ЧПср – среднегодовая величина чистой прибыли за расчетный период и высчитывается по формуле 7.17.

|  |  |  |
| --- | --- | --- |
|  |  | (7.17) |

Итого рентабельность инвестиций:

Чистый дисконтированный доход по истечении четырёх лет имеет значение больше нуля, то есть проект эффективен и инвестиции в разработку данного программного модуля обработки звуковой дорожки в реальном времени экономически целесообразны.

Полученные при расчёте данные, занесённые в таблицу 7.3, свидетельствуют о том, что чистый дисконтированный доход нарастающим итогом превышает значение нуля в 2019 году. Это означает, что срок окупаемости проекта равен трём годам.

В результате технико-экономического обоснования эффективности разработки и реализации программного продукта были определены следующие экономические показатели:

* + - чистая дисконтированная стоимость за четыре года работы программного продукта составит 2731,32 рублей;
    - срок окупаемости проекта равен трём годам;
    - рентабельность инвестиций составила 51%.

Таким образом, исходя из вышеперечисленных показателей можно сделать вывод, что проект является прибыльным и затраты на его разработку целесообразны.

# ЗАКЛЮЧЕНИЕ

В результате работы над дипломным проектом был реализован программный модуль обработки звуковой дорожки в реальном времени. Успешно выполнены основные задачи: разработан модуль эффекта Delay и модуль эффекта Reverb.

При разработке дипломного проекта была прочитана техническая литература, позволившая расширить круг знаний в областях программирования и программных средств разработки. Также были изучены средства работы со звуком на персональном компьютере, алгоритмы его обработки.

Исходя из требований к дипломному проекту, была произведена декомпозиция программного модуля на блоки: определены их назначения и связи блоков между собой. Следуя методологии объектно-ориентированного программирования, блоки были реализованы в виде классов.

Основные преимущества разработки:

* + - регулировка уровня входного и выходного сигналов;
    - наличие двух эффектов обработки в одном модуле;
    - удобное и гибкое управление временем задержки эффекта Delay.

В будущем возможны следующие улучшения программного модуля обработки звуковой дорожки в реальном времени:

* + - реализация модуля эффекта Octaver;
    - реализация возможности управления параметрами обработки звуковой дорожки по протоколу MIDI.

Во время разработки модуля обработки звуковой дорожки в реальном времени было проведено его тщательное тестирование. Оно позволило выявить незначительное количество ошибок, которые были успешно устранены. В результате программный модуль является работоспособным и надёжным.

Результатом проведённого технико-экономического обоснования разработки и реализации программного продукта стали следующие значения показателей экономической эффективности: срок окупаемости продукта составил 3 года, чистый дисконтированный доход за четыре года составил 2731 рубль, рентабельность инвестиций составила 51%. Таким образом данный программный продукт является прибыльным и затраты на его разработку целесообразны.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Wikipedia [Электронный ресурс]. – Режим доступа: https://ru.wikipedia.org/wiki/Цифровая звуковая рабочая станция.
2. Wikipedia [Электронный ресурс]. – Режим доступа: <https://ru.wikipedia.org/wiki/Плагин>.
3. Wikipedia [Электронный ресурс]. – Режим доступа: <https://ru.wikipedia.org/wiki/Virtual_Studio_Technology>.
4. Wikipedia [Электронный ресурс]. – Режим доступа: <https://ru.wikipedia.org/wiki/Октавер>.
5. Лайонс, Р. Цифровая обработка сигналов: Второе издание/ Лайонс Р. – Пер с англ. – М. : ООО «Бином-Пресс», 2006 г. – 656 с.
6. Cooley J. W., Tukey J. W. An algorithm for the machine calculation of complex Fourier series // Mathematics of Computation. 1965. V. 19. No. 90. P. 297-301.
7. DSPLIB [Электронный ресурс]. – Режим доступа: <http://ru.dsplib.org/content/fft_dec_in_time.html>.
8. Wikipedia [Электронный ресурс]. – Режим доступа: <https://ru.wikipedia.org/wiki/Дилэй>.
9. Audio Artillery [Электронный ресурс]. – Режим доступа: <http://audioartillery.com/projects/tonecore_dsp_dev_kit_guide>.
10. Wikipedia [Электронный ресурс]. – Режим доступа: <https://ru.wikipedia.org/wiki/Реверберация>.
11. Open Source Acoustic Design [Электронный ресурс] : Algorithmic Reverberation. – Режим доступа: http://arqen.com/wp-content/docs/Hybrid-Convolution-Algorithmic-Reverb.pdf.
12. Christianfloisand wordpress [Электронный ресурс]. – Режим доступа: <https://christianfloisand.wordpress.com/2012/10/18/algorithmic-reverbs-the-moorer-design>/.
13. Wikipedia [Электронный ресурс]. – Режим доступа: <https://ru.wikipedia.org/wiki/Juce>.
14. Juce [Электронный ресурс]. – Режим доступа: https://www.juce.com/doc/classes.
15. Juce [Электронный ресурс]. – Режим доступа: <https://www.juce.com/doc/tutorial_create_projucer_basic_plugin>.
16. GitHub [Электронный ресурс]. – Режим доступа: <https://github.com/>.

Палицын, В. А. Технико-экономическое обоснование дипломных проектов: методическое пособие в 4 ч. – Минск : БГУИР, 2005 г.

# ПРИЛОЖЕНИЕ А

(обязательное)

Листинг

Файл PluginEditor.h

#ifndef PLUGINEDITOR\_H\_INCLUDED

#define PLUGINEDITOR\_H\_INCLUDED

#include "../JuceLibraryCode/JuceHeader.h"

#include "PluginProcessor.h"

#include "./Images/Images.h"

class ShimmerAudioProcessorEditor : public AudioProcessorEditor,

public Timer,

public SliderListener,

public ButtonListener

{

public:

ShimmerAudioProcessorEditor (ShimmerAudioProcessor&);

~ShimmerAudioProcessorEditor();

//==============================================================================

void paint (Graphics&) override;

void resized() override;

void sliderValueChanged (Slider\* sliderThatWasMoved) override;

void buttonClicked (Button\* buttonThatWasClicked) override;

void visibilityChanged();

void timerCallback();

const String setSyncLabel(int sl);

private:

static const int window\_width = 550;

static const int window\_height = 550;

Image background\_image;

ShimmerAudioProcessor& processor;

ScopedPointer<Label> InputGainLabel;

ScopedPointer<Slider> InputGainKnob;

ScopedPointer<Label> OutputGainLabel;

ScopedPointer<Slider> OutputGainKnob;

struct Octaver\_GUI\_Components{

Octaver\_GUI\_Components(AudioProcessorEditor\* ae, ButtonListener\* bl, SliderListener\* sl)

{

ae->addAndMakeVisible (groupComponent = new GroupComponent ("Octaver group",

TRANS("Octaver ")));

groupComponent->setColour (GroupComponent::outlineColourId, Colour (0x66ebe9e9));

groupComponent->setColour (GroupComponent::textColourId, Colours::grey);

ae->addAndMakeVisible (MixLabel = new Label ("Mix Label",

TRANS("Mix")));

MixLabel->setTooltip (TRANS("Wet/Dry Mix"));

MixLabel->setFont (Font (Font::getDefaultMonospacedFontName(), 18.00f, Font::bold));

MixLabel->setJustificationType (Justification::centred);

MixLabel->setEditable (false, false, false);

MixLabel->setColour (Label::textColourId, Colours::azure);

MixLabel->setColour (TextEditor::textColourId, Colours::black);

MixLabel->setColour (TextEditor::backgroundColourId, Colour (0x00ffffff));

ae->addAndMakeVisible (OctaveLowLabel = new Label ("Octave Low Label",

TRANS("Octave Low")));

OctaveLowLabel->setFont (Font (Font::getDefaultMonospacedFontName(), 18.00f, Font::bold));

OctaveLowLabel->setJustificationType (Justification::centred);

OctaveLowLabel->setEditable (false, false, false);

OctaveLowLabel->setColour (Label::textColourId, Colours::azure);

OctaveLowLabel->setColour (TextEditor::textColourId, Colours::black);

OctaveLowLabel->setColour (TextEditor::backgroundColourId, Colour (0x00ffffff));

ae->addAndMakeVisible (OctaveHighLabel = new Label ("Octave High Label",

TRANS("Octave high")));

OctaveHighLabel->setFont (Font (Font::getDefaultMonospacedFontName(), 18.00f, Font::bold));

OctaveHighLabel->setJustificationType (Justification::centred);

OctaveHighLabel->setEditable (false, false, false);

OctaveHighLabel->setColour (Label::textColourId, Colours::azure);

OctaveHighLabel->setColour (TextEditor::textColourId, Colours::black);

OctaveHighLabel->setColour (TextEditor::backgroundColourId, Colour (0x00ffffff));

ae->addAndMakeVisible (MixKnob = new Slider ("Mix Knob"));

MixKnob->setExplicitFocusOrder (1);

MixKnob->setRange (0, 1, 0.01);

MixKnob->setSliderStyle (Slider::Rotary);

MixKnob->setTextBoxStyle (Slider::TextBoxBelow, false, 80, 20);

MixKnob->setColour (Slider::backgroundColourId, Colours::cadetblue);

MixKnob->setColour (Slider::trackColourId, Colours::coral);

MixKnob->setColour (Slider::rotarySliderFillColourId, Colours::aquamarine);

MixKnob->setColour (Slider::rotarySliderOutlineColourId, Colours::cadetblue);

MixKnob->setColour (Slider::textBoxTextColourId, Colours::aliceblue);

MixKnob->setColour (Slider::textBoxBackgroundColourId, Colours::cadetblue);

MixKnob->setColour (Slider::textBoxHighlightColourId, Colours::coral);

MixKnob->setColour (Slider::textBoxOutlineColourId, Colours::cadetblue);

MixKnob->addListener (sl);

ae->addAndMakeVisible (OctaveLowKnob = new Slider ("Octave Low Knob"));

OctaveLowKnob->setRange (0, 1, 0);

OctaveLowKnob->setSliderStyle (Slider::Rotary);

OctaveLowKnob->setTextBoxStyle (Slider::TextBoxBelow, false, 80, 20);

OctaveLowKnob->setColour (Slider::backgroundColourId, Colours::cadetblue);

OctaveLowKnob->setColour (Slider::trackColourId, Colours::coral);

OctaveLowKnob->setColour (Slider::rotarySliderFillColourId, Colours::aquamarine);

OctaveLowKnob->setColour (Slider::rotarySliderOutlineColourId, Colours::cadetblue);

OctaveLowKnob->setColour (Slider::textBoxTextColourId, Colours::aliceblue);

OctaveLowKnob->setColour (Slider::textBoxBackgroundColourId, Colours::cadetblue);

OctaveLowKnob->setColour (Slider::textBoxHighlightColourId, Colours::coral);

OctaveLowKnob->setColour (Slider::textBoxOutlineColourId, Colours::cadetblue);

OctaveLowKnob->addListener (sl);

ae->addAndMakeVisible (OctaveHighKnob = new Slider ("Octave High Knob"));

OctaveHighKnob->setExplicitFocusOrder (1);

OctaveHighKnob->setRange (0, 1, 0);

OctaveHighKnob->setSliderStyle (Slider::Rotary);

OctaveHighKnob->setTextBoxStyle (Slider::TextBoxBelow, false, 80, 20);

OctaveHighKnob->setColour (Slider::backgroundColourId, Colours::cadetblue);

OctaveHighKnob->setColour (Slider::trackColourId, Colours::coral);

OctaveHighKnob->setColour (Slider::rotarySliderFillColourId, Colours::aquamarine);

OctaveHighKnob->setColour (Slider::rotarySliderOutlineColourId, Colours::cadetblue);

OctaveHighKnob->setColour (Slider::textBoxTextColourId, Colours::aliceblue);

OctaveHighKnob->setColour (Slider::textBoxBackgroundColourId, Colours::cadetblue);

OctaveHighKnob->setColour (Slider::textBoxHighlightColourId, Colours::coral);

OctaveHighKnob->setColour (Slider::textBoxOutlineColourId, Colours::cadetblue);

OctaveHighKnob->addListener (sl);

ae->addAndMakeVisible (BypassButton = new TextButton ("Bypass Button"));

BypassButton->setButtonText (TRANS("Bypass"));

BypassButton->addListener (bl);

BypassButton->setColour (TextButton::buttonColourId, Colours::cadetblue);

BypassButton->setColour (TextButton::buttonOnColourId, Colours::coral);

BypassButton->setColour (TextButton::textColourOnId, Colours::azure);

BypassButton->setColour (TextButton::textColourOffId, Colours::azure);

BypassButton->setClickingTogglesState(true);

}

void resized(Rectangle<int> r)

{

int indent = 15;

int button\_width = 60;

int button\_height = 25;

int text\_height = 40;

groupComponent->setBounds(r);

r.reduce(indent, indent);

Rectangle<int> button\_bypass\_area (r.removeFromTop (button\_height));

button\_bypass\_area.setWidth(button\_bypass\_area.proportionOfWidth(0.15f));

BypassButton->setBounds(button\_bypass\_area);

Rectangle<int> octaver\_area (r);

Rectangle<int> slider\_area (octaver\_area.removeFromTop (octaver\_area.getHeight()));

Rectangle<int> mix\_area(slider\_area.removeFromLeft (slider\_area.getWidth()/3));

MixLabel->setBounds (mix\_area.removeFromTop(text\_height));

MixKnob->setBounds (mix\_area);

Rectangle<int> octave\_low\_area(slider\_area.removeFromLeft (slider\_area.getWidth()/2));

OctaveLowLabel->setBounds (octave\_low\_area.removeFromTop(text\_height));

OctaveLowKnob->setBounds (octave\_low\_area);

Rectangle<int> octave\_high(slider\_area.removeFromLeft (slider\_area.getWidth()));

OctaveHighLabel->setBounds (octave\_high.removeFromTop (text\_height));

OctaveHighKnob->setBounds (octave\_high);

}

ScopedPointer<Label> MixLabel;

ScopedPointer<Label> OctaveLowLabel;

ScopedPointer<Label> OctaveHighLabel;

ScopedPointer<Slider> MixKnob;

ScopedPointer<Slider> OctaveLowKnob;

ScopedPointer<Slider> OctaveHighKnob;

ScopedPointer<TextButton> BypassButton;

ScopedPointer<GroupComponent> groupComponent;

} octaver\_gui\_components;

struct Delay\_GUI\_Components{

Delay\_GUI\_Components(AudioProcessorEditor\* ae, ButtonListener\* bl, SliderListener\* sl)

{

ae->addAndMakeVisible (groupComponent = new GroupComponent ("Delay group",

TRANS("Delay ")));

groupComponent->setColour (GroupComponent::outlineColourId, Colour (0x66ebe9e9));

groupComponent->setColour (GroupComponent::textColourId, Colours::grey);

ae->addAndMakeVisible (timecodeDisplayLabel = new Label ("Time Code",

TRANS("")));

timecodeDisplayLabel->setJustificationType (Justification::centred);

timecodeDisplayLabel->setEditable (false, false, false);

timecodeDisplayLabel->setColour (Label::textColourId, Colours::azure);

timecodeDisplayLabel->setColour (TextEditor::textColourId, Colours::black);

timecodeDisplayLabel->setColour (TextEditor::backgroundColourId, Colour (0x00ffffff));

ae->addAndMakeVisible (DelayLabel = new Label ("Delay 1 Label",

TRANS("Delay")));

DelayLabel->setFont (Font (Font::getDefaultMonospacedFontName(), 18.00f, Font::bold));

DelayLabel->setJustificationType (Justification::centred);

DelayLabel->setEditable (false, false, false);

DelayLabel->setColour (Label::textColourId, Colours::azure);

DelayLabel->setColour (TextEditor::textColourId, Colours::black);

DelayLabel->setColour (TextEditor::backgroundColourId, Colour (0x00ffffff));

ae->addAndMakeVisible (MixLabel = new Label ("Mix Label",

TRANS("Mix")));

MixLabel->setTooltip (TRANS("Wet/Dry Mix"));

MixLabel->setFont (Font (Font::getDefaultMonospacedFontName(), 18.00f, Font::bold));

MixLabel->setJustificationType (Justification::centred);

MixLabel->setEditable (false, false, false);

MixLabel->setColour (Label::textColourId, Colours::azure);

MixLabel->setColour (TextEditor::textColourId, Colours::black);

MixLabel->setColour (TextEditor::backgroundColourId, Colour (0x00ffffff));

ae->addAndMakeVisible (FeedbackLabel = new Label ("Feedback",

TRANS("Feedback")));

FeedbackLabel->setTooltip (TRANS("Feedback Percentage"));

FeedbackLabel->setFont (Font (Font::getDefaultMonospacedFontName(), 18.00f, Font::bold));

FeedbackLabel->setJustificationType (Justification::centred);

FeedbackLabel->setEditable (false, false, false);

FeedbackLabel->setColour (Label::textColourId, Colours::azure);

FeedbackLabel->setColour (TextEditor::textColourId, Colours::black);

FeedbackLabel->setColour (TextEditor::backgroundColourId, Colour (0x00ffffff));

ae->addAndMakeVisible (DelayKnob = new Slider ("Delay Knob"));

DelayKnob->setRange (0, 2000, 1);

DelayKnob->setSliderStyle (Slider::Rotary);

DelayKnob->setTextBoxStyle (Slider::TextBoxBelow, false, 80, 20);

DelayKnob->setColour (Slider::backgroundColourId, Colours::cadetblue);

DelayKnob->setColour (Slider::trackColourId, Colours::coral);

DelayKnob->setColour (Slider::rotarySliderFillColourId, Colours::aquamarine);

DelayKnob->setColour (Slider::rotarySliderOutlineColourId, Colours::cadetblue);

DelayKnob->setColour (Slider::textBoxTextColourId, Colours::aliceblue);

DelayKnob->setColour (Slider::textBoxBackgroundColourId, Colours::cadetblue);

DelayKnob->setColour (Slider::textBoxHighlightColourId, Colours::coral);

DelayKnob->setColour (Slider::textBoxOutlineColourId, Colours::cadetblue);

DelayKnob->addListener (sl);

ae->addAndMakeVisible (MixKnob = new Slider ("Mix Knob"));

MixKnob->setRange (0, 1, 0.01);

MixKnob->setSliderStyle (Slider::Rotary);

MixKnob->setTextBoxStyle (Slider::TextBoxBelow, false, 80, 20);

MixKnob->setColour (Slider::backgroundColourId, Colours::cadetblue);

MixKnob->setColour (Slider::trackColourId, Colours::coral);

MixKnob->setColour (Slider::rotarySliderFillColourId, Colours::aquamarine);

MixKnob->setColour (Slider::rotarySliderOutlineColourId, Colours::cadetblue);

MixKnob->setColour (Slider::textBoxTextColourId, Colours::aliceblue);

MixKnob->setColour (Slider::textBoxBackgroundColourId, Colours::cadetblue);

MixKnob->setColour (Slider::textBoxHighlightColourId, Colours::coral);

MixKnob->setColour (Slider::textBoxOutlineColourId, Colours::cadetblue);

MixKnob->addListener (sl);

ae->addAndMakeVisible (FeedbackKnob = new Slider ("Feedback Knob"));

FeedbackKnob->setTooltip (TRANS("Feedback"));

FeedbackKnob->setExplicitFocusOrder (2);

FeedbackKnob->setRange (0, 1, 0.01);

FeedbackKnob->setSliderStyle (Slider::Rotary);

FeedbackKnob->setTextBoxStyle (Slider::TextBoxBelow, false, 80, 20);

FeedbackKnob->setColour (Slider::backgroundColourId, Colours::cadetblue);

FeedbackKnob->setColour (Slider::trackColourId, Colours::coral);

FeedbackKnob->setColour (Slider::rotarySliderFillColourId, Colours::aquamarine);

FeedbackKnob->setColour (Slider::rotarySliderOutlineColourId, Colours::cadetblue);

FeedbackKnob->setColour (Slider::textBoxTextColourId, Colours::aliceblue);

FeedbackKnob->setColour (Slider::textBoxBackgroundColourId, Colours::cadetblue);

FeedbackKnob->setColour (Slider::textBoxHighlightColourId, Colours::coral);

FeedbackKnob->setColour (Slider::textBoxOutlineColourId, Colours::cadetblue);

FeedbackKnob->addListener (sl);

ae->addAndMakeVisible (BypassButton = new TextButton ("Bypass Button"));

BypassButton->setButtonText (TRANS("Bypass"));

BypassButton->addListener (bl);

BypassButton->setColour (TextButton::buttonColourId, Colours::cadetblue);

BypassButton->setColour (TextButton::buttonOnColourId, Colours::coral);

BypassButton->setColour (TextButton::textColourOnId, Colours::azure);

BypassButton->setColour (TextButton::textColourOffId, Colours::azure);

ae->addAndMakeVisible (SynchButton = new TextButton ("Synch Button"));

SynchButton->setButtonText (TRANS("Synchronize"));

SynchButton->addListener (bl);

SynchButton->setColour (TextButton::buttonColourId, Colours::cadetblue);

SynchButton->setColour (TextButton::buttonOnColourId, Colours::coral);

SynchButton->setColour (TextButton::textColourOnId, Colours::azure);

SynchButton->setColour (TextButton::textColourOffId, Colours::azure);

ae->addAndMakeVisible (DottedButton = new TextButton ("Dotted Button"));

DottedButton->setButtonText (TRANS("Dotted"));

DottedButton->addListener (bl);

DottedButton->setColour (TextButton::buttonColourId, Colours::cadetblue);

DottedButton->setColour (TextButton::buttonOnColourId, Colours::coral);

DottedButton->setColour (TextButton::textColourOnId, Colours::azure);

DottedButton->setColour (TextButton::textColourOffId, Colours::azure);

DottedButton->setVisible(false);

ae->addAndMakeVisible (SecondDottedButton = new TextButton ("Second Dotted Button"));

SecondDottedButton->setButtonText (TRANS("Second dotted"));

SecondDottedButton->addListener (bl);

SecondDottedButton->setColour (TextButton::buttonColourId, Colours::cadetblue);

SecondDottedButton->setColour (TextButton::buttonOnColourId, Colours::coral);

SecondDottedButton->setColour (TextButton::textColourOnId, Colours::azure);

SecondDottedButton->setColour (TextButton::textColourOffId, Colours::azure);

SecondDottedButton->setVisible(false);

BypassButton->setClickingTogglesState(true);

SynchButton->setClickingTogglesState(true);

DottedButton->setClickingTogglesState(true);

SecondDottedButton->setClickingTogglesState(true);

}

void resized(Rectangle<int> r)

{

int indent = 15;

int button\_width = 60;

int button\_height = 25;

int text\_height = 40;

groupComponent->setBounds(r);

r.reduce(indent, indent);

Rectangle<int> button\_bypass\_area (r.removeFromTop (button\_height));

BypassButton->setBounds(button\_bypass\_area.removeFromLeft(button\_bypass\_area.proportionOfWidth(0.15f)));

timecodeDisplayLabel->setBounds (button\_bypass\_area);

Rectangle<int> button\_area (r.removeFromBottom (button\_height));

SynchButton->setBounds(button\_area.removeFromLeft(button\_area.getWidth()/3));

DottedButton->setBounds(button\_area.removeFromLeft(button\_area.getWidth()/2));

SecondDottedButton->setBounds(button\_area.removeFromLeft(button\_area.getWidth()));

Rectangle<int> delay\_area (r);

Rectangle<int> slider\_area (delay\_area.removeFromTop (delay\_area.getHeight()));

Rectangle<int> mix\_area(slider\_area.removeFromLeft (slider\_area.getWidth()/3));

MixLabel->setBounds (mix\_area.removeFromTop(text\_height));

MixKnob->setBounds (mix\_area);

Rectangle<int> time\_area(slider\_area.removeFromLeft (slider\_area.getWidth()/2));

DelayLabel->setBounds(time\_area.removeFromTop(text\_height));

DelayKnob->setBounds (time\_area);

Rectangle<int> feedback\_area(slider\_area.removeFromLeft (slider\_area.getWidth()));

FeedbackLabel->setBounds (feedback\_area.removeFromTop (text\_height));

FeedbackKnob->setBounds (feedback\_area);

}

ScopedPointer<GroupComponent> groupComponent;

ScopedPointer<Label> timecodeDisplayLabel;

ScopedPointer<Label> DelayLabel;

ScopedPointer<Label> MixLabel;

ScopedPointer<Label> FeedbackLabel;

ScopedPointer<Slider> MixKnob;

ScopedPointer<Slider> FeedbackKnob;

ScopedPointer<Slider> DelayKnob;

ScopedPointer<TextButton> BypassButton;

ScopedPointer<TextButton> SynchButton;

ScopedPointer<TextButton> DottedButton;

ScopedPointer<TextButton> SecondDottedButton;

} delay\_gui\_components;

struct Reverb\_GUI\_Components{

Reverb\_GUI\_Components(AudioProcessorEditor\* ae, ButtonListener\* bl, SliderListener\* sl)

{

ae->addAndMakeVisible (groupComponent = new GroupComponent ("Reverb group",

TRANS("Reverb ")));

groupComponent->setColour (GroupComponent::outlineColourId, Colour (0x66ebe9e9));

groupComponent->setColour (GroupComponent::textColourId, Colours::grey);

ae->addAndMakeVisible (DecayLabel = new Label ("Decay Label",

TRANS("Decay")));

DecayLabel->setFont (Font (Font::getDefaultMonospacedFontName(), 18.00f, Font::bold));

DecayLabel->setJustificationType (Justification::centred);

DecayLabel->setEditable (false, false, false);

DecayLabel->setColour (Label::textColourId, Colours::azure);

DecayLabel->setColour (TextEditor::textColourId, Colours::black);

DecayLabel->setColour (TextEditor::backgroundColourId, Colour (0x00ffffff));

ae->addAndMakeVisible (MixLabel = new Label ("Mix Label",

TRANS("Mix")));

MixLabel->setTooltip (TRANS("Wet/Dry Mix"));

MixLabel->setFont (Font (Font::getDefaultMonospacedFontName(), 18.00f, Font::bold));

MixLabel->setJustificationType (Justification::centred);

MixLabel->setEditable (false, false, false);

MixLabel->setColour (Label::textColourId, Colours::azure);

MixLabel->setColour (TextEditor::textColourId, Colours::black);

MixLabel->setColour (TextEditor::backgroundColourId, Colour (0x00ffffff));

ae->addAndMakeVisible (DecayKnob = new Slider ("Decay Knob"));

DecayKnob->setTooltip (TRANS("Decay"));

DecayKnob->setExplicitFocusOrder (1);

DecayKnob->setRange (0.01, 60, 0.01);

DecayKnob->setSliderStyle (Slider::Rotary);

DecayKnob->setTextBoxStyle (Slider::TextBoxBelow, false, 80, 20);

DecayKnob->setColour (Slider::backgroundColourId, Colours::cadetblue);

DecayKnob->setColour (Slider::trackColourId, Colours::coral);

DecayKnob->setColour (Slider::rotarySliderFillColourId, Colours::aquamarine);

DecayKnob->setColour (Slider::rotarySliderOutlineColourId, Colours::cadetblue);

DecayKnob->setColour (Slider::textBoxTextColourId, Colours::aliceblue);

DecayKnob->setColour (Slider::textBoxBackgroundColourId, Colours::cadetblue);

DecayKnob->setColour (Slider::textBoxHighlightColourId, Colours::coral);

DecayKnob->setColour (Slider::textBoxOutlineColourId, Colours::cadetblue);

DecayKnob->addListener (sl);

ae->addAndMakeVisible (MixKnob = new Slider ("Mix Knob"));

MixKnob->setExplicitFocusOrder (1);

MixKnob->setRange (0, 1, 0.01);

MixKnob->setSliderStyle (Slider::Rotary);

MixKnob->setTextBoxStyle (Slider::TextBoxBelow, false, 80, 20);

MixKnob->setColour (Slider::backgroundColourId, Colours::cadetblue);

MixKnob->setColour (Slider::trackColourId, Colours::coral);

MixKnob->setColour (Slider::rotarySliderFillColourId, Colours::aquamarine);

MixKnob->setColour (Slider::rotarySliderOutlineColourId, Colours::cadetblue);

MixKnob->setColour (Slider::textBoxTextColourId, Colours::aliceblue);

MixKnob->setColour (Slider::textBoxBackgroundColourId, Colours::cadetblue);

MixKnob->setColour (Slider::textBoxHighlightColourId, Colours::coral);

MixKnob->setColour (Slider::textBoxOutlineColourId, Colours::cadetblue);

MixKnob->addListener (sl);

ae->addAndMakeVisible (BypassButton = new TextButton ("Bypass Button"));

BypassButton->setButtonText (TRANS("Bypass"));

BypassButton->addListener (bl);

BypassButton->setColour (TextButton::buttonColourId, Colours::cadetblue);

BypassButton->setColour (TextButton::buttonOnColourId, Colours::coral);

BypassButton->setColour (TextButton::textColourOnId, Colours::azure);

BypassButton->setColour (TextButton::textColourOffId, Colours::azure);

BypassButton->setClickingTogglesState(true);

}

void resized(Rectangle<int> r)

{

int indent = 15;

int button\_width = 60;

int button\_height = 25;

int text\_height = 40;

groupComponent->setBounds(r);

r.reduce(indent, indent);

Rectangle<int> button\_bypass\_area (r.removeFromTop (button\_height));

button\_bypass\_area.setWidth(button\_bypass\_area.proportionOfWidth(0.15f));

BypassButton->setBounds(button\_bypass\_area);

Rectangle<int> reverb\_area (r);

Rectangle<int> slider\_area (reverb\_area.removeFromTop (reverb\_area.getHeight()));

Rectangle<int> mix\_area(slider\_area.removeFromLeft (slider\_area.getWidth()/2));

MixLabel->setBounds (mix\_area.removeFromTop(text\_height));

MixKnob->setBounds (mix\_area);

Rectangle<int> decay\_area(slider\_area.removeFromLeft (slider\_area.getWidth()));

DecayLabel->setBounds (decay\_area.removeFromTop (text\_height));

DecayKnob->setBounds (decay\_area);

}

ScopedPointer<Label> DecayLabel;

ScopedPointer<Label> MixLabel;

ScopedPointer<Slider> DecayKnob;

ScopedPointer<Slider> MixKnob;

ScopedPointer<TextButton> BypassButton;

ScopedPointer<GroupComponent> groupComponent;

} reverb\_gui\_components;

void updateTimecodeDisplay (AudioPlayHead::CurrentPositionInfo pos);

JUCE\_DECLARE\_NON\_COPYABLE\_WITH\_LEAK\_DETECTOR (ShimmerAudioProcessorEditor)

};

#endif // PLUGINEDITOR\_H\_INCLUDED

Файл PluginEditor.cpp

#include "PluginProcessor.h"

#include "PluginEditor.h"

ShimmerAudioProcessorEditor::ShimmerAudioProcessorEditor (ShimmerAudioProcessor& p)

: AudioProcessorEditor (&p), processor (p),

octaver\_gui\_components(this, this, this),

delay\_gui\_components(this, this, this),

reverb\_gui\_components(this, this, this)

{

delay\_gui\_components.DelayKnob->setValue(processor.getParameter(ShimmerAudioProcessor::DelayTime));

delay\_gui\_components.FeedbackKnob->setValue(processor.getParameter(ShimmerAudioProcessor::DelayFeedback));

delay\_gui\_components.MixKnob->setValue(processor.getParameter(ShimmerAudioProcessor::DelayMix));

reverb\_gui\_components.DecayKnob->setValue(processor.getParameter(ShimmerAudioProcessor::ReverbDecay));

reverb\_gui\_components.MixKnob->setValue(processor.getParameter(ShimmerAudioProcessor::ReverbMix));

background\_image = ImageCache::getFromMemory(Images::background\_jpg, Images::background\_jpgSize);

addAndMakeVisible (InputGainLabel = new Label ("Input Gain",

TRANS("Input Gain")));

InputGainLabel->setFont (Font (Font::getDefaultMonospacedFontName(), 18.00f, Font::bold));

InputGainLabel->setJustificationType (Justification::centred);

InputGainLabel->setEditable (false, false, false);

InputGainLabel->setColour (Label::textColourId, Colours::azure);

InputGainLabel->setColour (TextEditor::textColourId, Colours::black);

InputGainLabel->setColour (TextEditor::backgroundColourId, Colour (0x00ffffff));

addAndMakeVisible (InputGainKnob = new Slider ("Input Gain Knob"));

InputGainKnob->setExplicitFocusOrder (1);

InputGainKnob->setRange (0, 1.5, 0.01);//(0, 1.1, 0.01);

InputGainKnob->setSliderStyle (Slider::Rotary);

InputGainKnob->setTextBoxStyle (Slider::TextBoxBelow, false, 80, 20);

InputGainKnob->setColour (Slider::backgroundColourId, Colours::cadetblue);

InputGainKnob->setColour (Slider::trackColourId, Colours::coral);

InputGainKnob->setColour (Slider::rotarySliderFillColourId, Colours::aquamarine);

InputGainKnob->setColour (Slider::rotarySliderOutlineColourId, Colours::cadetblue);

InputGainKnob->setColour (Slider::textBoxTextColourId, Colours::aliceblue);

InputGainKnob->setColour (Slider::textBoxBackgroundColourId, Colours::cadetblue);

InputGainKnob->setColour (Slider::textBoxHighlightColourId, Colours::coral);

InputGainKnob->setColour (Slider::textBoxOutlineColourId, Colours::cadetblue);

InputGainKnob->addListener (this);

InputGainKnob->setValue(processor.getParameter(ShimmerAudioProcessor::InputGain));

addAndMakeVisible (OutputGainLabel = new Label ("Output Gain",

TRANS("Output Gain")));

OutputGainLabel->setFont (Font (Font::getDefaultMonospacedFontName(), 18.00f, Font::bold));

OutputGainLabel->setJustificationType (Justification::centred);

OutputGainLabel->setEditable (false, false, false);

OutputGainLabel->setColour (Label::textColourId, Colours::azure);

OutputGainLabel->setColour (TextEditor::textColourId, Colours::black);

OutputGainLabel->setColour (TextEditor::backgroundColourId, Colour (0x00ffffff));

addAndMakeVisible (OutputGainKnob = new Slider ("Output Gain Knob"));

OutputGainKnob->setExplicitFocusOrder (1);

OutputGainKnob->setRange (0, 1.5, 0.01);

OutputGainKnob->setSliderStyle (Slider::Rotary);

OutputGainKnob->setTextBoxStyle (Slider::TextBoxBelow, false, 80, 20);

OutputGainKnob->setColour (Slider::backgroundColourId, Colours::cadetblue);

OutputGainKnob->setColour (Slider::trackColourId, Colours::coral);

OutputGainKnob->setColour (Slider::rotarySliderFillColourId, Colours::aquamarine);

OutputGainKnob->setColour (Slider::rotarySliderOutlineColourId, Colours::cadetblue);

OutputGainKnob->setColour (Slider::textBoxTextColourId, Colours::aliceblue);

OutputGainKnob->setColour (Slider::textBoxBackgroundColourId, Colours::cadetblue);

OutputGainKnob->setColour (Slider::textBoxHighlightColourId, Colours::coral);

OutputGainKnob->setColour (Slider::textBoxOutlineColourId, Colours::cadetblue);

OutputGainKnob->addListener (this);

OutputGainKnob->setValue(processor.getParameter(ShimmerAudioProcessor::OutputGain));

setSize(window\_width,window\_height);

startTimer(200);

}

ShimmerAudioProcessorEditor::~ShimmerAudioProcessorEditor()

{

delay\_gui\_components.groupComponent = nullptr;

delay\_gui\_components.DelayLabel = nullptr;

delay\_gui\_components.MixLabel = nullptr;

delay\_gui\_components.FeedbackLabel = nullptr;

delay\_gui\_components.DelayKnob = nullptr;

delay\_gui\_components.FeedbackKnob = nullptr;

delay\_gui\_components.MixKnob = nullptr;

delay\_gui\_components.BypassButton = nullptr;

delay\_gui\_components.SynchButton = nullptr;

delay\_gui\_components.DottedButton = nullptr;

delay\_gui\_components.SecondDottedButton = nullptr;

}

//==============================================================================

void ShimmerAudioProcessorEditor::paint (Graphics& g)

{

//g.fillAll (Colour (0xff353131));

//g.setColour (Colours::black.withAlpha (0.850f));

g.drawImage (background\_image, 0, 0, getWidth(), getHeight(), 200, 200, getWidth(), getHeight());

}

void ShimmerAudioProcessorEditor::resized()

{

int text\_height = 30;

int knob\_size = 80;

int gain\_area\_width = 100;

int octaver\_height = 175;

int delay\_height = 200;

int reverb\_height = 175;

Rectangle<int> window\_rectangle = getLocalBounds();

Rectangle<int> gain\_area = window\_rectangle.removeFromLeft(gain\_area\_width);

Rectangle<int> octaver\_rectangle = window\_rectangle.removeFromTop(octaver\_height);

Rectangle<int> delay\_rectangle = window\_rectangle.removeFromTop(delay\_height);

Rectangle<int> reverb\_rectangle = window\_rectangle.removeFromTop(reverb\_height);

octaver\_gui\_components.resized(octaver\_rectangle);

delay\_gui\_components.resized(delay\_rectangle);

reverb\_gui\_components.resized(reverb\_rectangle);

InputGainLabel->setBounds (gain\_area.removeFromTop(text\_height));

InputGainKnob->setBounds (gain\_area.removeFromTop(knob\_size));

OutputGainKnob->setBounds (gain\_area.removeFromBottom(knob\_size));

OutputGainLabel->setBounds (gain\_area.removeFromBottom(text\_height));

}

void ShimmerAudioProcessorEditor::sliderValueChanged (Slider\* sliderThatWasMoved)

{

if (sliderThatWasMoved == delay\_gui\_components.MixKnob)

{

processor.setParameterNotifyingHost(ShimmerAudioProcessor::DelayMix, (float)delay\_gui\_components.MixKnob->getValue());

}

else if (sliderThatWasMoved == delay\_gui\_components.FeedbackKnob)

{

processor.setParameterNotifyingHost(ShimmerAudioProcessor::DelayFeedback, (float)delay\_gui\_components.FeedbackKnob->getValue());

}

else if (sliderThatWasMoved == delay\_gui\_components.DelayKnob)

{

processor.setParameterNotifyingHost(ShimmerAudioProcessor::DelayTime, (float)delay\_gui\_components.DelayKnob->getValue());

}

else if (sliderThatWasMoved == reverb\_gui\_components.MixKnob)

{

processor.setParameterNotifyingHost(ShimmerAudioProcessor::ReverbMix, (float)reverb\_gui\_components.MixKnob->getValue());

}

else if (sliderThatWasMoved == reverb\_gui\_components.DecayKnob)

{

processor.setParameterNotifyingHost(ShimmerAudioProcessor::ReverbDecay, (float)reverb\_gui\_components.DecayKnob->getValue());

}

else if (sliderThatWasMoved == InputGainKnob)

{

processor.setParameterNotifyingHost(ShimmerAudioProcessor::InputGain, (float)sliderThatWasMoved->getValue());

}

else if (sliderThatWasMoved == OutputGainKnob)

{

processor.setParameterNotifyingHost(ShimmerAudioProcessor::OutputGain, (float)sliderThatWasMoved->getValue());

}

}

void ShimmerAudioProcessorEditor::buttonClicked (Button\* buttonThatWasClicked)

{

static bool synch\_mode\_on = false;

static float old\_delay;

if (buttonThatWasClicked == delay\_gui\_components.BypassButton)

{

processor.setParameterNotifyingHost(ShimmerAudioProcessor::DelayBypass, (float)delay\_gui\_components.BypassButton->getToggleState());

}

else if (buttonThatWasClicked == delay\_gui\_components.SynchButton)

{

synch\_mode\_on = !synch\_mode\_on;

delay\_gui\_components.DottedButton->setVisible(synch\_mode\_on);

delay\_gui\_components.SecondDottedButton->setVisible(synch\_mode\_on);

processor.setParameterNotifyingHost(ShimmerAudioProcessor::DelaySynch, (float)delay\_gui\_components.SynchButton->getToggleState());

if(synch\_mode\_on)

{

delay\_gui\_components.DelayLabel->setText("Denominator",NotificationType::dontSendNotification);

delay\_gui\_components.DelayKnob->setRange (1, 8, 1);

}

else

{

delay\_gui\_components.DelayLabel->setText("Delay", NotificationType::dontSendNotification);

delay\_gui\_components.DelayKnob->setRange (0, 2000, 5);

}

delay\_gui\_components.DelayKnob->setValue(processor.getDelayParam());

}

else if (buttonThatWasClicked == delay\_gui\_components.DottedButton)

{

processor.setParameterNotifyingHost(ShimmerAudioProcessor::DelayDot, (float)delay\_gui\_components.DottedButton->getToggleState());

}

else if (buttonThatWasClicked == delay\_gui\_components.SecondDottedButton)

{

processor.setParameterNotifyingHost(ShimmerAudioProcessor::DelaySecondDot, (float)delay\_gui\_components.SecondDottedButton->getToggleState());

}

else if (buttonThatWasClicked == reverb\_gui\_components.BypassButton)

{

processor.setParameterNotifyingHost(ShimmerAudioProcessor::ReverbBypass, (float)reverb\_gui\_components.BypassButton->getToggleState());

}

}

void ShimmerAudioProcessorEditor::visibilityChanged()

{

processor.RaiseUIUpdateFlag();

}

//[MiscUserCode] You can add your own definitions of your custom methods or any other code here...

void ShimmerAudioProcessorEditor::timerCallback(){

updateTimecodeDisplay (processor.lastPosInfo);

//exchange any data you want between UI elements and the plugin "ourProcessor"

// if(processor.NeedsUIUpdate()){

// BypassButton->setToggleState(1.0f == processor.getParameter(ShimmerAudioProcessor::MasterBypass), juce::dontSendNotification);

//SynchButton->setToggleState(1.0f == processor.getParameter(ShimmerAudioProcessor::Synch), juce::dontSendNotification);

//DottedButton->setToggleState(1.0f == processor.getParameter(ShimmerAudioProcessor::Dot), juce::dontSendNotification);

//SecondDottedButton->setToggleState(1.0f == processor.getParameter(ShimmerAudioProcessor::SecondDot), juce::dontSendNotification);

//DelayKnob->setValue(processor.getParameter(ShimmerAudioProcessor::Time), juce::dontSendNotification);

// FeedbackKnob->setValue(processor.getParameter(ShimmerAudioProcessor::Feedback), juce::dontSendNotification);

// MixKnob->setValue(processor.getParameter(ShimmerAudioProcessor::Mix), juce::dontSendNotification);

// processor.ClearUIUpdateFlag();

//}

}

//==============================================================================

// quick-and-dirty function to format a timecode string

static String timeToTimecodeString (double seconds)

{

const int millisecs = roundToInt (seconds \* 1000.0);

const int absMillisecs = std::abs (millisecs);

return String::formatted ("%02d:%02d:%02d.%03d",

millisecs / 360000,

(absMillisecs / 60000) % 60,

(absMillisecs / 1000) % 60,

absMillisecs % 1000);

}

const String ShimmerAudioProcessorEditor::setSyncLabel(int sl){

switch(sl){

case 0:

return "1";

case 1:

return "1/2";

case 2:

return "1/4";

case 3:

return "1/8";

case 4:

return "1/16";

default:

return "0";

}

}

// Updates the text in our position label.

void ShimmerAudioProcessorEditor::updateTimecodeDisplay (AudioPlayHead::CurrentPositionInfo pos)

{

MemoryOutputStream displayText;

displayText << String (pos.bpm, 2) << " bpm, "

<< pos.timeSigNumerator << '/' << pos.timeSigDenominator

<< " - " << timeToTimecodeString (pos.timeInSeconds);

if (pos.isRecording)

displayText << " (recording)";

else if (pos.isPlaying)

displayText << " (playing)";

delay\_gui\_components.timecodeDisplayLabel->setText (displayText.toString(), dontSendNotification);

}

Файл PluginProcessor.h

#ifndef PLUGINPROCESSOR\_H\_INCLUDED

#define PLUGINPROCESSOR\_H\_INCLUDED

#include "../JuceLibraryCode/JuceHeader.h"

#include "./Delay/Delay.h"

#include "./Reverb/Reverb.h"

//==============================================================================

/\*\*

\*/

class ShimmerAudioProcessor : public AudioProcessor

{

public:

//==============================================================================

ShimmerAudioProcessor();

~ShimmerAudioProcessor();

//==============================================================================

void prepareToPlay (double sampleRate, int samplesPerBlock) override;

void releaseResources() override;

#ifndef JucePlugin\_PreferredChannelConfigurations

bool isBusesLayoutSupported (const BusesLayout& layouts) const override;

#endif

//==============================================================================

void processBlock (AudioSampleBuffer&, MidiBuffer&) override;

void processBlockBypassed (AudioSampleBuffer& buffer, MidiBuffer& midiMessages) override;

//==============================================================================

AudioProcessorEditor\* createEditor() override;

bool hasEditor() const override;

//==============================================================================

const String getName() const override;

bool acceptsMidi() const override;

bool producesMidi() const override;

bool silenceInProducesSilenceOut() const;

double getTailLengthSeconds() const override;

//==============================================================================

int getNumPrograms() override;

int getCurrentProgram() override;

void setCurrentProgram (int index) override;

const String getProgramName (int index) override;

void changeProgramName (int index, const String& newName) override;

//==============================================================================

void getStateInformation (MemoryBlock& destData) override;

void setStateInformation (const void\* data, int sizeInBytes) override;

//==============================================================================

//Custom Methods, Params, and Public Data

int getNumParameters();

float getParameter (int index);

void setParameter (int index, float newValue);

//const String getParameterName (int index);

const String getParameterText (int index);

const String getInputChannelName (int channelIndex) const;

const String getOutputChannelName (int channelIndex) const;

bool isInputChannelStereoPair (int index) const;

bool isOutputChannelStereoPair (int index) const;

enum Parameters{

DelayBypass = 0,

DelayTime,

DelayFeedback,

DelayMix,

DelaySynch,

DelayDot,

DelaySecondDot,

ReverbBypass,

ReverbMix,

ReverbDecay,

InputGain,

OutputGain,

NumParams

};

bool NeedsUIUpdate(){return UIUpdateFlag;};

void ClearUIUpdateFlag(){UIUpdateFlag = false;};

void RaiseUIUpdateFlag(){UIUpdateFlag = true;};

//function added to calculate delay tap note values

float calculateDelayTap(float tap);

float getDelayParam();

AudioPlayHead::CurrentPositionInfo lastPosInfo;

private:

void updateCurrentTimeInfoFromHost();

float UserParams[NumParams];

double hostBPM;

Delay delay;

SReverb reverb;

bool UIUpdateFlag;

//==============================================================================

JUCE\_DECLARE\_NON\_COPYABLE\_WITH\_LEAK\_DETECTOR (ShimmerAudioProcessor)

};

#endif // PLUGINPROCESSOR\_H\_INCLUDED

Файл PluginProcessor.cpp

#include "PluginProcessor.h"

#include "PluginEditor.h"

//==============================================================================

ShimmerAudioProcessor::ShimmerAudioProcessor()

#ifndef JucePlugin\_PreferredChannelConfigurations

: AudioProcessor (BusesProperties()

#if ! JucePlugin\_IsMidiEffect

#if ! JucePlugin\_IsSynth

.withInput ("Input", AudioChannelSet::stereo(), true)

#endif

.withOutput ("Output", AudioChannelSet::stereo(), true)

#endif

),

#endif

delay(3500.0f)

{

//default values for plugin parameters

lastPosInfo.resetToDefault();

UserParams[DelayBypass] = delay.getByPass();

UserParams[DelayTime] = delay.getDelayTimeMS();

UserParams[DelayFeedback] = delay.getFeedback();

UserParams[DelayMix] = delay.getMix();

UserParams[DelaySynch] = 0.0f;

UserParams[DelayDot] = 0.0f;

UserParams[DelaySecondDot] = 0.0f;

UserParams[ReverbBypass] = reverb.getBypass();

UserParams[ReverbMix] = reverb.getMix();

UserParams[ReverbDecay] = reverb.getDecayFactor();

UserParams[InputGain] = 1.0f;

UserParams[OutputGain] = 1.0f;

//default host BPM to 120, default in most DAWs

hostBPM = 120;

}

ShimmerAudioProcessor::~ShimmerAudioProcessor()

{

}

//==============================================================================

const String ShimmerAudioProcessor::getName() const

{

return JucePlugin\_Name;

}

bool ShimmerAudioProcessor::acceptsMidi() const

{

#if JucePlugin\_WantsMidiInput

return true;

#else

return false;

#endif

}

bool ShimmerAudioProcessor::producesMidi() const

{

#if JucePlugin\_ProducesMidiOutput

return true;

#else

return false;

#endif

}

double ShimmerAudioProcessor::getTailLengthSeconds() const

{

return 0.0;

}

float ShimmerAudioProcessor::getParameter (int index)

{

//return the value of the parameter based on which indexed parameter is selected

if(index < NumParams)

return UserParams[index];

else

return 0.0f;

}

template <class T>

void swap(T& a, T& b)

{

T temp;

temp = a;

a = b;

b = temp;

}

float ShimmerAudioProcessor::getDelayParam()

{

return UserParams[DelayTime];

}

void ShimmerAudioProcessor::setParameter (int index, float newValue)

{

//set the parameter in the UI when user interacts and pass this

//value back through to the underlying Delay classes left and right

static float old\_delay = 1.0f;

switch (index) {

case DelayBypass:

UserParams[DelayBypass] = newValue;

delay.setByPass((bool)UserParams[DelayBypass]);

break;

case DelayTime:

UserParams[DelayTime] = newValue;

if(UserParams[DelaySynch] == 1.0f)

{

delay.setDelay(calculateDelayTap(UserParams[DelayTime]));

}

else

{

delay.setDelay(UserParams[DelayTime]);

}

break;

case DelayFeedback:

UserParams[DelayFeedback] = newValue;

//Feedback is received in 0 to +100

delay.setFeedback(UserParams[DelayFeedback]);

break;

case DelayMix:

UserParams[DelayMix] = newValue;

delay.setMix(UserParams[DelayMix]);

break;

case DelaySynch://???

swap(UserParams[DelayTime], old\_delay);

if(UserParams[DelaySynch] = newValue > 0)

{

delay.setDelay(calculateDelayTap(UserParams[DelayTime]));

}

else

{

delay.setDelay(UserParams[DelayTime]);

}

break;

case DelayDot:

UserParams[DelayDot] = newValue;

delay.setDelay(calculateDelayTap(UserParams[DelayTime]));

break;

case DelaySecondDot:

UserParams[DelaySecondDot] = newValue;

delay.setDelay(calculateDelayTap(UserParams[DelayTime]));

break;

case ReverbBypass:

UserParams[ReverbBypass] = newValue;

reverb.setBypass((bool)UserParams[ReverbBypass]);

break;

case ReverbMix:

UserParams[ReverbMix] = newValue;

reverb.setMix(UserParams[ReverbMix]);

break;

case ReverbDecay:

UserParams[ReverbDecay] = newValue;

reverb.setDecayFactor(UserParams[ReverbDecay]);

break;

case InputGain:

UserParams[InputGain] = newValue;

break;

case OutputGain:

UserParams[OutputGain] = newValue;

break;

default:

break;

}

}

//const String ShimmerAudioProcessor::getParameterName (int index)

//{

// switch (index) {

// case MasterBypass:

// return "MasterBypass";

// case Time:

// return "Delay";

// case Feedback:

// return "Feedback";

// case Mix:

// return "Mix";

// default:

// return String::empty;

// }

//}

const String ShimmerAudioProcessor::getParameterText (int index)

{

if(index >= 0 && index < NumParams){

return String(UserParams[index]);

}

else{

return String::empty;

}

}

const String ShimmerAudioProcessor::getInputChannelName (int channelIndex) const

{

return String (channelIndex + 1);

}

const String ShimmerAudioProcessor::getOutputChannelName (int channelIndex) const

{

return String (channelIndex + 1);

}

bool ShimmerAudioProcessor::isInputChannelStereoPair (int index) const

{

return true;

}

bool ShimmerAudioProcessor::isOutputChannelStereoPair (int index) const

{

return true;

}

bool ShimmerAudioProcessor::silenceInProducesSilenceOut() const

{

return false;

}

int ShimmerAudioProcessor::getNumParameters()

{

return NumParams;

}

//-----------------------------------------

int ShimmerAudioProcessor::getNumPrograms()

{

return 1; // NB: some hosts don't cope very well if you tell them there are 0 programs,

// so this should be at least 1, even if you're not really implementing programs.

}

int ShimmerAudioProcessor::getCurrentProgram()

{

return 0;

}

void ShimmerAudioProcessor::setCurrentProgram (int index)

{

}

const String ShimmerAudioProcessor::getProgramName (int index)

{

return String();

}

void ShimmerAudioProcessor::changeProgramName (int index, const String& newName)

{

}

//==============================================================================

void ShimmerAudioProcessor::prepareToPlay (double sampleRate, int samplesPerBlock)

{

// Use this method as the place to do any pre-playback

// initialisation that you need..

}

void ShimmerAudioProcessor::releaseResources()

{

// When playback stops, you can use this as an opportunity to free up any

// spare memory, etc.

}

#ifndef JucePlugin\_PreferredChannelConfigurations

bool ShimmerAudioProcessor::isBusesLayoutSupported (const BusesLayout& layouts) const

{

#if JucePlugin\_IsMidiEffect

ignoreUnused (layouts);

return true;

#else

// This is the place where you check if the layout is supported.

// In this template code we only support mono or stereo.

if (layouts.getMainOutputChannelSet() != AudioChannelSet::mono()

&& layouts.getMainOutputChannelSet() != AudioChannelSet::stereo())

return false;

// This checks if the input layout matches the output layout

#if ! JucePlugin\_IsSynth

if (layouts.getMainOutputChannelSet() != layouts.getMainInputChannelSet())

return false;

#endif

return true;

#endif

}

#endif

//processing

// The I/O buffers are interleaved depending on the number of channels. If NumChannels = 2, then the

// buffer is L/R/L/R/L/R etc...

void ShimmerAudioProcessor::processBlock (AudioSampleBuffer& buffer, MidiBuffer& midiMessages)

{

// int numSamples = buffer.getNumSamples(); //THIS IS NUM SAMPLES PER CHANNEL

// for(int channel = 0; channel < getNumInputChannels(); channel++){

// float\* channelData = buffer.getWritePointer(channel);

// for(int i = 0; i < numSamples; i+=2){

// channelData[i] = channelData[i] \* UserParams[InputGain];

//channelData[i] = delay.next(channelData[i] \* UserParams[InputGain]);

// }

// }

//get the host BPM and sync playhead to it

juce::AudioPlayHead::CurrentPositionInfo result;

juce::AudioPlayHead\* jap = getPlayHead();

jap->getCurrentPosition(result);

if(result.bpm != 0)

this->hostBPM = result.bpm;

int numSamples = buffer.getNumSamples(); //THIS IS NUM SAMPLES PER CHANNEL

float\* channelDataLeft = buffer.getWritePointer(0); //buffer.getSampleData(0);

float\* channelDataRight = buffer.getWritePointer(1);

const float\* inChannelDataLeft = buffer.getReadPointer(0);

const float\* inChannelDataRight = buffer.getReadPointer(1);

float out\_l = 0.0f, out\_r = 0.0f;

for(int i = 0; i < numSamples; i++)

{

channelDataLeft[i] = inChannelDataLeft[i] \* UserParams[InputGain];

channelDataRight[i] = inChannelDataRight[i] \* UserParams[InputGain];

out\_l = delay.next(channelDataLeft[i]);

out\_l = reverb.next(out\_l);

channelDataLeft[i] = out\_l \* UserParams[OutputGain];

channelDataRight[i] = channelDataLeft[i];

}

for (int i = getNumInputChannels(); i < getNumOutputChannels(); ++i)

{

buffer.clear (i, 0, buffer.getNumSamples());

}

updateCurrentTimeInfoFromHost();

}

void ShimmerAudioProcessor::processBlockBypassed (AudioSampleBuffer& buffer, MidiBuffer& midiMessages){

int numSamples = buffer.getNumSamples(); //THIS IS NUM SAMPLES PER CHANNEL

for(int channel = 0; channel < getNumInputChannels(); channel++){

float\* channelData = buffer.getWritePointer(channel);

for(int i = 0; i < numSamples; i++){

channelData[i] = channelData[i];

}

}

return;

}

//==============================================================================

bool ShimmerAudioProcessor::hasEditor() const

{

return true; // (change this to false if you choose to not supply an editor)

}

AudioProcessorEditor\* ShimmerAudioProcessor::createEditor()

{

return new ShimmerAudioProcessorEditor (\*this);

}

//==============================================================================

void ShimmerAudioProcessor::getStateInformation (MemoryBlock& destData)//???

{

// You should use this method to store your parameters in the memory block.

// You could do that either as raw data, or use the XML or ValueTree classes

// as intermediaries to make it easy to save and load complex data.

// XmlElement root("Root");

// XmlElement \*el;

// el = root.createNewChildElement("MasterBypass");

// el->addTextElement(String(UserParams[MasterBypass]));

// el = root.createNewChildElement("Delay");

// el->addTextElement(String(UserParams[Time]));

// el = root.createNewChildElement("Feedback");

// el->addTextElement(String(UserParams[Feedback]));

// el = root.createNewChildElement("Mix");

// el->addTextElement(String(UserParams[Mix]));

//el = root.createNewChildElement("Synch");

// el->addTextElement(String(UserParams[Synch]));

//el = root.createNewChildElement("Dot");

// el->addTextElement(String(UserParams[Dot]));

//el = root.createNewChildElement("SecondDot");

// el->addTextElement(String(UserParams[SecondDot]));

// //el = root.createNewChildElement("Tap1Delay");

// //el->addTextElement(String(UserParams[Tap1Delay]));

// //el = root.createNewChildElement("Tap1Level");

// //el->addTextElement(String(UserParams[Tap1Level]));

// copyXmlToBinary(root, destData);

}

void ShimmerAudioProcessor::setStateInformation (const void\* data, int sizeInBytes)//???

{

// You should use this method to restore your parameters from this memory block,

// whose contents will have been created by the getStateInformation() call.

// XmlElement \*pRoot = getXmlFromBinary(data, sizeInBytes);

// if(pRoot != NULL){

// forEachXmlChildElement((\*pRoot), pChild){

// if(pChild->hasTagName("MasterBypass")){

// String text = pChild->getAllSubText();

// setParameter(MasterBypass, text.getFloatValue());

// }

// if(pChild->hasTagName("Delay")){

// String text = pChild->getAllSubText();

// setParameter(Delay, text.getFloatValue());

// }

// if(pChild->hasTagName("Feedback")){

// String text = pChild->getAllSubText();

// setParameter(Feedback, text.getFloatValue());

// }

// if(pChild->hasTagName("Mix")){

// String text = pChild->getAllSubText();

// setParameter(Mix, text.getFloatValue());

// }

//if(pChild->hasTagName("Synch")){

// String text = pChild->getAllSubText();

// setParameter(Synch, text.getFloatValue());

// }

//if(pChild->hasTagName("Dot")){

// String text = pChild->getAllSubText();

// setParameter(Dot, text.getFloatValue());

// }

//if(pChild->hasTagName("SecondDot")){

// String text = pChild->getAllSubText();

// setParameter(SecondDot, text.getFloatValue());

// }

// //if(pChild->hasTagName("Tap1Delay")){

// // String text = pChild->getAllSubText();

// // setParameter(Tap1Delay, text.getFloatValue());

// //}

// //if(pChild->hasTagName("Tap1Level")){

// // String text = pChild->getAllSubText();

// // setParameter(Tap1Level, text.getFloatValue());

// //}

// }

// delete pRoot;

// }

// UIUpdateFlag = true; //Request UI update

}

//-------------------------------------------------------------------------

// calculateDelayTap

//

// function that calculates the Tap for given note values

// 0 = 1 note

// 1 = 1/2 note

// 2 = 1/4 note

// 3 = 1/8 note

// 4 = 1/16 note

// then sends the result back

//-------------------------------------------------------------------------

float ShimmerAudioProcessor::calculateDelayTap(float tp)

{

float result = .0f;

float temp = .0f;

int tap = static\_cast<int>(tp);//!!!

result = temp = 60\*4\*1000/(hostBPM\*(tap));

if(UserParams[DelayDot])

result += temp/2;

if(UserParams[DelaySecondDot])

result += temp/4;

return result;

}

void ShimmerAudioProcessor::updateCurrentTimeInfoFromHost()

{

if (AudioPlayHead\* ph = getPlayHead())

{

AudioPlayHead::CurrentPositionInfo newTime;

if (ph->getCurrentPosition (newTime))

{

lastPosInfo = newTime; // Successfully got the current time from the host..

if(lastPosInfo.bpm == 0)

lastPosInfo.bpm = this->hostBPM = 120;

return;

}

}

// If the host fails to provide the current time, we'll just reset our copy to a default..

lastPosInfo.resetToDefault();

}

//==============================================================================

// This creates new instances of the plugin..

AudioProcessor\* JUCE\_CALLTYPE createPluginFilter()

{

return new ShimmerAudioProcessor();

}

Файл Delay.h

#ifndef DELAY\_H\_INCLUDED

#define DELAY\_H\_INCLUDED

#include "../DelayLine/DelayLine.h"

class Delay {

public:

Delay(float d\_ms\_max = 0.0f, int sr = 44100, float d\_ms = 800.0f, float feedback = 0.0f, float mixLevel = 0.3f, bool bp = false);

~Delay();

float next(float in);

float getDelayTimeMS();

float getFeedback();

float getMix();

bool getByPass();

void setFeedback(float f\_pct);

void setMix(float m\_pct);

void setByPass(bool bp);

void setDelay(float d);

private:

DelayLine delay\_line;

float feedback, mixLevel;

bool delay\_bypass;

};

#endif // DELAY\_H\_INCLUDED

Файл Delay.cpp

#include "Delay.h"

Delay::Delay(float d\_ms\_max, int sr, float d\_ms, float fb, float mix, bool bp)

: delay\_line(d\_ms\_max, sr, d\_ms)

{

feedback = fb;

mixLevel = mix;

delay\_bypass = bp;

}

Delay::~Delay(){}

float Delay::getDelayTimeMS() {return delay\_line.getDelayTimeMS();}

float Delay::getFeedback(){return feedback;}

float Delay::getMix(){return mixLevel;}

bool Delay::getByPass(){return delay\_bypass;}

void Delay::setFeedback(float f){feedback = f;}

void Delay::setMix(float m){mixLevel = m;}

void Delay::setByPass(bool bp){delay\_bypass = bp;}

float Delay::next(const float in)

{

if(delay\_bypass)

return in;

float value\_to\_read = delay\_line.readDelay();

delay\_line.writeDelay(in + feedback\*value\_to\_read);

return mixLevel \* value\_to\_read + (1-mixLevel) \* in;

}

void Delay::setDelay(float d) { delay\_line.setDelay(d); } //!!!

Файл DelayLine.h

//#define NDEBUG // в начале файла исходного кода, перед включением заголовочного файла <cassert>

#ifndef DelayLine\_H\_INCLUDED

#define DelayLine\_H\_INCLUDED

#include <algorithm>

#include <cmath>

#include <cstdlib>

#include <iostream>

#include <cassert>

class DelayLine{

public:

//constructor / destructor

DelayLine(float d\_ms\_max = 0.0f, int sr = 44100, float d\_ms = 0.0f);

~DelayLine();

//getters

float getDelayTimeMS();

float getMaxDelayTimeMS();

//setters

void setDelayTimeMS(float time\_ms);

void setDelay(float d\_ms);

//business functions

float readDelay();

void writeDelay(float in);

float next(const float in);

void resetDelayLine();

void clearBuffer();

private:

int sample\_rate;

int writePos, readPos, MAX\_DELAY\_SAMPLES;

float delay\_ms, delay\_samples, fraction, MAX\_DELAY\_MS;

float \*buffer;

};

#endif // DelayLine\_H\_INCLUDED

Файл DelayLine.cpp

#include "DelayLine.h"

//helper function

//-------------------------------------------------------------------------

// numSamplesFromMSf :

// Determine the number of samples from the number of milliseconds delay

// passed to function

//-------------------------------------------------------------------------

inline float numSamplesFromMSf(const int sr, const float d\_ms){

return sr \* d\_ms \* .001;

}

//----------------------------------------------------------------------

// Linear Interpolation Function

// x1 : weighting 1

// x2 : weighting 2

// y1 : output y(n) at read index in buffer

// y2 : output y(n-1) at read index minus 1 in buffer

// x : fractional value between samples

//----------------------------------------------------------------------

inline float linInterp(float x1, float x2, float y1, float y2, float x){

float denom = x2 - x1;

if(denom == 0)

return y1; // should not ever happen

// calculate decimal position of x

float dx = (x - x1)/(x2 - x1);

// use weighted sum method of interpolating

float result = dx\*y2 + (1-dx)\*y1;

return result;

}

//constructor

DelayLine::DelayLine(float d\_ms\_max, int sr, float d\_ms){

assert(d\_ms <= d\_ms\_max);//check bound on delay time

sample\_rate = sr;

buffer = NULL;

readPos = writePos = 0;

delay\_samples = 0.0f;

delay\_ms = d\_ms;

MAX\_DELAY\_SAMPLES = ceil(numSamplesFromMSf(sample\_rate, d\_ms\_max));

MAX\_DELAY\_MS = MAX\_DELAY\_SAMPLES \* 1000.0f / sample\_rate;

//number of delay samples

float delay\_samplesf = numSamplesFromMSf(sample\_rate, d\_ms);

delay\_samples = floor(delay\_samplesf);

//storing fractional delay time, will be interpolated

fraction = delay\_samplesf - delay\_samples;

buffer = new float[MAX\_DELAY\_SAMPLES];

memset(buffer, 0, MAX\_DELAY\_SAMPLES\*sizeof(float));

}

//destructor

DelayLine::~DelayLine(){

delete[] buffer;

}

//getters

float DelayLine::getDelayTimeMS(){return delay\_ms;}

float DelayLine::getMaxDelayTimeMS(){return MAX\_DELAY\_MS;}

void DelayLine::setDelayTimeMS(float time\_ms){

assert(time\_ms <= MAX\_DELAY\_MS);//check bound on delay time

float delay\_samplesf = numSamplesFromMSf(sample\_rate, time\_ms);

delay\_samples = floor(delay\_samplesf);

fraction = delay\_samplesf - delay\_samples;//storing fractional delay time, output will be interpolated

readPos = writePos - (int)delay\_samples;

if(readPos < 0)

readPos += MAX\_DELAY\_SAMPLES;

}

void DelayLine::setDelay(float d){

//receiving the delay value through here in milliseconds 0 to 2000

delay\_ms = d;

setDelayTimeMS(delay\_ms);

}

float DelayLine::next(const float in){

//input

float xn = in;

float yn;

if(delay\_samples == 0)

yn = xn;

else

yn = readDelay();

writeDelay(xn);

float out = yn;

return out;

}

float DelayLine::readDelay(){

float yn = buffer[readPos];

int readPos\_minus1 = readPos - 1;

if(readPos\_minus1 < 0)

readPos\_minus1 = MAX\_DELAY\_SAMPLES - 1; //MAX\_DELAY\_SAMPLES - 1 is the last location of the buffer

//get y(n-1)

float yn\_minus1 = buffer[readPos\_minus1];

readPos++;

if(readPos >= MAX\_DELAY\_SAMPLES)

readPos = 0;

//perform linear interpolation of : (0,yn) and (1,yn\_minus1) by the ammount of fractional delay(fraction)

return linInterp(0, 1, yn, yn\_minus1, fraction);

}

void DelayLine::writeDelay(float in)

{

buffer[writePos] = in;

writePos++;

if(writePos >= MAX\_DELAY\_SAMPLES)

writePos = 0;

}

void DelayLine::resetDelayLine(){

if(buffer)

delete [] buffer;

buffer = new float[MAX\_DELAY\_SAMPLES];

clearBuffer();

readPos = writePos = 0;

setDelayTimeMS(delay\_ms);

}

void DelayLine::clearBuffer(){

if(buffer)

memset(buffer, 0, MAX\_DELAY\_SAMPLES\*sizeof(float));

}

Файл Allpass.h

#ifndef ALLPASS\_H\_INCLUDED

#define ALLPASS\_H\_INCLUDED

#include "../../DelayLine/DelayLine.h"

class Allpass{

public:

//constructor

Allpass(const int sr, const float d\_ms, const float d\_ms\_max, const float g);

~Allpass();

//getters

float getGain();

float getDelayTimeMS();

//setters

void setGain(const float g);

void setDelayTimeMS(float time\_ms);

//business methods

float next(const float in);

private:

float gain;

DelayLine \*delay;

};

#endif // ALLPASS\_H\_INCLUDED

Файл Allpass.cpp

#include "AllPass.h"

//constructor

Allpass::Allpass(const int sr, const float d\_ms, const float d\_ms\_max, const float g){

gain = g;

delay = new DelayLine(sr, d\_ms, d\_ms\_max);

}

Allpass::~Allpass(){

delete delay;

}

//getters

float Allpass::getGain(){return gain;}

float Allpass::getDelayTimeMS(){return delay->getDelayTimeMS();}

//setters

void Allpass::setGain(const float g){gain = g;}

void Allpass::setDelayTimeMS(float time\_ms){return delay->setDelayTimeMS(time\_ms);}

//business methods

float Allpass::next(const float in){

float dL = delay->readDelay();

float fW = in + (gain\*dL);

delay->writeDelay(fW);

float out = -gain\*fW + dL;

return out;

}

Файл Comb.h

#ifndef COMB\_H\_INCLUDED

#define COMB\_H\_INCLUDED

#include "../../DelayLine/DelayLine.h"

#include "../LowPass/LowPass.h"

class Comb{

public:

//constructor / destructor

Comb(const int sr, const float d\_ms, const float d\_ms\_max, const float g);

~Comb();

//getters

float getGain();

float getDelayTimeMS();

//setters

void setGain(const float g);

void setDelayTimeMS(float time\_ms);

//business methods

float next(const float in);

private:

float gain;

DelayLine \*delay;

Lowpass \*lpFilter;

};

#endif // COMB\_H\_INCLUDED

Файл Comb.cpp

#include "Comb.h"

//constructor

Comb::Comb(const int sr, const float d\_ms, const float d\_ms\_max, const float g){

gain = g;

delay = new DelayLine(d\_ms\_max, sr, d\_ms);

lpFilter = new Lowpass(sr, 10000.0f);

}

// destructor

Comb::~Comb(){

delete delay;

delete lpFilter;

}

//getters

float Comb::getGain(){return gain;}

float Comb::getDelayTimeMS(){return delay->getDelayTimeMS();}

//setters

void Comb::setGain(const float g){gain = g;}

void Comb::setDelayTimeMS(float time\_ms){return delay->setDelayTimeMS(time\_ms);}

//business methods

float Comb::next(const float in){

float dL = delay->readDelay();

float lpRetVal = lpFilter->next(dL);

float dLW = in + lpRetVal\*gain;

delay->writeDelay(dLW);

return dL;

}

Файл Lowpass.h

#ifndef LOWPASS\_H\_INCLUDED

#define LOWPASS\_H\_INCLUDED

#include <cmath>

class Lowpass{

public:

//constructor

Lowpass(const int sr, const float cf\_hz);

//getters

float getCutoff();

//setters

void setCutoff(const int sr, const float cf\_hz);

//business methods

float next(const float in);

private:

float cutoff, coef, prev;

};

#endif // LOWPASS\_H\_INCLUDED

Файл Lowpass.cpp

#include "LowPass.h"

const float M\_PI = 3.14159365359f;

//constructor

Lowpass::Lowpass(const int sr, const float cf\_hz){

prev = 0;

cutoff = cf\_hz;

float costh = 2.0 - cos(2.0 \* M\_PI \* cutoff / sr);

coef = sqrt(costh \* costh - 1.0) - costh;

}

//getters

float Lowpass::getCutoff(){return cutoff;}

//setters

void Lowpass::setCutoff(const int sr, const float cf\_hz){

cutoff = cf\_hz;

float costh = 2.0 - cos(2.0 \* M\_PI \* cutoff / sr);

coef = sqrt(costh \* costh - 1.0) - costh;

}

//business methods

float Lowpass::next(const float in){

prev = in \* (1 + coef) - (prev \* coef);

return prev;

}

Файл Images.h

/\*

==============================================================================

This is an automatically generated GUI class created by the Projucer!

Be careful when adding custom code to these files, as only the code within

the "//[xyz]" and "//[/xyz]" sections will be retained when the file is loaded

and re-saved.

Created with Projucer version: 4.3.1

------------------------------------------------------------------------------

The Projucer is part of the JUCE library - "Jules' Utility Class Extensions"

Copyright (c) 2015 - ROLI Ltd.

==============================================================================

\*/

#ifndef \_\_JUCE\_HEADER\_61C39FB97870974\_\_

#define \_\_JUCE\_HEADER\_61C39FB97870974\_\_

//[Headers] -- You can add your own extra header files here --

#include "../../JuceLibraryCode/JuceHeader.h"

//[/Headers]

//==============================================================================

/\*\*

//[Comments]

An auto-generated component, created by the Projucer.

Describe your class and how it works here!

//[/Comments]

\*/

class Images : public Component

{

public:

//==============================================================================

Images ();

~Images();

//==============================================================================

//[UserMethods] -- You can add your own custom methods in this section.

//[/UserMethods]

void paint (Graphics& g) override;

void resized() override;

// Binary resources:

static const char\* background\_jpg;

static const int background\_jpgSize;

private:

//[UserVariables] -- You can add your own custom variables in this section.

//[/UserVariables]

//==============================================================================

//==============================================================================

JUCE\_DECLARE\_NON\_COPYABLE\_WITH\_LEAK\_DETECTOR (Images)

};

//[EndFile] You can add extra defines here...

//[/EndFile]

#endif // \_\_JUCE\_HEADER\_61C39FB97870974\_\_

Файл Reverb.h

#ifndef Reverb\_H\_INCLUDED

#define Reverb\_H\_INCLUDED

#include "../DelayLine/DelayLine.h"

#include "../Filters/Comb/Comb.h"

#include "../Filters/AllPass/AllPass.h"

#include "../Filters/LowPass/LowPass.h"

#define NUM\_COMBS 4

#define NUM\_ALLPASSES 2

#define NUM\_LOWPASSES 1

class SReverb{

public:

//constructor / destructor

SReverb(const int sr = 44100, const float rt60 = 3.0f,

const float cDelay1 = 29.7, const float cDelay2 = 37.1, const float cDelay3 = 41.1, const float cDelay4 = 43.7,

const float aDelay1 = 5.0, const float aDelay2 = 1.7,

const float aGain1 = 0.707, const float aGain2 = 0.5,

const float lCutoff1 = 2300.0f, const float mx = 0.3f); //const float lCutoff1 = 2300.0f

~SReverb();

//getters

float getDecayFactor();

float getCombDelay(const int id);

float getAllpassDelay(const int id);

float getAllpassGain(const int id);

float getLowpassCutoff(const int id);

bool getBypass();

float getMix();

//setters

void setDecayFactor(const float df);

void setCombDelay(const int id, const float sr, const float d\_ms);

void setAllpassGain(const int id, const float g);

void setAllpassDelay(const int id, const int sr, const float d\_ms);

void setLowpassCutoff(const int id, const int sr, const float cf\_hz);

void setBypass(bool bp);

void setMix(float value);

//business methods

float next(const float in);

private:

float decayFactor, ALLPASS\_GAIN\_LIMIT;//to keep the allpasses from exploding

float mix;

bool bypass;

Comb \*combs[NUM\_COMBS];

Allpass \*allpasses[NUM\_ALLPASSES];

Lowpass \*lowpasses[NUM\_LOWPASSES];

};

#endif // SReverb\_H\_INCLUDED

Файл Reverb.cpp

#include "Reverb.h"

#define NUM\_COMBS 4

#define NUM\_ALLPASSES 2

//helper functions

inline float calcCombGain(const float d\_ms, const float rt60){

return pow(10.0, ((-3.0 \* d\_ms) / (rt60 \* 1000.0)));

//(((return pow(10.0, (-3.0 \* (d\_ms\*44100\*.001) \* (1.0/44100)) / (rt60 \* 1000.0));

}

inline float calcAPGain(const float d\_ms, const float rt60){

return pow(10.0, ((-3.0 \* d\_ms) / (rt60 \* 1000.0)));

}

//constructor / destructor

SReverb::SReverb(const int sr, const float rt60,

const float cDelay1, const float cDelay2, const float cDelay3, const float cDelay4,

const float aDelay1, const float aDelay2, const float aGain1, const float aGain2, const float lCutoff1, const float mx){ //const float lCutoff1

ALLPASS\_GAIN\_LIMIT = 0.107f; //0.707f;

decayFactor = rt60;

float d\_ms, d\_ms\_max = 100.0f, gain;

d\_ms = cDelay1;

bypass = false;

mix = mx;

gain = calcCombGain(d\_ms, decayFactor);

combs[0] = new Comb(sr, d\_ms, d\_ms\_max, gain);

setCombDelay(0,sr,d\_ms);

d\_ms = cDelay2;

gain = calcCombGain(d\_ms, decayFactor);

combs[1] = new Comb(sr, d\_ms, d\_ms\_max, gain);

setCombDelay(1,sr,d\_ms);

d\_ms = cDelay3;

gain = calcCombGain(d\_ms, decayFactor);

combs[2] = new Comb(sr, d\_ms, d\_ms\_max, gain);

setCombDelay(2,sr,d\_ms);

d\_ms = cDelay4;

gain = calcCombGain(d\_ms, decayFactor);

combs[3] = new Comb(sr, d\_ms, d\_ms\_max, gain);

setCombDelay(3,sr,d\_ms);

d\_ms\_max = 20.0f;

allpasses[0] = new Allpass(sr, aDelay1, d\_ms\_max, aGain1);

allpasses[1] = new Allpass(sr, aDelay2, d\_ms\_max, aGain2);

lowpasses[0] = new Lowpass(sr, lCutoff1);

//std::cout << "SReverb constructor called" << std::endl;

}

SReverb::~SReverb(){

//std::cout << "SReverb destructor called" << std::endl;

for(int i = 0; i < NUM\_COMBS; i++){

delete combs[i];

}

for(int i = 0; i < NUM\_ALLPASSES; i++){

delete allpasses[i];

}

}

//getters

float SReverb::getMix() {return mix;}

float SReverb::getDecayFactor(){return decayFactor;}

float SReverb::getCombDelay(const int id){return combs[id]->getDelayTimeMS();}

float SReverb::getAllpassDelay(const int id){return allpasses[id]->getDelayTimeMS();}

float SReverb::getAllpassGain(const int id){return allpasses[id]->getGain();}

float SReverb::getLowpassCutoff(const int id){return lowpasses[id]->getCutoff();}

bool SReverb::getBypass(){return bypass;}

//setters

void SReverb::setDecayFactor(const float df){

decayFactor = df;

for(int i = 0; i < NUM\_COMBS; i++){

combs[i]->setGain(calcCombGain(combs[i]->getDelayTimeMS(), decayFactor));

}

};

void SReverb::setCombDelay(const int id, const float sr, const float d\_ms){

combs[id]->setGain(calcCombGain(d\_ms, decayFactor));

combs[id]->setDelayTimeMS(d\_ms);

}

void SReverb::setAllpassGain(const int id, const float g){allpasses[id]->setGain(g \* ALLPASS\_GAIN\_LIMIT);}

void SReverb::setAllpassDelay(const int id, const int sr, const float d\_ms){allpasses[id]->setDelayTimeMS(d\_ms);}

void SReverb::setLowpassCutoff(const int id, const int sr, const float cf\_hz){lowpasses[id]->setCutoff(sr, cf\_hz);}

void SReverb::setBypass(bool bp){bypass = bp;}

//business methods

float SReverb::next(const float in){

if(bypass)

return in;

float out = 0.0f;

for(int i = 0; i < NUM\_COMBS; i++){

out += combs[i]->next(in \* 0.125f); //scale down to avoid clipping

//out+= combs[i]->next(in \* 0.5);

}

float passOut = 0.0f;

float passOut2 = 0.0f;

passOut = allpasses[0]->next(out);

passOut2 = allpasses[1]->next(passOut);

return mix \* passOut2 \* NUM\_COMBS + (1.0f - mix) \* in;

//return passOut2;

}

void SReverb::setMix(float value)

{

mix = value;

}

# ПРИЛОЖЕНИЕ Б

(обязательное)

Спецификация программного дипломного проекта

# ПРИЛОЖЕНИЕ В

(обязательное)

Ведомость документов