Реализация сайдчейн-технологии в VST-плагине

Кошелев Егор Евгеньевич

Санкт-Петербургский государственный архитектурно-строительный университет, Санкт-Петербург, Россия

АННОТАЦИЯ

Введение: Приведено описание аудио плагина с собственным графическим интерфейсом пользователя. В плагине реализована сайдчейн-технология, позволяющая фильтровать нижние частоты исходного сигнала в зависимости от уровня управляющего сигнала. Для связи уровня фильтрации исходного сигнала с уровнем громкости управляющего сигнала был разработан соответствующий алгоритм. Для удобства работы с плагином был создан графический пользовательский интерфейс в виде детекторов уровней громкости исходного и управляющего сигнала. Интерфейс содержит ручки управления уровнями громкостей этих сигналов, ручки детектора уровня исходного сигнала для фильтрации управляющего сигнала, а также ручку плавности применения фильтрации. Разработанный плагин имеет формат VST и может использоваться практически в любой современной цифровой звуковой рабочей станции. Для создания плагина использовалась среда разработки Microsoft Visual Studio и библиотека WDL-OL. В качестве цифровой звуковой рабочей станции для загрузки готового плагина использовалась цифровая аудиостанция Reaper.

КЛЮЧЕВЫЕ СЛОВА: цифровая обработка звука; VST-плагин; цифровая рабочая станция; сайдчейн-технология; объектно-ориентированное программирование; графический интерфейс пользователя.

Введение

В сфере обработки звуковой информации физические устройства или специализированные программные комплексы решают большой спектр задач. Основными среди них являются частотная, динамическая, пространственная и модуляционная обработка звука, а также различные искажения сигнала. Выбор конкретных устройств или программных решений зависит от сферы их применения. Она может заключаться в сведении дорожек, например, музыкальных композиций, кинематографа или телепередач. Также это может быть проведение живых выступлений, эфиров, конференций, где обработка звука происходит в реальном времени.

Одной из важнейших технологий обработки звука во всех вышеописанных сферах является технология Сайдчейн. Кратко ее суть заключается в обработке некоторого исходного сигнала, используя при этом управляющий сигнал, различные параметры которого напрямую влияют на характер обработки исходного сигнала. Изначально сайдчейн-технология была придумана во времена аналоговых физических устройств, но с развитием компьютерных цифровых технологий и появлением программного обеспечения и плагинов по обработке звука, данная технология практически полностью производится в цифровом виде [1].

Постановка задачи

Обработка цифрового звука производится в специализированных цифровых звуковых рабочих станциях с использованием программных модулей (плагинов) по обработке, синтезу и анализу цифрового звука. Аудиоплагины разрабатываются различными компаниями по всему миру, и огромной проблемой является то, что отечественные разработки в этой сфере практически не ведутся. В особенности данная проблема раскрывается в текущий момент, когда многие зарубежные компании просто ограничивают доступ к приобретению программных средств в нашей стране, а также отзывают уже купленные лицензии.

Помимо этого, сфера разработки аудиоплагинов включает в себя творческую составляющую, позволяя экспериментировать с методами и алгоритмами обработки звука для получения новых видов обработки и, как результат, нового звучания. Поэтому исследование области обработки и генерации цифровых звуковых сигналов, изучение способов разработки программного обеспечения для работы со звуком и реализация полученных знаний в виде конечных программных продуктов несомненно являются важнейшими задачами при проектировании программных комплексов для работы с цифровым звуком.

Научной новизной данной работы является его уникальность в среде отечественных научных трудов, посвященных изучению аудио плагинов в целом и их разработке, в особенности теме реализации сайдчейн-технологии, так как отечественные исследования и разработки в данной сфере практически не ведутся. Об этом может свидетельствовать наличие лишь статей с обзорами конкретных зарубежных плагинов или цифровых звуковых рабочих станций, а также нескольких публикаций, написанных лично автором данной статьи, посвященных основным аспектам разработки VST-плагина эффекта и синтезатора.

Целью данной статьи является разработка аудиоплагина в формате VST и реализация в нем сайдчейн-технологии, позволяющей фильтровать исходный сигнал в зависимости от управляющего сигнала.

Сфера применения технологии

Сама сайдчейн-технология уходит своей историей примерно в 1930-е годы. Звукорежиссер кино Дуглас Ширер искал способ борьбы с сибилянтами (резкие и громкие звуки «с») в записях голосов. Тогда он задумал компрессор с побочной цепью сигналов, по которой шла копия исходного сигнала с голосом [2]. На копии сигнала была произведена эквализация, фильтрующая частоты в том диапазоне, где сильнее всего звучат сибилянты. В результате, когда в сигнале побочной цепи появлялись характерные громкие звуки, компрессор начинал свою работу, делая тише громкость исходного сигнала с голосом, занижая громкость сибилянтов в звуке. Копия сигнала с побочной цепи не воспроизводилась, так как она использовалась только в качестве триггера для компрессора сигнала.

В течение десятилетий, последовавших за изобретением сайдчейн-технологии, она использовалась в основном как эффект для понижения громкости музыкального сопровождения, начиная от теле- и радиоэфиров, конференций, и заканчивая продуктовыми магазинами, понижая громкость музыки при различных объявлениях от персонала. Данный эффект также имеет название «дакинг».

В 80-е – 90-е годы XX-го века огромную популярность начала приобретать электронная музыка. Вместе с этим в сфере создания музыки появилась тенденция делать музыку все более и более громкой. Этому явлению даже был дан термин «война громкостей». Усилить громкость сигнала, который уже находится в допустимых пределах носителя, можно было только лишь компрессируя его с последующим повышением громкости. Но данный способ имеет ряд больших недостатков в виде потери динамики, искажений и клиппирования. Одним из решений этой проблемы примерно в 90-х прошлого века стало применение сайдчейн-технологии [3]. Заключалось оно в том, что один звук являлся триггером для управления громкостью другого звука. Как только первый звук превышал заданный порог громкости, второй звук занижался по громкости, в результате чего выстраивался определенный баланс инструментов.

В современном мире сайдчейн-технология применяется до сих пор в тех же направлениях. Однако сейчас практически всегда частично или полностью работа идет с цифровым звуком, и в большей части для решения задач, в которых необходима данная технология, используются программные решения в виде аудиоплагинов, подключаемых к звуковым рабочим станциям [4]. При этом плагин с сайдчейн-технологией применяется к исходному сигналу, например, звуку басового инструмента. А управляющий сигнал в виде, к примеру, большого барабана, посылается на вход плагина с помощью средств «посыл-возврат», которые есть во всех современных цифровых рабочих станциях. В результате во время удара большого барабана громкость басового инструмента будет занижаться.

Принцип работы технологии. Сайдчейн-технология позволяет расширить функциональность алгоритмов обработки сигнала, изменяя в реальном времени параметры обработки посредством использования стороннего (управляющего) сигнала, влияя тем самым на характер вносимых в исходный сигнал изменений. Суть ее работы заключается в том, что на вход устройства или плагина, в котором реализована данная технология, помимо исходного сигнала подается также управляющий сигнал со своими характеристиками.

Зачастую в качестве основной характеристики управляющего сигнала выступает его громкость [5]. Устройство распознает уровень громкости управляющего сигнала в конкретный момент времени с помощью детектора. Значение уровня громкости передается в блок

эффектов обработки исходного сигнала. В зависимости от его значения будет изменяться степень применения конкретного эффекта обработки. Как именно она меняется зависит от конкретных целей, на которые направлено устройство или плагин, и реализуется с помощью алгоритмов, описывающих связь эффектов обработки и громкость управляющего сигнала.

Устройство или плагин с данной технологией никаким образом не влияет на управляющий сигнал. Изменения происходят лишь с исходным сигналом [6]. Управляющий сигнал либо посылается в физическое устройство непосредственно по проводам, либо попадает на специальный вход плагина с помощью стандартных функций посыл-возврат любой цифровой звуковой рабочей станции. После обработки управляющий сигнал не попадает на выход плагина или устройства. На выходе получается лишь обработанный исходный сигнал [7]. Схема работы сайдчейн-технологии представлена на рис. 1.

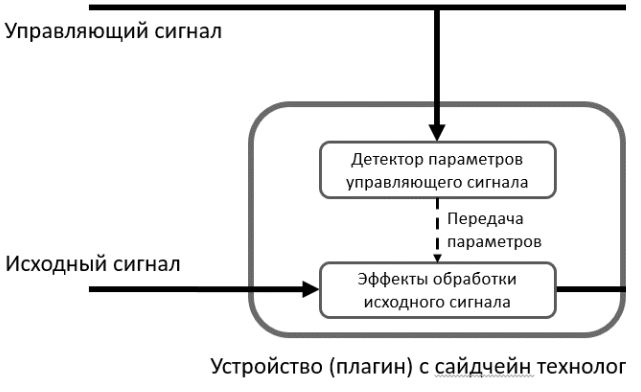

Рис.1. Схема работы сайдчейн-технологи

Стоит отметить, что исходный сигнал и управляющий сигнал изначально являются независимыми частями общей фонограммы. При этом на вход попадает копия управляющего сигнала для управления параметрами обработки исходного сигнала [8]. В итоге будет получено звучание исходного сигнала после обработки, а также оригинала управляющего сигнала, идущего независимо от исходного сигнала и его обработок. Оригинальный управляющий сигнал может как воспроизводится, так и быть отключен.

Используемые технологии. Для создания плагина использовалась кросплатформенная библиотека WDL-OL, позволяющая разрабатывать плагины форматов VST, AXX, AU. Также ее достоинства заключатся в разрешительной лицензии, позволяющей использовать разработанные плагины в коммерческих целях, а также возможность детально работать с графическим пользовательским интерфейсом в коде, прописывая в файле ресурсов png-изображения.

В качестве среды программирования использовалась Microsoft Visual Studio и язык программирования C++. Он используется практически во всех современных фреймворках, которые предназначены для решения подобных задач.

Плагин был разработан в формате VST (Visual Studio Technology), так как именно он является самым широко используемым форматом аудиоплагинов, работая при этом на всех операционных системах, в том числе Windows.

Разработанный плагин загружается в цифровую звуковую станцию Reaper, так как она является портативной и бесплатной.

Архитектура приложения. VST-плагины в операционной системе Windows являются динамически подключаемыми библиотеками типа .dll [9]. Библиотека WDL-OL подключается к проекту в Visual Studio через свойства проекта. Данную библиотеку можно скачать на официальном портале разработчика (например, на GitHub).

Разработка производится посредством объектно-ориентированного программирования, когда плагин является совокупностью классов, написанных на языке C++.

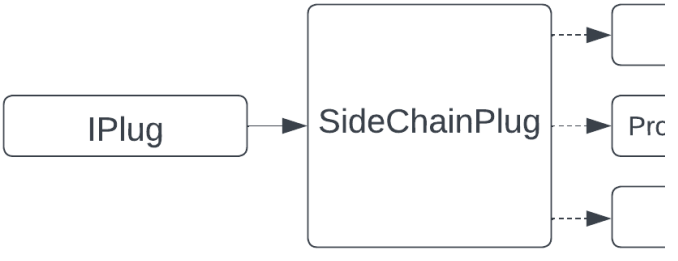

Рис.2. Диаграмма классов и методов

Диаграмма разработана на графическом языке UML. В данном случае сплошные стрелки обозначают наследования классов, а штриховые – методы класса. Основой плагина является класс SideChainPlug. Он является наследником класса IPlug библиотеки WDL-OL. Помимо конструктора и деструктора, класс SideChainPlug содержит в себе следующие методы:

– метод OnParamChange(), который вызывается при изменении параметров плагина (ручек, переключателей, фейдеров);

– метод ProcessDoubleReplacing(), в котором содержится алгоритм обработки входящего сигнала;

– метод LowPass(), который отвечает за фильтрацию входящего сигнала.

Особенности программной реализации. Суть работы плагина заключается в следующем – как только поступивший на дополнительный вход плагина управляющий сигнал превысил по громкости некоторый заданный порог, плагин начинает фильтровать исходный сигнал. В самом простом варианте блок-схема алгоритма выглядит следующим образом (рис. 3):

Рис. 3. Блок-схема алгоритма плагина

Подробное описание плагинов обработки и синтеза сигналов приведено автором в [10] и [11]. В данном случае потребуется реализовать получение плагином управляющего сигнала. Для этого необходимо прописать соответствующий параметр в заголовочном файле ресурсов плагина (рис. 4).

```
#define PLUG_CHANNEL_IO "2-2 4-
#define PLUG_SC_CHANS 2
```

Рис. 4. Константы для плагинов с сайдчейн-технологией

Здесь указывается количество входов и выходов плагина, а также количество каналов для сайдчейн-сигнала. В данном случае их два – правый и левый каналы, соответственно, в качестве управляющего сигнала может подаваться стереосигнал. Эти константы определяют то, что плагин, помимо исходного сигнала, может получать на вход звуковую информацию на дополнительный канал с помощью функций «посыл-возврат» цифровой рабочей станции [12]. Забегая вперед, можно продемонстрировать, как будет выглядеть маршрутизация сигнала в плагине с помощью матрицы роутинга цифровой звуковой рабочей станции (рис. 5).

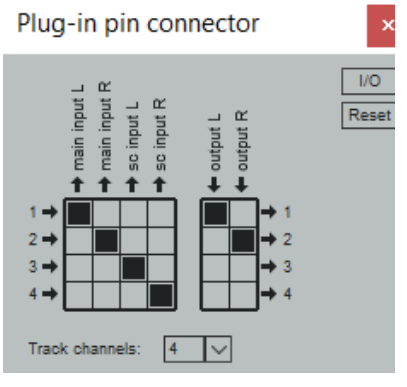

Рис. 5. Матрица маршрутизации сигналов в Reaper

Первая матрица обозначает входы плагина. Как видно, плагин имеет 4 входа, 1-й и 2-й имеют названия main input L и main input R соответственно. На эти каналы попадают левый и правый каналы исходного сигнала. Затем идет 3-й и 4-й каналы, на которые попадает левый и правый каналы управляющего сигнала. Рядом расположена матрица выходов, которых у плагина всего два – output L и output R – на эти выходы будет подаваться левый и правый канал обработанного исходного сигнала. Управляющий сигнал лишь попадает в плагин для участия в обработке, но не выходит из него [13].

Под параметрами плагина понимаются те переменные, которые выводятся в виде ручек, кнопок и переключателей на интерфейс плагина [14]. В качестве параметров плагина можно выделить следующие:

– Threshold – порог срабатывания эффекта (англ. Threshold – порог);

– Time – время, за которое эффект будет постепенно применяться и отключаться (англ. Time – время);

– Level – параметры уровня громкости для исходного и управляющего сигналов (англ. Level – уровень).

Каждому параметру будут соответствовать свои ручки управления. Как именно отобразить ручки на интерфейсе плагина, связать их с параметрами плагина и брать с них значение также описано в [10].

Чтобы управлять громкостью исходного и управляющего сигналов, необходимо создать две ручки громкости в интерфейсе плагина, а также две переменные, например, mGain и mGain2 соответственно. Для отображения уровней громкости сигналов и работы с ними в основном алгоритме плагина понадобятся четыре переменные: mVolumeL, mVolumeR, mVolumeLS, mVolumeRS, где первые две отвечают за правый и левый каналы исходного сигнала, а последние две – управляющего сигнала. Также необходимо создать переменную, отвечающую за порог срабатывания фильтрации – Thresh.

Как говорилось ранее, чтобы избежать щелчков и артефактов при применении эффекта фильтрации сигнала, необходимо плавно изменять частоту среза. Для этого нужно определить переменную типа FadeTime, отвечающую за то, как быстро будет происходить изменение частоты среза. Время в цифровом звуке выражается через количество семплов сигнала [15]. Максимальным временем изменения частоты была выбрана одна секунда. Одна секунда для стандартной частоты дискретизации равна 44100 семплам. Однако, частота дискретизации в проекте цифровой звуковой рабочей станции может меняться, поэтому необходимо также написать функцию определения частоты дискретизации. В итоге максимальное значение переменной FadeTime будет не больше, чем частота дискретизации проекта, потому что она отображает количество семплов в секунду [16].

В качестве эффекта, который будет применяться к исходному сигналу, был выбран фильтр нижних частот первого порядка с обратной связью, что подразумевает под собой использование выхода в качестве входа, то есть бесконечную импульсную характеристику. К фильтру будет применяться линейный оператор частоты среза. Математически данный фильтр записывается таким образом:

$$y(n) = y(n-1) + f * \left(x(n) - y(n-1)\right)$$
$$f \in [0;1],$$

где $x(n)$ – значение текущего входящего семпла, $y(n)$ – значение выходящего семпла, которое нужно рассчитать, $y(n-1)$ – значение предыдущего рассчитанного семпла, f – коэффициент, определяющий частоту среза. Для реализации фильтрации сигнала необходимо создать функцию, например, LowPass.

Основной алгоритм обработки исходного сигнала (рис. 6) находится в функции ProcessDoubleReplacing. Условный оператор проверяет превышение управляющим сигналом значения порога, обозначенного переменной Thresh. Если управляющий сигнал превышает порог, то указателям на элементы массива выходящего сигнала (*out1 и *out2 для левого и правого каналов соответственно) присваивается значение функции LowPass, отвечающей за фильтрацию сигнала. Данная функция также подробна была рассмотрена автором в [10] и [11].

```
191     if (mVolumeRS > Thresh || mVolumeLS > Thresh)
192     {
193         *out1 = LowPass(out1,LowPassFreq[iter]);
194         *out2 = LowPass(out2,LowPassFreq[iter]);
195         iter = iter + 1;
196         if (iter > FadeTime)
197         {
198             iter = FadeTime;
199         }
200      }
201     else
202     {
203         *out1 = LowPass(out1, LowPassFreq[iter]);
204         *out2 = LowPass(out2, LowPassFreq[iter]);
205         iter--;
206         if (iter < 0)
207         {
208             iter = 0;
209         }
210     }
```

Рис. 6. Код алгоритма обработки сигнала

Массив LowPassFerq содержит в себе частоты, которые будут меняться в зависимости от переменной iter, она будет накапливаться с каждым последующим проходом цикла. Если переменная iter становится равной по значению с переменной FadeTime, отвечающей за время изменения частоты фильтрации, выраженного в семплах, частота среза достигнет своего крайнего значения, определенного заранее в 2 кГц, и далее перестанет изменяться, оставаясь на этой частоте и срезая все частоты выше данной. Таким образом происходит постепенный переход от исходного сигнала к отфильтрованному.

Как только управляющий сигнал перестает превышать порог, алгоритм начинает идти по ветке else. Происходит обратная ситуация, частота среза начинает меняться с 2 кГц, повышаясь и постепенно открывая весь частотный спектр сигнала. Переменная iter при этом начинает уменьшаться, и, как только она дойдет до нулевого значения, ниже она уже не будет оставаться. Нулевое значение определяет то, что частотный спектр никак не изменяется.

Массив LowPassFerq, определяющий значения частот, высчитывается заранее относительно того, каким именно будет значение переменной FadeTime, отвечающей за время изменения частоты среза. Если время будет очень коротким, то частоты будут быстро меняться от максимального значения до указанного в алгоритме значения в 2 кГц, внося при этом некоторые артефакты в звук. Ручкой «Time» на плагине можно подобрать необходимо время, когда частоты начинают довольно быстро подавляться, но при этом никаких искажений в звуке не появляется.

Как может показаться, даже в том случае, когда фильтрация не должна производиться, значение выходящих семплов все равно проходит через функцию фильтрации сигнала. Однако в ней при нулевом значении частоты условный оператор сразу присваивает выходящему сигналу его же значение, оставляя его без изменения, не проходя цикл фильтрации и не затрачивая на это ресурсы процессора.

Обсуждение результатов

В результате разработки был получен плагин, реализующий фильтрацию исходного сигнала с помощью управляющего сигнала, используя сайдчейн-технологию. Интерфейс плагина представлен на рис. 7.

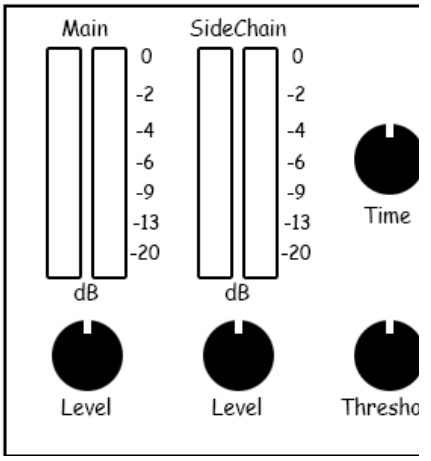

Рис. 7. Интерфейс плагина

Как только на вход плагина поступает исходный сигнал с той дорожки, на которой открыт данный плагин, определитель уровня исходного сигнала (индикатор Main) начинает отображать в реальном времени значение уровня громкости исходного сигнала. Уровень громкости этого сигнала можно изменить с помощью ручки громкости (ручка Level), находящейся под индикатором данного сигнала. Пока на вход плагина не начнет подаваться управляющий сигнал, изменение остальных ручек плагина не приведет ни к каким изменениям исходного сигнала.

Если на дополнительный вход с помощью функции «посыл-возврат» будет отправлен управляющий сигнал с другой дорожки, индикатор его громкости (индикатор SideChain) начнет в реальном времени отображать уровень громкости данного сигнала. Ручка Level, находящаяся под индикатором громкости управляющего сигнала, позволяет увеличить или понизить его громкость.

Ручка порога срабатывания эффекта (ручка Threshold) определяет порог громкости. Если громкость управляющего сигнала превысит данный порог, плагин начнет вносить изменения в исходный сигнал, фильтруя его, срезая его высокие частоты. Как только управляющий сигнал станет тише, чем значение ручки Threshold, эффект перестанет применяться, воспроизводя исходный сигнал без изменений.

Плавность применения эффекта определяется ручкой Time. Если значение ручки будет минимально, эффект будет применяться сразу же, как только управляющий сигнал превысит заданный порог. В таком случае в исходный сигнал будет фильтроваться мгновенно, от чего будут появляться искажения в исходном сигнале. При увеличении значения ручки Time, плавность применения эффекта будет изменяться. В данном случае будет происходить постепенное изменение частоты фильтрации. Чем больше значение ручки Time, тем дольше частота среза будет изменяться, доходя до своего основного значения в 2 кГц. Как только управляющий сигнал перестанет превышать порог, частота среза будет расти, постепенно

возвращая весь частотный спектр исходного сигнала. Максимальным временем, которое можно выставить ручкой Time и за которое частота может меняться, в данном случае взята одна секунда.

Для наглядной демонстрации плагина можно воспользоваться сторонним плагином-анализатором спектра сигнала. В цифровой звуковой рабочей станции необходимо создать две дорожки:

1) исходный сигнал (например, музыкальный подклад, который будет обрабатываться);

2) управляющий сигнал (например, голос диктора, во время звучания которого высокие частоты музыкального подклада будут подавляться).

Разработанный плагин открывается на дорожке с исходным сигналом, так как именно его плагин и будет обрабатывать. Сигнал дорожки с голосом диктора отправляется с помощью функции «посыл-возврат» на дополнительный канал с исходным сигналом. Плагин автоматически начинает распознавать управляющий сигнал, сверяя его громкость со значением порога, заданного ручкой Threshold, в случае превышения которого начинает происходить фильтрация частот. На рис. 8 слева представлен разработанный плагин, который отображает уровень исходного сигнала (Main), то есть музыкальный подклад уже воспроизводится, однако управляющий сигнал в виде голоса диктора еще не звучит, поэтому индикатор громкости канала SideChain ничего не отображает. Справа расположен частотный спектр музыкального подклада, который практически равномерно занимает весь выбранный частотный диапазон, отложенный по оси абсцисс, в данном случае от 600 Гц до 20 кГц, для большей наглядности, так как последующие изменения частот будут происходить в данном диапазоне. Ось ординат обозначает громкость в децибелах.

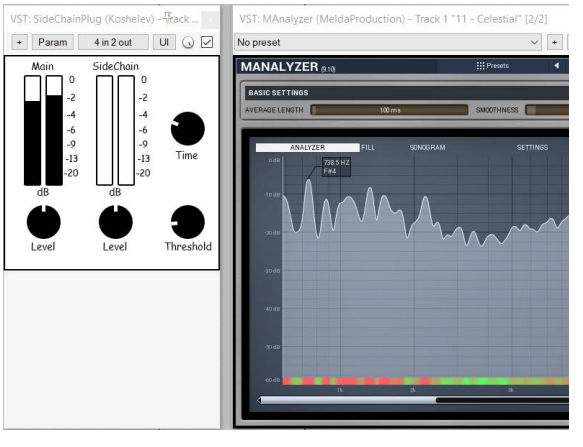

Рис. 8. Частотный спектр основного сигнала

Как только управляющий сигнал начинает звучать, индикатор громкости SideChain отображать ее уровень. Если уровень данного сигнала превышает порог, заданный ручкой Threshold, то анализатор спектра исходного сигнала будет отображать примерно следующее (рис. 9):

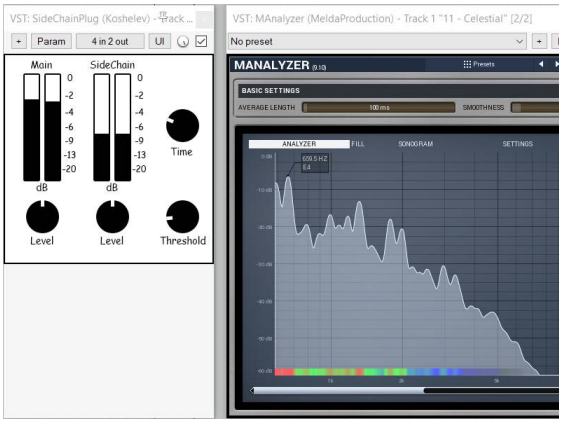

Рис. 9. Частотный спектр обработанного сигнала

На индикаторе громкости управляющего сигнала отображается его уровень, а на анализаторе спектра видно, как в исходном сигнале начали пропадать высокие частоты начиная примерно с 2 кГц. Как только управляющий сигнал станет тише заданного порога, частоты музыкального подклада начнут возвращаться до исходного состояния за время, заданного ручкой Time.

Заключение

С помощью языка программирования C++ и библиотеки WDL-OL разработан и реализован VST-плагин для обработки цифрового звукового сигнала с использованием сайдчейн-технологии. Плагин способен фильтровать исходный звуковой сигнал в зависимости от уровня громкости управляющего сигнала. Основными возможностями плагина являются индикация уровней исходного и управляющего сигналов, регулировка уровней исходного и управляющего сигналов, детектор уровня управляющего сигнала, фильтрация исходного сигнала в зависимости от уровня управляющего сигнала и плавность применения фильтрации. Также плагин имеет собственный графический интерфейс. Полный код плагина размещен на портале GitHub [17].

Литература

1. Sidechain Compression: Part 1 – Concepts and History URL: https://www.ableton.com/en/blog/sidechain-compression-part-1 (дата обращения 21.12.2022).

2. Sidechain Compression: Part 2 – Common and Uncommon Uses URL: https://www.ableton.com/en/blog/sidechain-compression-part-2-common-and-uncommon-uses/ (дата обращения 21.12.2022).

3. *Айфичер Э., Джервис Б.* Цифровая обработка сигналов: практический подход, 2-е издание. Пер. с англ. Москва: Вильямс, 2004. 992 с.

4. *Pirkle W.* Designing Audio Effect Plug-Ins in C++: With Digital Audio Signal Processing Theory. Oxford: Focal Press, 2012. 535 p.

5. *Gazi O.* Understanding Digital Signal Processing. Singapore: Springer, 2018. 303 p.

6. *Boulanger R., Lazzarini V.* The Audio Programming Book. Cambridge: The MIT Press, 2010. 900 p.

7. *Lazzarini V.* Computer Music Instruments: Foundations, Design and Development. Cham: Springer, 2017. 361 p.

8. Wise D. K. Concept, Design, and Implementation of a General Dynamic Parametric Equalizer // Journal of the Audio Engineering Society. 2009. Vol. 57. Pp. 16-28.

9. *Giannoulis et al.* D. A Tutorial on Digital Dynamic Range Compressor Design // Journal of the Audio Engineering Society. 2012. Vol. 60. Pp. 399-408.

10. *Кошелев Е. Е.* Разработка аудиоплагина для цифровой рабочей станции / Е. Е. Кошелев, С. В. Букунов // Вестник Российского нового университета серия «Сложные системы: модели. модели, анализ и управление» 2020. Т. Анализ. С. 85–97.

11. *Кошелев Е. Е.* Плагин-синтезатор для цифровой рабочей станции / Е. Е. Кошелев, С. В. Букунов // Известия СПбГТУ ЛЭТИ. 2021. № 4. С. 72–83.

12. *Loy G.* Musimathics, Volume 1: The Mathematical Foundations of Music. Cambridge: The MIT Press, 2011. 504 p.

13. *Reiss J. D.* Design of Audio Parametric Equalizer Filters Directly in the Digital Domain, IEEE Transactions on Audio // Speech and Language Processing. 2011. Vol. 19. Pp. 1843–1848.

14. *Lehmann E., Johansson A.* Prediction of Energy Decay in Room Impulse Responses Simulated with an Image-Source Model. Journal of the Acoustical Society of America. 2008. Vol. 124. Pp. 241-252.

15. *Kientzle T. A* Programmer's Guide to Sound. Boston: Addison Wesley, 1997. 464 p.

16. *Landford S.* Digital Audio Editing. New York: Routledge, 2013. 336 p.

17. GitHub. URL: https://github.com/egorrrkkkaaa/SideChainPlug (дата обращения 01.03.2022).

IMPLEMENTING SIDECHAIN TECHNOLOGY IN A VST PLUG-IN

EGOR E. KOSHELEV
Saint Petersburg State University of Architecture and Civil Engineering, St Petersburg, Russia

ABSTRACT

Introduction: A developed audio plug-in with its own graphical user interface is described. The plug-in implements a sidechain technology that allows you to filter the low frequencies of the main signal depending on the level of the side signal. An appropriate algorithm was developed to link the filtering level of the main signal with the volume level of the side signal. For the convenience of working with the plug-in, a graphical user interface was created in the form of detectors for the volume levels of the main and side signal. The interface contains knobs for changing the volume levels of these signals, knobs for the side signal level detector for filtering the main signal, and a knob for changing the smoothness of filtering. The developed plug-in is in VST format and can be used in almost any modern digital sound workstation. The plug-in is implemented in the C ++ language using the object-oriented programming approach. To create the plug-in, the Microsoft Visual Studio development environment and the WDL-OL library were used. The digital audio station Reaper was used as a digital sound workstation for testing of the developed plug-in.

Keywords: digital audio processing; VST plug-in; digital audio workstation; sidechain technology; object-oriented programming; graphic user interface.

REFERENCES

1. Sidechain Compression: Part 1 – Concepts and History URL: https://www.ableton.com/en/blog/sidechain-compression-part-1 (accessed 21.12.2022).
2. Sidechain Compression: Part 2 – Common and Uncommon Uses URL: https://www.ableton.com/en/blog/sidechain-compression-part-2-common-and-uncommon-uses/ (accessed 21.12.2022).
3. Ifeachor E. C., Jervis B. W. Tsifrovaya obrabotka signalov: prakticheskiy podkhod, 2-ye izdaniye. Per. s angl. [Digital Signal Processing: A Practical Approach 2nd Edition. Trans. From eng.] Moskva: Williams, 2004. 992 p. (In Russian)
4. Pirkle W. Designing Audio Effect Plug-Ins in C++: With Digital Audio Signal Processing Theory. Oxford: Focal Press, 2012. 535 p.
5. Gazi O. Understanding Digital Signal Processing. Singapore: Springer, 2018. 303 p.
6. Boulanger R., Lazzarini V. The Audio Programming Book. Cambridge: The MIT Press, 2010. 900 p.
7. Lazzarini V. Computer Music Instruments: Foundations, Design and Development. Cham: Springer, 2017. 361 p.
8. Wise D. K. Concept, Design, and Implementation of a General Dynamic Parametric Equalizer. Journal of the Audio Engineering Society. 2009. Vol. 57. Pp. 16-28.
9. Giannoulis et al. D. A Tutorial on Digital Dynamic Range Compressor Design. Journal of the Audio Engineering Society. 2012. Vol. 60. Pp. 399-408.
10. Koshelev E. E., Bukunov S.V. Development of an audio plug-in for a digital workstation // Vestnik Rossiyskogo novogo universiteta seriya «Slozhnyye sistemy: modeli. modeli, analiz i upravleniye» 2020. T. analiz. Pp. 85–97.
11. Koshelev E. E., Bukunov S.V. Plug-in synthesizer for a digital workstation // Izvestija SPbGETU «LETI». 2021. No 4. Pp. 72-83 (In Russ.)
12. Loy G. Musimathics, Volume 1: The Mathematical Foundations of Music. Cambridge: The MIT Press, 2011. 504 p.
13. Reiss J. D. Design of Audio Parametric Equalizer Filters Directly in the Digital Domain, IEEE Transactions on Audio. Speech and Language Processing. 2011. Vol. 19. Pp. 1843–1848.
14. Lehmann E., Johansson A. Prediction of Energy Decay in Room Impulse Responses Simulated with an Image-Source Model. Journal of the Acoustical Society of America. 2008. Vol. 124. Pp. 241-252.
15. Kientzle T. A Programmer's Guide to Sound. Boston: Addison Wesley, 1997. 464 p.
16. Landford S. Digital Audio Editing. New York: Routledge, 2013. 336 p.
17. GitHub. URL: https://github.com/egorrrkkkaaa/SideChainPlug (accessed 01.03.2022).