ИНФОРМАТИКА И ВЫЧИСЛИТЕЛЬНАЯ ТЕХНИКА

DOI: 10.25586/RNU.V9187.20.04.P.085 УДК 004.42

Е.Е. Кошелев, С.В. Букунов

РАЗРАБОТКА АУДИОПЛАГИНА ДЛЯ ЦИФРОВОЙ РАБОЧЕЙ СТАНЦИИ

Разработан аудиоплагин с пятью эффектами цифровой обработки звука. Все эффекты могут комбинироваться между собой в любой пропорции. Приведены программные коды для каждого алгоритма. Для удобства работы с плагином создан графический пользовательский интерфейс.
Ключевые слова: цифровая обработка звука, аудиоплагин, объектно ориентированное программирование.

E.E. Koshelev, S.V. Bukunov

DEVELOPMENT OF AN AUDIO PLUG-IN FOR A DIGITAL WORKSTATION

An audio plug-in with five digital audio processing effects has developed. All effects can be combined with each other in any proportion. Program codes for each algorithm are given. A graphical user interface has been created for the convenience of working with the plug-in.
Keywords: digital audio processing, audio plug-in, object-oriented programming.

Введение

В настоящее время рост вычислительной мощности современных компьютеров обусловил постепенную замену физических устройств, применяемых для обработки звука, на их программные аналоги [2, 3]. В современных студиях звукозаписи ключевым объектом является компьютер с цифровой звуковой рабочей станцией. Цифровая звуковая рабочая станция – это компьютерное программное обеспечение, предназначенное для записи, хранения, обработки и воспроизведения цифрового звука [11]. Одной из самых важных компонент цифровой рабочей станции является возможность подключения специальных модулей обработки звука – плагинов. Плагин представляет собой независимо компилируемый программный модуль, который подключается к основной программе для расширения ее возможностей [7]. Плагины, созданные для работы со звуком, называются аудиоплагинами.

Все аудиоплагины делятся на три основные группы: плагины-эффекты, плагины-анализаторы и плагины-инструменты [1; 4; 8; 10]. Каждая категория плагинов выполняет свои определенные функции.

Рынок плагинов постоянно обновляется, как за счет обновления существующих плагинов, так и в результате создания новых плагинов, реализующих новые алгоритмы обработки звука, поскольку любой аудиоплагин – это компьютерная программа. В отличие от физических устройств при разработке таких плагинов не существует каких-либо ограничений на реализуемые алгоритмы обработки звука. Именно поэтому многие музыканты, композиторы, звукорежиссеры используют в своей работе плагины собственной

разработки, реализующие нужные им эффекты, на основе какого-то нового уникального алгоритма либо реализующие в одном плагине эффект, получающийся путем последовательного соединения нескольких плагинов. Разработка собственного аудиоплагина становится весьма актуальной задачей еще и потому, что в литературе очень мало примеров программной реализации алгоритмов, лежащих в основе того или иного эффекта обработки звука.

Представленный в данной работе аудиоплагин представляет собой плагин-эффект, который реализует такие эффекты, как задержка входного сигнала, его прерывистое звучание, перегрузка (искажение) сигнала путем ограничения по амплитуде, фильтрация нижних частот, расширение стереообраза сигнала, регулировка громкости выходного сигнала.

Основные алгоритмы

Любой цифровой звуковой сигнал на компьютере представляется в виде последовательности семплов, полученных при преобразовании аналогового сигнала в цифровой (для плагинов-эффектов и плагинов-анализаторов) или при генерации цифрового сигнала (для плагинов-инструментов). Последовательность семплов – это последовательность значений уровня сигнала, записанных через равный временной интервал. Семпл сигнала по своей сути представляет собой вещественное число, значение которого обозначает уровень громкости сигнала в определенный момент времени. Аудиоплагин, в свою очередь, принимает на входе последовательно каждый семпл. В зависимости от алгоритма обработки плагин может обрабатывать как каждый семпл в отдельности, так и последовательность семплов.

В разработанном плагине реализовано несколько отличающихся по своей направленности алгоритмов обработки входящего сигнала, влияющих на различные его параметры: амплитуду, частоту и временное расположение.

Далее при рассмотрении каждого из этих эффектов использовались следующие обозначения:

$x(n)$ – значение текущего входящего семпла;

$y(n)$ – значение текущего выходящего семпла;

n – переменная, отвечающая за порядковый номер семпла.

Следует отметить, что при разработке алгоритмов аудиоплагинов помимо знаний в области физики и программирования используется достаточно сложный математический аппарат [5, 6, 10].

Эффект дисторшн. Дисторшн (англ. *distortion* – искажение) – звуковой эффект, достигаемый искажением сигнала путем его жесткого ограничения по амплитуде. Это значит, что когда сигнал превышает определенный заданный порог, все, что выше этого порога, обрезается и заменяется значением порога, при этом в сигнал вносятся искажения. Данный эффект произвел своего рода революцию гитарной музыки в свое время, а различные алгоритмы искажения сигнала по сей день используются в создании различных эффектов абсолютно любых направлений музыки.

При создании дисторшн-эффекта следует иметь в виду, что амплитуда цифрового звукового сигнала изменяется от –1 до 1. Поэтому амплитуды, превышающие определенный

порог, следует понимать как превышающие положительный порог или опускающиеся ниже отрицательного порога.

Математический алгоритм эффекта дисторшн может быть записан следующим образом:

$$\begin{cases} y(n) = \min\left(x(n); d\right), \; x(n) \geq 0, \\ y(n) = \max\left(x(n); -d\right), \; x(n) < 0; \end{cases}$$

$$0 \leq d \leq 1, \;\; d \in R;$$

$$n \geq 0, \;\; n \in Z.$$

В данном случае d – это значение порога дисторшн-эффекта. Если значение входящего семпла $x(n)$ больше нуля, то выбирается минимум между значением семпла и заданным значением порога d. Соответственно, если значение семпла выше указанного порога, то значение выходящего семпла $y(n)$ в данный момент будет равно значению порога d. Аналогично в случае, когда значение входящего семпла меньше нуля и опускается ниже отрицательного значения порога, значение выходящего семпла равно отрицательному значению порога – d.

Фильтр нижних частот. Фильтр нижних частот (англ. *low-pass filter*) – эффект, изменяющий амплитудно-частотную характеристику сигнала, пропуская частотный спектр сигнала ниже частоты среза и подавляя частоты сигнала выше этой частоты. Разработка алгоритмов фильтрации представляет собой одну из самых сложных сфер обработки цифровых звуковых сигналов, основанную на сложном математическом аппарате обработки дискретных сигналов [1, 10].

В данной работе реализован линейный фильтр нижних частот первого порядка с бесконечной импульсной характеристикой. Это значит, что данный фильтр будет использовать один свой выход в качестве входа, то есть образовывать обратную связь. Линейным он является потому, что ко входящему сигналу будет применен линейный оператор (частота 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 $y(n)$ – значение текущего выходящего семпла, которое нужно рассчитать; $y(n-1)$ – значение предыдущего рассчитанного семпла; f – коэффициент, определяющий частоту среза.

Эффект гейтера. Гейтер (от англ. *gate* – ворота) – это эффект попеременного включения и выключения звукового сигнала на определенный временной интервал. Результатом работы данного эффекта будет прерывистое звучание исходного сигнала. Временной интервал, в который сигнал то звучит, то не звучит, называется временем работы гейтера.

Как правило, в плагинах, работающих с временными характеристиками, время работы гейтера задается в миллисекундах, однако сам звуковой сигнал представлен в виде последовательности семплов. Количество семплов во временном интервале определено частотой дискретизации сигнала. Соответственно, для работы с временным расположением последовательности семплов необходимо время перевести в количество семплов. Частота дискретизации говорит о том, какое количество семплов сигнала находится в одной

секунде сигнала. Исходя из этого, можно получить формулу для определения количества семплов в произвольном интервале времени:

$$s = \frac{t}{1000} r, \tag{1}$$

где t – временной интервал, млс; r – текущая частота дискретизации, Гц; s – количество семплов в данном временном интервале t. К примеру, самой распространенной частотой дискретизации является 44 100 Гц. В этом случае временной интервал в 200 млс будет равен 8820 семплам.

Результатом работы данного эффекта является выходящий сигнал, определенное количество семплов которого попеременно либо равны входящему сигналу, либо равны нулю. Допустим, первые 5000 семплов мы слышим звук, а следующие 5000 семплов – нет, и так попеременно.

Математическая запись алгоритма работы данного эффекта выглядит следующим образом:

$$\begin{cases} y(n) = x(n),\ n \in [0; sk), \\ y(n) = 0;\ n \in [sk; sk + s), \end{cases}$$
$$k = 1,\ 3,\ 5,\ ...,$$

где s – количество семплов в заданном временном интервале; k – коэффициент, принимающий нечетные значения, определяющий разбиение на интервалы, которые имеют длительность s. В зависимости от того, в каком интервале находится семпл n, выходящий сигнал $y(n)$ будет либо равным входящему сигналу $x(n)$, либо равным нулю.

Эффект дилэй. Эффект дилэй (англ. *delay* – задержка) в чистом виде – это эффект задержки сигнала на определенный временной интервал. На практике под данным эффектом, как правило, подразумевается повторение сигнала через заданные интервалы времени. Время обычно задается в миллисекундах. Зачастую дилэй применяется в качестве художественного приема для создания определенных эффектов с повторением сигнала.

Повторение сигнала означает то, что результирующий сигнал определяется в виде суммы входящего сигнала в настоящий момент времени и входящего сигнала в момент времени, отстающий на определенное значение.

Математически алгоритм вычисления семплов дилэй-эффекта можно записать как

$$y(n) = x(n) + x(n - T),$$

где T – задержка, заданная в семплах.

Как и практически в любых других эффектах, связанных с временно́й обработкой, время задержки сигнала дилэй-эффекта обычно задается в млс и впоследствии переводится в количество семплов с помощью формулы (1).

Эффект Хааса. Данный эффект относится к психоакустическим, потому что связан с особенностями восприятия звука, а именно его пространственной локализацией. Суть его заключается в том, что если пустить из двух равноудаленных источников один и тот же сигнал, восприниматься он будет по центру. Однако если из одного источника сигнал задержать на определенный временной интервал (до 40 млс), то звук будет восприниматься с широким стереообразом. Это и называется эффектом Хааса. Побочным эффектом

здесь является то, что сигнал, который пришел первым, воспринимается громче, чем тот, который пришел вторым, хотя сигналы имеют одинаковую громкость. Именно поэтому данный эффект также именуется эффектом приоритета. Эффект Хааса позволяет преобразовать монофонический сигнал в стереофонический.

Реализация данного эффекта подобна реализации дилэй-эффекта, но обработка происходит только на одном канале, и обработанный сигнал не суммируется с исходным, а воспроизводится с некоторой задержкой. Другой канал остается нетронутым. Если же сделать задержку больше 40 млс, то эффект перестанет давать нужный результат, то есть будет слышен только исходный сигнал и его повторение, поскольку при задержках до 40 млс два сигнала воспринимаются как один широкий сигнал.

Используемые технологии

Для разработки плагина была выбрана библиотека WDL-OL. Данная библиотека позволяет создавать кроссплатформенные плагины – то есть плагины различных форматов (VST, AU, AXX). Библиотека содержит большое количество классов и функций, необходимых для создания аудиоплагинов. Также к ее достоинствам можно отнести наличие разрешительной лицензии для свободного использования данной библиотеки при разработке плагинов даже в коммерческих целях. Кроме того, библиотека имеет средства для создания собственного пользовательского интерфейса. Следует отметить, что графический интерфейс пользователя в библиотеке WDL-OL собирается в коде, то есть все ресурсы необходимо самостоятельно прописывать в файле ресурсов (rc-файле). С одной стороны, это можно отнести к ее недостаткам, с другой – это позволяет разработчику создавать свои собственные графические изображения элементов пользовательского интерфейса (ручки регулировки, переключатели) в виде bmp-файлов.

Для создания плагина использовалась среда разработки Microsoft Visual Studio.

В качестве языка программирования был выбран язык C++, поскольку именно этот язык используется во всех основных фреймворках, предназначенных для решения такого рода задач [7; 8].

В качестве формата плагина был выбран формат VST (Visual Studio Technology), который является самым популярным и широкоиспользуемым форматом аудиоплагинов. Кроме того, данный формат работает во всех основных операционных системах.

В качестве цифровой звуковой рабочей станции, в которую будет загружаться готовый плагин, использовалась станция Reaper, поскольку она является бесплатной, имеет портативную версию и используется многими пользователями для работы с аудиоматериалом. Следует отметить, что плагин был протестирован и на других популярных цифровых звуковых рабочих станциях.

Программная реализация

В операционной системе Windows аудиоплагины в формате VST представлены в виде динамически подключаемых библиотек и имеют расширение .dll. Поэтому для разработки плагина в среде Microsoft Visual Studio необходимо создать соответствующий проект и дать ему какое-то имя (например, vkrPlug). В свойствах проекта следует указать путь к файлам библиотеки WDL-OL, которую можно скачать на официальной странице разработчика (например, на портале GitHub).

Программная реализация плагина представляет собой класс (в данном случае классу было дано имя vkrPlug), который является наследником класса IPlug библиотеки WDL-OL. В состав методов класса входят следующие методы:

- конструктор класса;
- метод, вызываемый при изменении параметров плагина (ручек, фейдеров, переключателей);
- метод, который будет осуществлять обработку входящего сигнала;
- методы, реализующие алгоритмы различных эффектов.

Реализация дисторшн-эффекта. Для программной реализации данного алгоритма в классе плагина был создан метод с именем Distortion(), который принимает семпл определенного канала и возвращает его. Поскольку значения семплов имеют тип double, то и метод, который возвращает обработанный семпл, должен иметь тип возвращаемого значения double. Также в private-секции класса плагина была создана переменная типа double с именем DistThreshold, хранящая значение порога.

Программный код метода, реализующего алгоритм дисторшн, представлен на рисунке 1.

```
double vkrPlug::Distortion(double* in)
{
    if (*in >= 0)
    {
        *in = fmin(*in, DistThreshold);
    }
    else
    {
        *in = fmax(*in, -DistThreshold);
    }
    *in = *in / DistThreshold;
    return *in;
}
```

Рис. 1. Программный код алгоритма дисторшн-эффекта

Как видно из представленного кода, перед возвращением из метода полученное значение делится на порог. Это сделано для того, чтобы компенсировать уменьшение громкости сигнала, так как чем сильнее происходит ограничение сигнала, тем слабее он становится по громкости. Также в данном случае не происходит присвоения выходящим семплам значений входящих, а происходит обработка самих входящих семплов, так как данная функция является частью программы.

Реализация фильтра нижних частот. Для программной реализации данного алгоритма в классе плагина была создана функция с именем LowPass(), которая принимает в качестве аргументов семпл определенного канала и номер канала. Так как значения семплов имеют тип double, то и функция, которая должна возвращать обработанный семпл, должна иметь тип возвращаемого значения double. Фильтр с бесконечной импульсной характеристикой подразумевает использование выхода в качестве входа, то есть в данном случае выходящий семпл $y(n)$ зависит от предыдущего значения выходящего семпла $y(n-1)$. Для этого необходимо создать буфер, хранящий предыдущее значение выходя-

щего семпла. Поскольку плагин работает с двумя каналами (левый и правый), то для каждого канала буфер должен быть свой. Поэтому в private-секции класса плагина был создан массив типа double размерностью в два элемента. Данный массив необходимо проинициализировать нулевыми значениями в конструкторе. Функция получает в качестве аргумента номер канала и использует соответствующий этому номеру элемент массива. Для хранения частоты среза в private-секции класса плагина была создана соответствующая переменная типа double, значение которой изменяется от 0 до 1.

Данный фильтр является фильтром первого порядка. Это означает, что каждую октаву амплитуда составляющих сигнала над частотой среза становится в два раза ниже, то есть громкость сигнала падает на 6 дБ. Эту характеристику фильтра можно изменить, сделав простое последовательное соединение фильтров. В данном плагине реализовано последовательное подключение четырех фильтров, работающих по одному алгоритму, что позволяет реализовать падение громкости на 24 дБ/окт (в отфильтрованном сигнале будет меньше высоких частот).

Усовершенствованный код метода, реализующего фильтр нижних частот, представлен на рисунке 2.

```
double vkrPlug::LowPass(double* in, int i)
{
    bufferLP1[i] += LowPassFreq * (*in - bufferLP1[i]);
    bufferLP2[i] += LowPassFreq * (bufferLP1[i] - bufferLP2[i]);
    bufferLP3[i] += LowPassFreq * (bufferLP2[i] - bufferLP3[i]);
    bufferLP4[i] += LowPassFreq * (bufferLP3[i] - bufferLP4[i]);
    return *in = bufferLP4[i];
}
```

Рис. 2. Программный код алгоритма фильтрации

Реализация алгоритма гейтера. При реализации данного алгоритма для задания времени в миллисекундах можно было бы создать переменную, которая бы принимала значения, например, от 0 до 1000. Однако в данном плагине был создан и реализован алгоритм, в котором время работы гейтера зависит от выбранной доли музыкального такта и темпа проекта в цифровой рабочей станции. В этом случае формула (1) для расчета количества семплов во временном интервале преобразуется к следующему виду:

$$s = \frac{240}{Tempo} pr,\ p = \frac{1}{2n},\ n = 1,\ 2,\ 3,\ ..., \qquad (2)$$

где p – определенная доля музыкального такта (может принимать значения 1, 1/2, 1/4 и т.д.); r – текущая частота дискретизации, Гц; s – количество семплов в данной доле такта p.

Для реализации алгоритма в private-секции класса плагина были созданы: переменная типа bool с именем Gater Switch и начальным значением false и две переменные типа double с именами Gater Time и Gater SampleCount, отвечающие за временной интервал в семплах и номер обрабатываемого семпла соответственно. При итерировании переменная Gater Sample Count будет наращиваться, пока не достигнет значения, равного значению переменной Gater Time. Как только это произойдет, значение переменной Gater Switch изменится на противоположное. Для реализации данного алгоритма была создана

соответствующая функция с именем Gater(), которая принимает в качестве аргументов семплы левого и правого каналов.

Для определения времени работы гейтера необходимо знать текущий темп в проекте цифровой рабочей станции и текущую частоту дискретизации. Темп можно определить с помощью библиотечной структуры ITime Info, предварительно создав соответствующую структурную переменную, например, Project Time Info. Тогда текущий темп будет храниться в поле mTempo переменной Project Time Info (Project Time Info.mTempo). Для определения частоты дискретизации можно воспользоваться библиотечной функцией Get Sample Rate(), присвоив ее значение новой переменной типа double, например, Current Sample Rate. Также необходимо создать переменную типа int, например, Gater Time Choice, которая будет принимать значения 2, 3, 4 или 5, для определения доли такта исходя из формулы (2). Переменная Gater Time Choice выступает в качестве показателя степени формулы (2). Свое значение она получает в зависимости от выбора переключателя в графическом интерфейсе.

Код алгоритма эффекта гейтера представлен на рисунке 3.

```
void vkrPlug::Gater(double* in1, double* in2)
{
    GaterTime = 240 / ProjectTimeInfo.mTempo / pow(2,GaterTimeChoice) * CurrentSampleRate;

    if (GaterSampleCount > GaterTime)
    {
        GaterSampleCount = 0;
        GaterSwitch = !GaterSwitch;
    }

    if (GaterSwitch == false) //звучание
    {
        *in1 = *in1;
        *in2 = *in2;
    }
    else //тишина
    {
        *in1 = 0;
        *in2 = 0;
    }
    GaterSampleCount++;
}
```

Рис. 3. Программный код алгоритма эффекта гейтера

Представленный код работает следующим образом. Сначала по формуле (2) определяется время работы гейтера Gater Time в семплах. Значение переменной Gater Sample Count изначально равно нулю, поэтому первый условный оператор не выполняется. Значение переменной Gater Switch изначально равно false, поэтому входящий сигнал равен самому себе и никак не меняется. Далее значение переменной Gater Sample Count инкрементируется. Когда значение переменной Gater Sample Count станет больше значения переменной Gater Time, значение переменной Gater Sample Count снова станет равным нулю, значение переключателя Gater Switch станет равным true, и входящий сигнал будет равным нулю, пока значение Gater Sample Count снова не станет больше времени работы гейтера, что повлечет за собой переключение Gater Switch. Данный алгоритм работает

с изменением входящего сигнала. Впоследствии он присваивается выходящему сигналу в главной функции обработки Process Double Replacing().

Недостаток данного алгоритма заключается в том, что громкость сигнала меняется достаточно резко, от чего появляются так называемые шумы дробления. Кроме того, значение переменной Gater Sample Count не обнуляется после остановки воспроизведения, что мешает корректной работе алгоритма при новом воспроизведении, потому что значение переменной Gater Sample Count начинается не с нуля и, соответственно, первый интервал всегда будет короче всех последующих.

Для решения первой проблемы необходимо было выбрать длину затухания и появления звука. В процессе тестирования алгоритма было выяснено, что оптимальным является значение в 200 семплов. Далее был создан массив из 201 элемента (чтобы исключить выход за границы массива) с проинициализированными элементами, значения которых линейно возрастали, начиная с единицы. В той части алгоритма, где происходило звучание сигнала, начальные и конечные 200 семплов сигнала определенным образом делились на элементы из данного массива для постепенного уменьшения или увеличения амплитуды, что позволяло избежать шумов дробления.

Для реализации сброса состояния счетчика семплов (перевода переключателя Gater Switch в положение false) можно снова воспользоваться библиотечной структурой ITimeInfo. Для этого в данной структуре существует поле mTransport Is Running, принимающее значения true и false, при этом значение false соответствует остановке режима воспроизведения. Таким образом, данная проблема успешно была решена использованием простого условного оператора.

Реализация алгоритма дилэй-эффекта. Для программной реализации данного алгоритма требуется буфер для хранения предыдущих значений входящего сигнала. Он был реализован в виде массивов типа double, например buffer L и buffer R. Поскольку плагин работает с двумя каналами (левый и правый), то были созданы два буфера. Время задержки обычно задается от 0 до 1000 млс (1 с). Количество семплов в одной секунде звука равно текущей частоте дискретизации. Это необходимо знать, чтобы проинициализировать данные массивы нулевыми значениями в конструкторе плагина.

Далее нужно создать переменную типа double, например Delay Time, которая будет хранить значение задержки сигнала в семплах. Также нужно создать переменную типа int, например Delay Sample Count, которая будет выполнять роль счетчика семплов. Для реализации алгоритма была создана функция Delay() с возвращаемым типом void, принимающая в качестве аргументов входящие и выходящие семплы обоих каналов.

Программный код алгоритма дилэй-эффекта представлен на рисунке 4.

```
void vkrPlug::Delay(double* in1, double* in2, double* out1, double* out2)
{
    *out1 = *in1 + bufferL[DelaySampleCount];
    *out2 = *in2 + bufferR[DelaySampleCount];

    bufferL[DelaySampleCount] = *in1;
    bufferR[DelaySampleCount] = *in2;
    DelaySampleCount++;

    if (DelaySampleCount > DelayTime)
        DelaySampleCount = 0;
}
```

Рис. 4. Программный код алгоритма дилэй-эффекта

Как только на вход пришел первый семпл, на выходе будет получена сумма этого семпла и первого элемента буфера, который изначально равен нулю. Затем в буфер будет записываться значение входящего семпла, а переменная Delay Sample Count для счета количества семплов будет увеличиваться на единицу, что будет говорить о переходе к следующему семплу и позволит записать новый входящий сигнал в следующий элемент массива буфера. Так будет происходить до тех пор, пока количество семплов Delay Sample Count не превысит время задержки в семплах Delay Time. Как только это произойдет, счетчик обнулится, и при входе следующего семпла на выход пойдет этот текущий входящий семпл и значение элемента буфера, записанное в самом начале, то есть значение самого первого входящего семпла. Затем, так как это элемент массива уже использовался, он перезаписывается новым текущим значением семпла. При следующем входящем семпле выходящий сигнал уже будет суммироваться как данный входящий семпл и второй элемент буфера, в котором хранится значение второго входящего семпла, записанного ранее. Затем этот элемент перезаписывается. Именно таким образом реализуется эффект повторения сигнала.

Реализация алгоритма эффекта Хааса. Для реализации данного эффекта необходимо создать в классе плагина буфер в виде массива типа double, например buffer H. Далее нужно создать функцию, например Haas(), которая будет принимать входящий сигнал одного канала и задерживать его. Алгоритм идентичен алгоритму, реализующему эффект дилэя, но в данном случае функция принимает только один входящий сигнал, и для его фиксирования в буфер необходимо создать отдельную переменную, например temp H. Данная переменная будет записывать значение входящего сигнала, а потом присваиваться элементу буфера, так как сам входящий сигнал будет изменяться и не может быть присвоен буферу, иначе работа станет некорректной. Поскольку время задержки постоянное и равно 30 млс, его перевод в количество семплов можно прописать в конструкторе. За роль счетчика семплов будет отвечать переменная типа double, например, Haas Sample Count.

Программный код алгоритма эффекта Хааса представлен на рисунке 5.

```
void vkrPlug::Haas(double* in)
{
    double tempH = *in;
    *in = bufferH[HaasSampleCount];

    bufferH[HaasSampleCount] = tempH;
    HaasSampleCount++;

    if (HaasSampleCount >= HaasTime)
        HaasSampleCount = 0;
}
```

Рис. 5. Программный код алгоритма эффекта Хааса

Как можно заметить, алгоритм данного эффекта аналогичен эффекту дилэя, только изменения происходят с самим входящим сигналом и лишь на одном канале. За включение и выключение эффекта будет отвечать специально созданный переключатель в графическом интерфейсе плагина.

Недостатком данного эффекта, как было отмечено выше, является то, что громкость сигнала, пришедшего первым, кажется выше. Поэтому такой звук сложнее обрабатывать в дальнейшем, так как физически уровень сигналов обоих каналов одинаковый, а на слух – разный. Это можно исправить с помощью других обработок или пренебречь этим.

Реализация пресетов. Любой плагин-эффект – это набор определенных параметров, которые настраиваются пользователем в зависимости от целей. Сохраненный набор предустановленных значений параметров плагина называется пресетом. Зачастую плагины имеют некоторое количество пресетов для упрощения пользования плагином. В данной работе было реализовано несколько пресетов, каждый из которых включает определенный эффект, а степень работы остальных равна нулю, то есть они не вносят никаких изменений в сигнал.

Для создания пресетов в библиотеке WDL-OL существует библиотечная функция Make Preset(), которая в качестве аргументов принимает название пресета и значения каждого параметра. Для удобства работы с несколькими пресетами была создана функция с именем Create Presets(), в которой были прописаны все пресеты. Вызов этой функции происходит в конструкторе плагина.

В данном плагине было создано несколько пресетов, каждый из которых включает определенный эффект, а степень работы остальных равна нулю, то есть они не вносят никаких изменений в сигнал.

Графический интерфейс пользователя

Любой современный аудиоплагин имеет свой графический интерфейс, включающий в себя различные регуляторы, фейдеры, переключатели и др. Для создания пользовательского интерфейса использовались возможности библиотеки WDL-OL. Был разработан дизайн графического интерфейса, выбраны графические изображения элементов управления и написаны функции, осуществляющие связь элементов управления с алгоритмами работы плагина.

Внешний вид графического пользовательского интерфейса плагина представлен на рисунке 6.

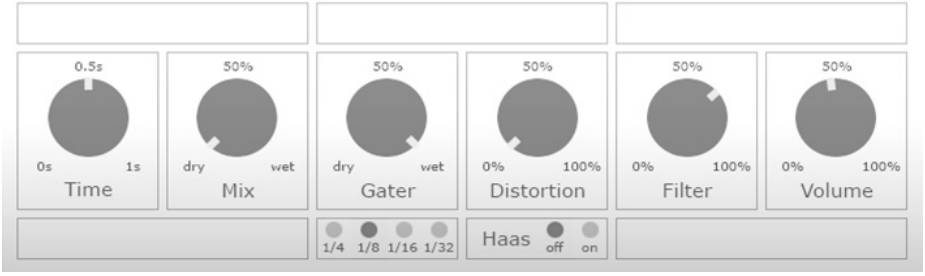

Рис. 6. Графический интерфейс пользователя плагина

Интерфейс имеет ручки регулировки степени обработки для каждого эффекта и несколько переключателей для изменения определенных параметров обработки сигнала. Все эффекты могут комбинироваться между собой в любой пропорции. Также есть ручка регулировки громкости выходного сигнала.

Заключение

В данной работе с помощью библиотеки WDL-OL и языка программирования C++ разработан и реализован аудиоплагин, позволяющий производить цифровую обработку звука с помощью различных эффектов. Плагин позволяет создавать прерывистое звучание исходного сигнала в зависимости от темпа и выбранной доли музыкального такта, осуществлять перегрузку сигнала, искажая сигнал путем ограничения по амплитуде, а также осуществлять фильтрацию низких частот в зависимости от выбранной частоты, расширять стереообраз сигнала путем задержки одного канала относительно другого и регулировать выходную громкость.

Для упрощения пользования плагином с помощью наборов предустановленных значений параметров плагина были реализованы несколько пресетов.

Для удобства работы с плагином был спроектирован и реализован графический интерфейс пользователя.

Разработанный аудиоплагин имеет формат VST, что позволяет подключать его практически ко всем цифровым звуковым рабочим станциям. Кроме того, в зависимости от свойств компиляции может быть создана как 64-битная версия плагина, так и 32-битная для работы в 32-битных операционных системах.

Литература

1. *Айфичер Э., Джервис Б.* Цифровая обработка сигналов: практический подход. 2-е изд. М.: Вильямс, 2004. 992 с.
2. *Загуменнов А.П.* Компьютерная обработка звука. М.: Пресс, 2017. 384 с.
3. *Gazi O.* Understanding Digital Signal. Singapore: Springer, 2018. 303 p.
4. *Lazzarini V.* Computer Music Instruments: Foundations, Design and Development. Cham: Springer, 2017. 361 p.
5. *Loy G.* Musimathics. The Mathematical Foundations of Music. Cambridge: The MIT Press, 2011. Vol. 1. 504 p.
6. *Loy G.* Musimathics. The Mathematical Foundations of Music. Cambridge: The MIT Press, 2011. Vol. 2. 584 p.
7. *Pirkle W.* Designing Audio Effect Plug-Ins in C++: With Digital Audio Signal Processing Theory. Oxford.: Focal Press, 2012. 525 p.
8. *Pirkle W.* Designing Software Synthesizer Plug-Ins in C++: For RackAFX, VST3, and Audio Units. London: Taylor & Francis Ltd, 2015. 760 p.
9. *Reiss J., McPherson A.* Audio Effects: Theory, Implementation and Application. Bosa Roca.: Taylor & Francis Inc, 2014. 367 p.
10. *Zavalishin V.* The Art of VA Filter Design. URL: https://www.native-instruments.com/fileadmin/ni_media/downloads/pdf/VAFilterDesign_1.1.1.pdf (date of the application: 19.07.2020).
11. *Zolzer U.* Digital Audio Signal Processing. Hoboken: Wiley-Blackwell, 2008. 340 p.

Literatura

1. *Ajficher E., Dzhervis B.* Cifrovaya obrabotka signalov: prakticheskij podhod. 2-e izd. M.: Vil'yams, 2004. 992 s.
2. *Zagumennov A.P.* Komp'yuternaya obrabotka zvuka. M.: Press, 2017. 384 s.
3. *Gazi O.* Understanding Digital Signal. Singapore: Springer, 2018. 303 p.
4. *Lazzarini V.* Computer Music Instruments: Foundations, Design and Development. Cham: Springer, 2017. 361 p.

5. *Loy G.* Musimathics. The Mathematical Foundations of Music. Cambridge: The MIT Press, 2011. Vol. 1. 504 p.
6. *Loy G.* Musimathics. The Mathematical Foundations of Music. Cambridge: The MIT Press, 2011. Vol. 2. 584 p.
7. *Pirkle W.* Designing Audio Effect Plug-Ins in C++: With Digital Audio Signal Processing Theory. Oxford.: Focal Press, 2012. 525 p.
8. *Pirkle W.* Designing Software Synthesizer Plug-Ins in C++: For RackAFX, VST3, and Audio Units. London: Taylor & Francis Ltd, 2015. 760 p.
9. *Reiss J., McPherson A.* Audio Effects: Theory, Implementation and Application. Bosa Roca.: Taylor & Francis Inc, 2014. 367 p.
10. *Zavalishin V.* The Art of VA Filter Design. URL: https://www.native-instruments.com/fileadmin/ni_media/downloads/pdf/VAFilterDesign_1.1.1.pdf (date of the application: 19.07.2020).
11. *Zolzer U.* Digital Audio Signal Processing. Hoboken: Wiley-Blackwell, 2008. 340 p.

DOI: 10.25586/RNU.V9187.20.04.P.097 УДК 681.31

А.Г. Басыров, В.В. Кузнецов, В.Г. Терехов

ПРОГНОЗИРОВАНИЕ ПОЛНОГО ОТКАЗА СПЕЦИАЛИЗИРОВАННОЙ ВЫЧИСЛИТЕЛЬНОЙ СИСТЕМЫ

Описан метод прогнозирования полного отказа специализированной вычислительной системы, базирующийся на зависимости вероятности отказа устройства от интенсивности его сбоев. Раскрыты способы контроля состояния и функционирования вычислительной системы. Приведен пример расчета прогнозируемой вероятности отказа вычислительной системы.
Ключевые слова: вычислительная система, отказ, сбой, контроль.

A.G. Basyrov, V.V. Kuznetsov, V.G. Terekhov

PREDICTING COMPLETE FAILURE OF A SPECIALIZED COMPUTER SYSTEM

A method for predicting the complete failure of a specialized computer system based on the dependence of the probability of device failure on the intensity of its failures is described. Methods for monitoring the state and functioning of the computer system are disclosed. An example of calculating the predicted probability of failure of a computer system is given.
Keywords: computer system, failure, interruption, control.

Введение

Срок активного функционирования любой современной сложной системы зависит от долговечности функционирования ее подсистем. Одной из важнейших подсистем, определяющих возможности применения системы по назначению, является специализированная вычислительная система (СВС). Для определения корректности функцио-