Санкт-Петербургский государственный университет

НАИМОВ Арсен

Выпускная квалификационная работа

«Разработка интерактивной инсталляции с использованием генеративной анимации на основе системы частиц»

Уровень образования: бакалавриат
Направление *09.03.03 «Прикладная информатика»*
Основная образовательная программа СВ.5078
Профиль *«Прикладная информатика в области искусств и гуманитарных наук»*

Научный руководитель:
Канд. физ.-мат. наук, доцент, кафедра информационных систем в искусстве и гуманитарных науках, СПбГУ
Щербаков Павел Петрович

Рецензент:
Общество с ограниченной Ответственностью «Резолла»
Гордеев Александр Владимирович, профессор, Кафедра вычислительных систем и сетей, «Санкт-Петербургский государственный университет аэрокосмического приборостроения»

Санкт-Петербург
2023

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ФАКУЛЬТЕТ ИСКУССТВ
Кафедра информационных систем в искусстве и гуманитарных науках
Основная образовательная программа
«Прикладная информатика в области искусств и гуманитарных наук»

УТВЕРЖДАЮ
Заведующий кафедрой

_____

ЗАДАНИЕ

по подготовке выпускной квалификационной работы студентки Наимова Арсена

1. Тема работы: Разработка интерактивной инсталляции с использованием генеративной анимации на основе системы частиц
2. Срок сдачи студентом законченной работы: июнь 2023
3. Срок сдачи текста выпускной квалификационной работы для выкладывания в Blackboard: 15 мая 2023 г.
4. Исходные данные к работе: Оригинальная идея
5. План-график выполнения квалификационной работы:

Номера и содержание этапов работы	**Плановая дата сдачи**
1.Планирование архитектуры приложения, выбор технологий для реализации	октябрь
2. Изучение технологий для реализации	декабрь
3. Написание приложения	март
4. Отладка и тестирование приложения	апрель
5. Написание теоретической части диплома	апрель-май
6. Защита теоретической и практической частей диплома	июнь

Руководитель от кафедры: Щербаков Павел Петрович

_____

(должность, Фамилия Имя Отчество, подпись)

Задание принял к исполнению_____ Подпись студента_____

_____ (дата)

АННОТАЦИЯ

выпускной квалификационной работы
Наимова Арсена

«Разработка интерактивной инсталляции с использованием генеративной анимации на основе системы частиц»

Целью работы является разработка аудиовизуализатора. Данная выпускная квалификационная работа состоит из 6 частей: анализ предметной области генеративного творчества, анализ возможных методов разработки интерактивных видеоинсталляций, проектирование архитектуры, разработка модулей потовой обработки звуки, разработка математических модулей, разработка визуальных модулей В первой части будет проанализирована предметная область. Во второй части будет проведен анализ методов разработки аудиовизуализаторов. В третьей части будет спроектирована подходящая архитектура. В последующих частях будет описаны основные сущности и результаты разработки.

Автор работы _____ _____
подпись (фамилия, имя, отчество)

Руководитель работы _____ _____
подпись (фамилия, имя, отчество)

Оглавление

Определения, обозначения и сокращения

Потоковая обработка звука - обработка звука в реальном времени

Частота дискретизации (или частота семплирования, англ. sample rate) — частота взятия отсчётов непрерывного по времени сигнала при его дискретизации (в частности, аналого-цифровым преобразователем). Измеряется в герцах.

Быстрое преобразование Фурье (БПФ, FFT) — алгоритм ускоренного вычисления дискретного преобразования Фурье, позволяющий получить результат за время, меньшее чем O(N^2) (требуемого для прямого, поформульного вычисления).

Функция оконного сглаживания - коэффициент, визуально улучшающий частотный спектр на границах разрыва окна

SFML (simple fast multimedia library) – графическая библиотека.

Rendering — «визуализация») — термин в компьютерной графике, обозначающий процесс получения изображения по модели с помощью компьютерной программы

Звуковой сервер — программное обеспечение, которое управляет использованием аудиоустройств и доступом к ним. Обычно он выполняется в качестве фонового процесса.

ALSA (англ. Advanced Linux Sound Architecture, Продвинутая звуковая архитектура Linux) — архитектура звуковой подсистемы, а также широкий набор звуковых драйверов для операционных систем на базе ядра Linux, призванный сменить Open Sound System (OSS). ALSA тесно связана с ядром Linux. ALSA — программный микшер, который эмулирует совместимость для других слоев. Также предоставляет API для программистов и работает с низкой и стабильной задержкой.

Введение

В качестве ВКР мною было выбрано создание аудиовизуальной инсталляции с использованием генеративной анимации на основе системы частиц. В ходе работы было принято решение называть программу просто “Визуализатор”. Актуальность темы можно увидеть в последней главе работы.

Целью работы, описанной в данном отчете, было проектирование и создание минимального работоспособного аудио визуализатора с использованием языка C++.

Для достижения данной цели в рамках выполнения производственной практики были поставлены следующие задачи:

- Анализ предметной области генеративного творчества
- Анализ возможных методов разработки интерактивных видеоинсталляций
- Проектирование архитектуры
- Разработка модулей потовой обработки звуки
- Разработка математических модулей
- Разработка визуальных модулей

1. Анализ предметной области генеративного творчества

Генеративное искусство относится к любой художественной практике, где программист-художник создает процесс, например, процедурное устройство, которое затем запускается в действие с определенным уровнем автономии, и в итоге создает произведение искусства. Таким образом, мы можем сказать, что это искусство, созданное с помощью кода с одной отличительной чертой: оно в той или иной мере включает в себя саморегулируемую или автономную систему.

Автономная система независимо определяет особенности произведения искусства. Случайность — один из видов автономной системы.

Добавляя случайные события в код-арт, вы получаете разные, совершенно уникальные произведения искусства каждый раз, когда запускаете скрипт. Существуют и более упорядоченные автономные системы. Например, фрактал Мандельброта, полученный из обманчиво простого уравнения.

Произведение искусства становится совместной работой компьютера и художника. При использовании автономной системы художник отказывается от контроля над творчеством, и компьютер делает это за него. Но при этом художник будет вовлечен в настройку системы для получения более желаемых результатов.

Процесс включает в себя эксперименты и счастливые случайности, что в корне меняет роль как программиста, так и художника.

2.Анализ возможных методов разработки интерактивных видеоинсталляций

Видеоинсталляция - один из возможный подтип генеративного искусства. Именно ее я и собираюсь реализовывать в ВКР.

Один из основных принципов, которым следуют художники при создании видеоинсталляции — использование пространства в качестве ключевого элемента нарративной структуры. Благодаря этому хорошо знакомое линейное кинематографическое повествование распределяется по всему пространству, обеспечивая эффект погружения. Зритель играет активную роль, поскольку именно он определяет повествовательную последовательность, продвигаясь в организованном пространстве

Иногда идея соучастия зрителей расширяет произведение до интерактивной видеоинсталляции. Также видеоряд может отображается таким образом, что зритель становится частью сюжета в качестве персонаж фильма.

Существует несколько основных методов разработки видеоинсталляций.

Первый и самый очевидный - использование игровых движков. Современные игровые движки предоставляют не только качественные графические решения, но и целый список инструментов для разработчиков. В видеоинсталляциях часто необходим элемент интерактивности, и у игровых движков есть готовые решения для реализации такого функционала. Рассмотрим основные игровые движки:

- Unity. К Unity обращаются не только молодые разработчики, но вообще львиная доля инди, особенно на iOS и Android, где Unity воспринимается как движок по умолчанию. Этому способствовало среди прочего огромное количество обучающих роликов на YouTube. В Unity нужно писать код на C#. Альтернатива — визуальное программирование

при помощи Bolt или Playmaker. Bolt и Playmaker подходят для того, чтобы на скорую руку сковать прототип и опробовать разные геймплейные задумки.

Unity — мультиплатформенный движок. Если сделаете инсталляцию для PC, потом без сложностей перебросите её на консоли, мобильные и VR-устройства. Основным же недостатком движка является рендер. Чем более фотореалистичная графика, тем проще угадать что сделана она была на Unity.

- CryEngine. Этот движок позволяет создавать пространства с высокой степенью детализации. Созданные на движке продукты известны продвинутой графикой, которую на «экстремальных» настройках не мог потянуть ни один компьютер тех лет. Для работы в CryEngine используется язык C++, однако движок часто ругают за недружелюбность к пользователю. Из всех представленных движков, этот позволяет реализовать самую качественную графику на слабом железе, и он же является наиболее сложным как в освоении, так и в использовании.

- Unreal Engine. Движок разрабатывается опытной командой, и в 2021 году вышла его пятая часть. Писать игру на этой платформе можно на C++ или используя «блюпринты» — инструмент визуального программирования.

Менее очевидные методы имеют одну общую черту - они выходят на более низкий от движков уровень абстракции. В таких методах между разработчиком и железом нет прослойки в виде движка, следовательно, разработчик создает собственные графические решения. Методология чистой разработки позволяет пользоваться сторонними библиотеками, однако тонкости реализации разработчик должен учитывать сам. Рассмотрим основные инструменты чистой разработки:

• Processing Foundation. Это группа библиотек, позволяющая художникам-разработчикам создавать предметы генеративного искусства. Первоначально созданный для использования в качестве альбома для эскизов программного обеспечения и для обучения основам программирования в визуальном контексте, Processing превратился в новую среду как для профессионалов, так и для студентов. Программное обеспечение Processing бесплатное, с открытым исходным кодом и работает на платформах Mac, Windows и GNU/Linux. С тех пор Processing превратился в инструмент развития для профессионалов, образовательный инструмент, используемый в классах по всей стране, а также в новый контекст и среду для художников. Сегодня десятки тысяч студентов, художников, дизайнеров, исследователей и любителей используют Processing для обучения, создания прототипов и производства. Библиотека доступна в различных вариантах: Processing (Java), p5.js (JavaScript), Processing.py (Python).

• SFML. Простая и быстрая мультимедийная библиотека (SFML) - это кроссплатформенная библиотека для разработки программного обеспечения разработан для обеспечения простого интерфейса прикладного программирования (API) для различных мультимедийных компонентов в компьютерах. Он написан на C ++ с привязками, доступными для C, Crystal , D, Euphoria , Go, Java , Julia , .NET. , Ним, OCaml, Python , Ruby и Rust . Экспериментальные мобильные порты стали доступны для Android и iOS с выпуском SFML 2.2.

SFML обрабатывает создание и ввод в windows, а также создание и управление контекстами OpenGL. Он также предоставляет графический модуль для простого аппаратного ускорения компьютерной графики 2D, который включает рендеринг текста с использованием FreeType, аудиомодуля, использующего OpenAL и сетевой модуль для базового

протокола управления передачей (TCP) и протокола пользовательских дейтаграмм (UDP).

- SDL. это кроссплатформенная библиотека разработки программного обеспечения разработан для обеспечения уровня абстракции оборудования для компьютерных мультимедиааппаратных компонентов. Разработчики программного обеспечения могут использовать его для написания высокопроизводительных компьютерных игр и других мультимедийных приложений, которые могут работать во многих операционных системах, таких как Android , iOS , Linux , macOS и Windows .

SDL управляет видео, аудио, устройствами ввода, CD-ROM, потоки, общий объект загрузка, сеть и таймеры. Для трехмерной графики он может обрабатывать контекст OpenGL, Vulkan, Metal или Direct3D11 (также поддерживается более старая версия Direct3d 9). Распространенное заблуждение состоит в том, что SDL - это игровой движок, но это неверно. Однако библиотека подходит для создания игр напрямую или косвенно может использоваться движками, построенными на ее основе.

Библиотека внутренне написана на C и, возможно, в зависимости от целевой платформы, C ++ или Objective-C, и предоставляет интерфейс прикладного программирования на языке C, с возможностью привязки к другим языкам.

Третьим и самым сложным методом разработки является непосредственно с графическим API. Тут бога нет. Это значит, что время на освоение и разработку может выйти далеко за пределы оговоренных сроков ВКР, зато взамен получить полностью контролируемую невероятно быструю графическую систему. Этот метод использоваться не будет, однако знания про API требуются чтобы подобрать правильную библиотеку для взаимодействия.

3. Подбор технологий для реализации

Согласно подобранным требованиям, для реализации инсталляции надо правильно подобрать технологии. Начнем с требований к графической составляющей, а конкретно к системе частиц.

Разрабатывать систему частиц я собирать с нуля, с использованием прямого программирования. Какие-либо движки не будут использованы, поскольку это скорее прерогатива художника нежели программиста.

Разработка будет проводиться на платформе Windows, следовательно, нужно подобрать графическое API и язык для эффективной работы в этой операционной системе. Возможные варианты языков:

- JavaScript
- Python
- C/C++

JavaScript это достаточно популярное решение для реализации графики. Однако, не следует забывать, что JS работает с движком WebGL (а значит рендерится в браузере), имеет динамическую типизацию и в принципе не может выполняться многопоточно. Кроме того, JS имеет огромные проблемы в вычислении чисел с плавающей точкой, что хоть и является общей проблемой любого языка, в JS выражено сильнее всего. А точность вычислений может оказаться критически важно при придании частицам физических свойств. Это самое простое в реализации решение, однако язык не сможет поддерживать должный уровень нагрузки.

Python, как интерпретируемый язык, имеет почти все те же проблемы что и JS. Однако у него есть неоспоримое преимущество – Python имеет невероятное количество математических и научных библиотек. Кроме того, на Python существует большое количество библиотек для работы с самыми разными графическими API. Хоть этот язык и может оказаться медленным,

он точно подойдет для разработки прототипа в силу своей простоты и библиотек буквально для всего.

C/C++ это компилируемый язык, на котором написано большинство игровых движков и высокопроизводительных приложений. Язык позволяет реализовывать ассемблерные вставки и вручную управлять памяти. Это может быть его слабой стороной, поскольку из-за этого программы намного сложнее реализовать. На C++ реализованы достаточно авторитетные библиотеки для работы с графикой. Он идеально подходит для нашей задачи.

Выбор библиотеки. Из всех рассмотренных библиотек для данной работы лучше всего подходит библиотека SFML, поскольку она реализована для C++, работает с OpenGL и имеет большое комьюнити и большое количество обучающего материала.

4. Проектирование архитектуры

Первым этапом создания приложения является его проектирование. Чтобы приложение спроектировать, надо составить список требований. Мое приложение должно:

- получать на вход потоковые звуковые данные
- выполнять над чистыми звуковыми данными математические преобразования, делая их удобными визуализации
- визуализировать данные

По умолчанию архитектура должна предусматривать возможность расширения рефакторинга кода.

К счастью, для таких проектов как представленный сложная архитектура не требуется. Это связано с тем что излишнее разделение ответственности уменьшает производительность. Также не ожидается что проект будет поддерживаться большой командой разработчиков, или долгое время. Потому было принято решение остановиться на следующих сущностях, можно увидеть на рисунке 1:

- Visualizer – класс оркестратор, именно через него будут вызываться методы визуализации
- AudioProcessor – семейство классов для обработки звука
- Модуль math — раздельные математические обработчики
- Mode – семейство классов, которые будут отрисовывать обработанный звук
- различные хранилища ресурсов, подключения к звуковому серверу, подключение к рендерингу экрана

Внутри Visualizer должен срабатывать так называемый “игровой цикл”. Каждый кадр входная информация и информация на экране должны обновляться.

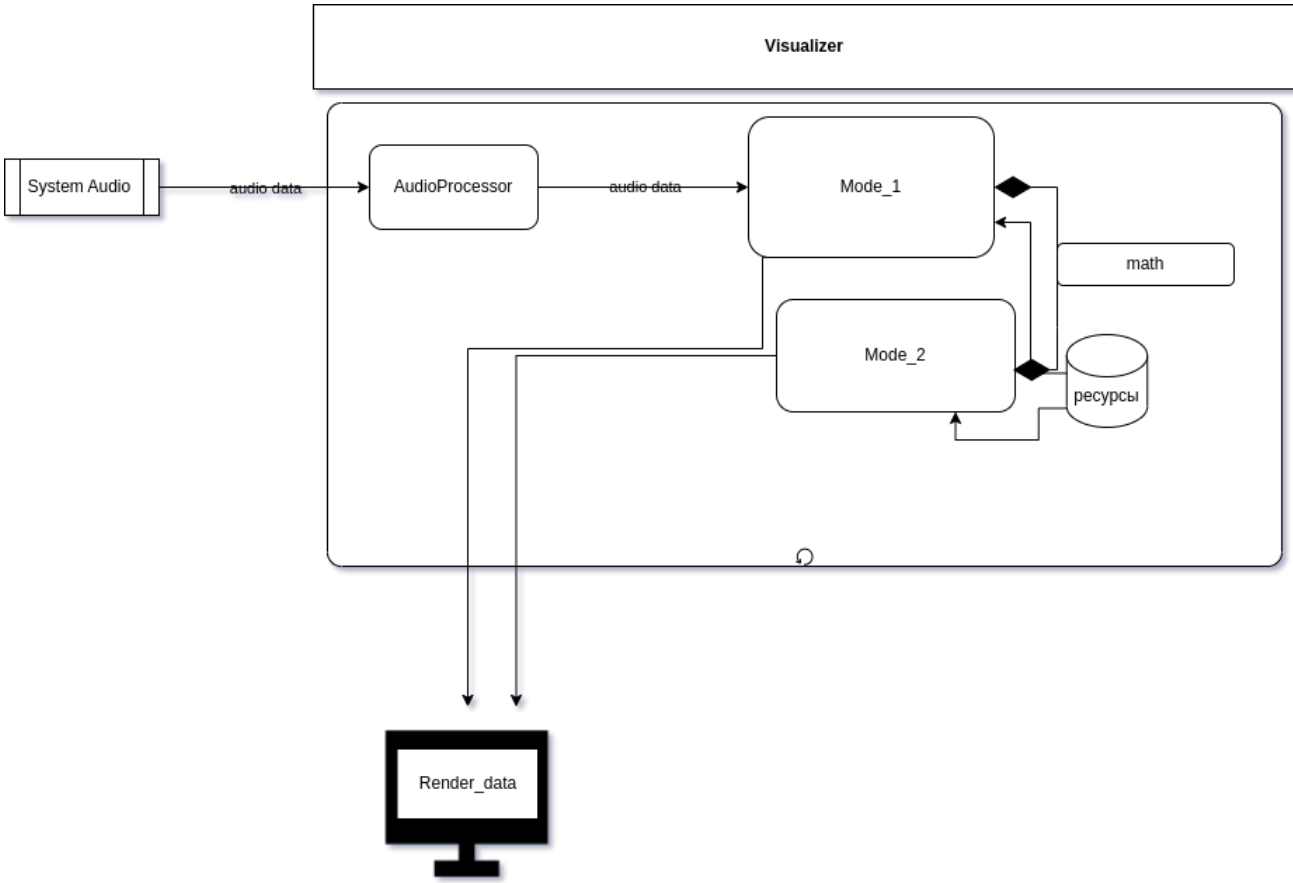

Рисунок 1 - упрощенная архитектура работы приложения

5. Разработка модулей потоковой обработки звука.

Обработчик звука должен подключаться к его источнику, считывать данные и выгружать их в удобном виде. Считывать звуки можно из разных источников, значит и AudioProcessor может быть реализован в нескольких классах. Значит, изначально необходимо создать базовый класс BaseAudioProcessor. Из функционала необходимо добавить:

- возможность считывать и возвращать в программу звук из источника
- возможность отследить время после начала считывания
- возможность узнать частоту дискретизации аудиодорожки. Это нужно для удобного чтения, а также для использования математики

Класс BaseAudioProcessor представлен ниже:

```
#pragma once

#include <vector>

//отслеживание звука будет при помощи библиотеки sfml
#include <SFML/Graphics.hpp>

using namespace std;

class BaseAudioProcessor
{
public:
//функция, чтобы узнать частоту дискретизации
   virtual double GetSampleRate() const = 0;

//две разные функции для чтения
```

```
    virtual vector<double> Read() = 0;
    virtual vector<double> Read(size_t size) = 0;

//функция для обнуления времени
    virtual void Start() {
        Clock.restart();
    }

//предусмотрена перегрузка деструктора
    virtual ~BaseAudioProcessor() {}
protected:
    sf::Clock Clock;
    double PrevTime = 0;
};
```

5.1 Обработка звука из mp3 файла.

Обработка mp3 выполняется при помощи сторонней библиотеки audiorw. На аудиопроцессору подается путь к файлу. На выходе мы имеем `vector<vector<double>>`, внутри которого хранятся левая и правая аудиодорожки, а также `double` sample_rate внутри которого хранится частота дискретизации. Обработка выполняется не потоково, а перед инициализацией отрисовки. Обращение к конкретному фрагменту аудиодорожки происходит при помощи знания о времени, которое продолжает обновляться весь игровой цикл.

Обработчик mp3 реализован в классе FileAudioProcessor

5.2 Обработка при помощи ALSA

ALSA (англ. Advanced Linux Sound Architecture, Продвинутая звуковая архитектура Linux) — архитектура звуковой подсистемы, а также широкий набор звуковых драйверов для операционных систем на базе ядра Linux, призванный сменить Open Sound System (OSS). ALSA тесно связана с ядром Linux. ALSA — программный микшер, который эмулирует совместимость для других слоев. Также предоставляет API для программистов и работает с низкой и стабильной задержкой.

Работа с ALSA выполняется при помощи библиотеки alsa, встроенной в ядро Linux. Основная сложность при работе с алса состоит в том, чтобы подобрать правильное имя девайса для подключения. Даже на слабеньком ноутбуке находится порядка 10 возможных аудиовыходов. Тем не менее, этот метод хоть и сложен в настройке, тоже работает. Обращение к конкретному фрагменту происходит в реальном времени, так как обработчик алса хранит только текущий фрагмент в виде `vector<double>`. Обработчик alsa реализован в классе PcmAudioProcessor.

5.3 Обработка при помощи PulseAudio.

PulseAudio (ранее PolypAudio) — кроссплатформенный звуковой сервер. На рисунке 2 можно увидеть уровень абстракции звуковой системы

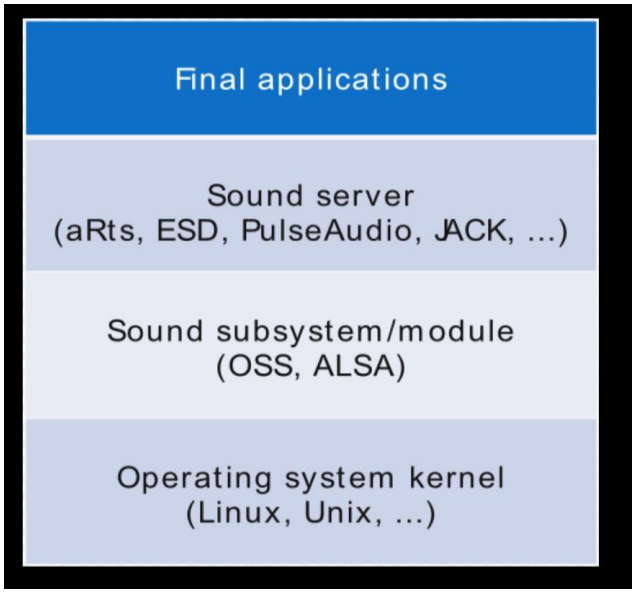

Рисунок 2. Уровни абстракции взаимодействия со звуком.

PulseAudio принимает звук от одного или нескольких источников (процессов или устройств) и направляет одному или нескольким приёмникам (звуковым платам, серверам PulseAudio или процессам). Одной из основных целей проекта является предоставление возможности перенаправления любых звуковых потоков, включая и потоки от процессов, требующих прямого доступа к аудиоустройствам.

В Linux пользователь может настроить систему ALSA так, чтобы она использовала виртуальное устройство, созданное сервером PulseAudio. Тогда программы, работающие со звуком через драйвер ALSA, будут на самом деле иметь дело с PulseAudio, который взаимодействует с помощью ALSA уже с реальным устройством.

Работа с PulseAudio выполняется при помощи библиотеки pulse, которая также встроена в ядро Linux. Обращение к конкретному фрагменту происходит так же как в alsa - в потоке хранится всего один звуковой чанк. Для PusleAudio частота дискредитации изначально установлена на значении 44100 Гц.

Обработчик PulseAudio реализован в классе PulseAudioProcessor. В процессе разработки в этом классе был реализован метод, при помощи системного вызова сам подбирающий один из трех возможных аудиовыходов (позволяет использовать блютуз девайсы). PulseAudioProcessor установлен в программе в качестве аудиопроцессора по умолчанию.

6. Разработка математических модулей.

AudioProcessor выдает голые данные о колебании мембраны звука + частоту дискретизации. Но для анализа, и, как следствие, для визуализации этого недостаточно.

Звуковые волны лучше всего анализировать при помощи быстрого преобразования Фурье. Для удобства далее в работе будет использоваться аббревиатура БПФ.

Для вычисления БПФ используется сторонняя библиотека fftw3.

Для удобного использования БПФ с библиотекой fftw3 необходимо:

- привести размер входных даннных к кратности степени 2
- применить ко входным данных функцию оконного сглаживания (необходимо для плавной визуализации)
- создать “план” БПФ, внеся него заранее данные

БПФ реализован в классе FFT

```
#pragma once

#include <vector>
#include <complex>
#include <cmath>
#include <memory>

#include <fftw3.h>

using namespace std;

class FFT
{
```

```
  public:
  //деструктор перегружен для очистки памяти
     ~FFT();

  //легаси функция для аккуратной инициализации стартовых данных БПФ
  //таких как частоты дискр. степени 2, плана и вычисления размера окна ханна
     void LazyInit(double sampleRateD);

  //преобразование Фурье происходит в этой функции
     unique_ptr<vector<complex<double>>> Calculate(vector<double>& input);

  //функция возвращает частоту дискр степени 2
     size_t GetReadSize() const;
  private:
  //в этой функции работает оконное сглаживание
     void CalculateWindowHann(const vector<double>& input);

     vector<double> _WindowHann;
     fftw_complex* OutFFT;
     fftw_plan Plan;
     size_t SampleRate = 1;
  };
```

Выполнив быстрое преобразование, объект FFT отдает `unique_ptr<vector<complex<double>>>`. Это уникальный указатель на динамический массив комплексных чисел. Уникальный

указатель тут использован для избегания копирования, а также для автоматической очистки памяти. Данные по указателю можно использовать в дальнейшем для визуализации.

7. Разработка модов визуализации.

После получения обработанных звуковых данных их можно начать анализировать и визуализировать.

Напомню, что для визуализации используется библиотека SFML.

Изобразить данные можно по-разному, от чисто научных графиков до абстрактных анимированных фигур. Для этого в проекте реализованы сущности модов визуализации. Каждый мод визуализирует данные своим способом, однако у них есть общие методы. Все моды унаследованы от базового класса AbstractMode. В базовом классе должны быть:

- функция обновления состояния
- функция отрисовки на экране
- возможность обратиться к математическому объекту

Базовый класс AbstractMode:

```
#pragma once

#include "../visualizer/Visualizer.h"
#include "../math/FFT.h"

#include <SFML/Audio.hpp>
#include <SFML/Graphics.hpp>

using namespace std;

class AbstractMode
```

```
{
public:
//конструктор будет вызывать LazyInit у объекта fft
    AbstractMode(double sample_rate_audio);

    virtual ~AbstractMode() {};

//перегруженная из SFML функция draw для отрисовки эффектов на экране
    virtual void draw(sf::RenderWindow& window) = 0;

//функция обновления состояние каждый кадр
    virtual void update(vector<double>& audio_in) = 0;

//функция для обращения во внутренний объект fft
    virtual size_t GetReadSize() const;

protected:
//математический объект
    FFT fft;

//данные для отрисовки
    vector<double> audio;

//частота дискретизации
    double sample_rate_audio;
};
```

7.1 Amplitude

Мод Амплитуда не использует БПФ - он показывает зрителю волну в чистом ее виде.

Для этого берется звуковой чанк `vector<double> data` из AudioProcessor-a, и визуализируется в виде массива точек `sf::VertexArray points`. `sf` – это пространство имен SFML. Каждая точка `points[i]` отклоняются от середины экрана на величину, пропорциональную `data[i]`. Таким образом зритель может видеть чистую волну, что можно увидеть на рисунке 3.

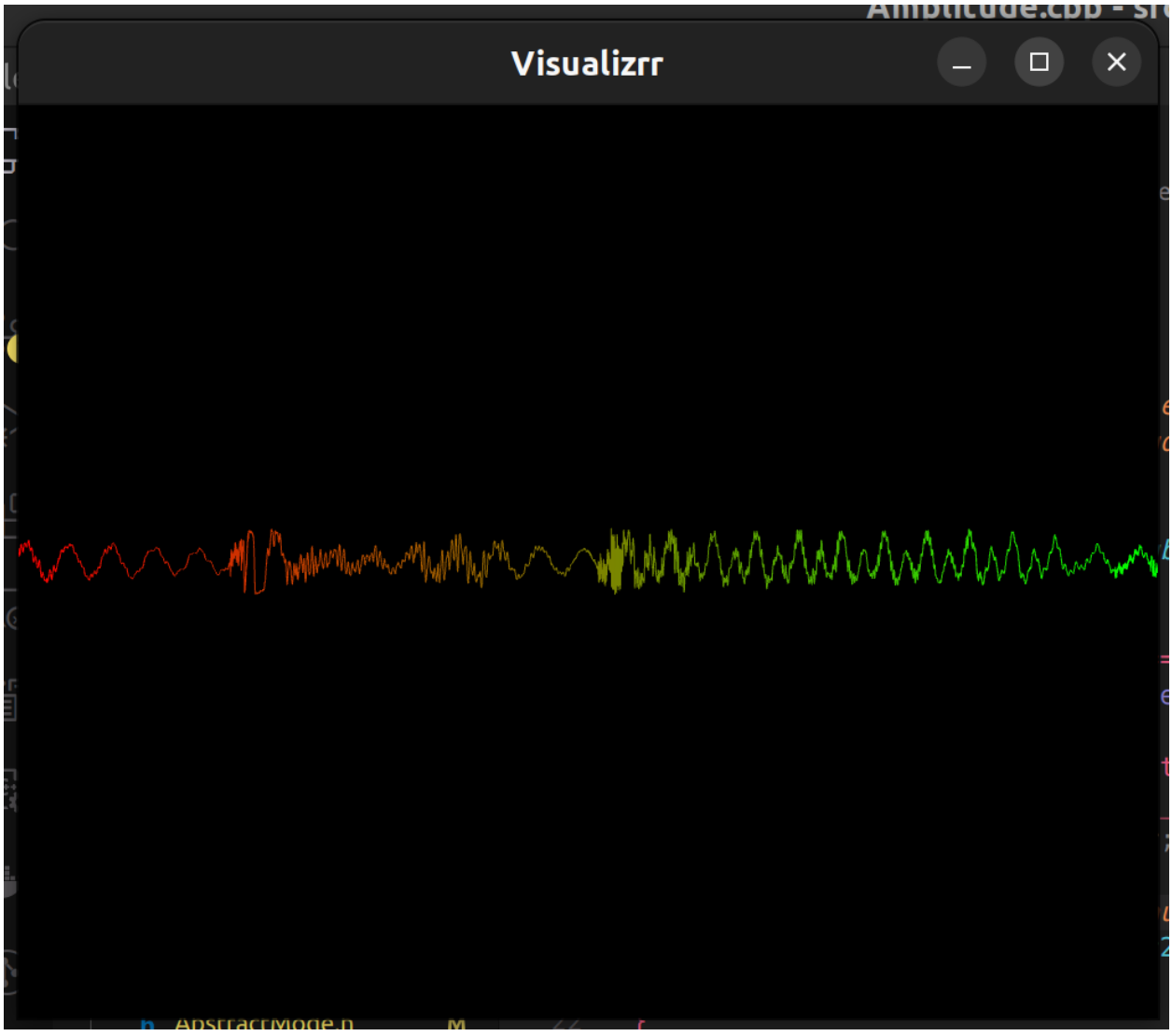

Рисунок 3. Амплитуда

7.2 Spectre

Мод Спектр использует БПФ.

Этот мод показывает то, что обычно называется “эквалайзером”. Каждой частоте на экране соответствует определенный прямоугольник. Прямоугольники отсортированы по возрастанию частоты. Высота прямоугольника интерпретирует нормализованный вектор частоты после БПФ. Его можно увидеть на рисунке 4.

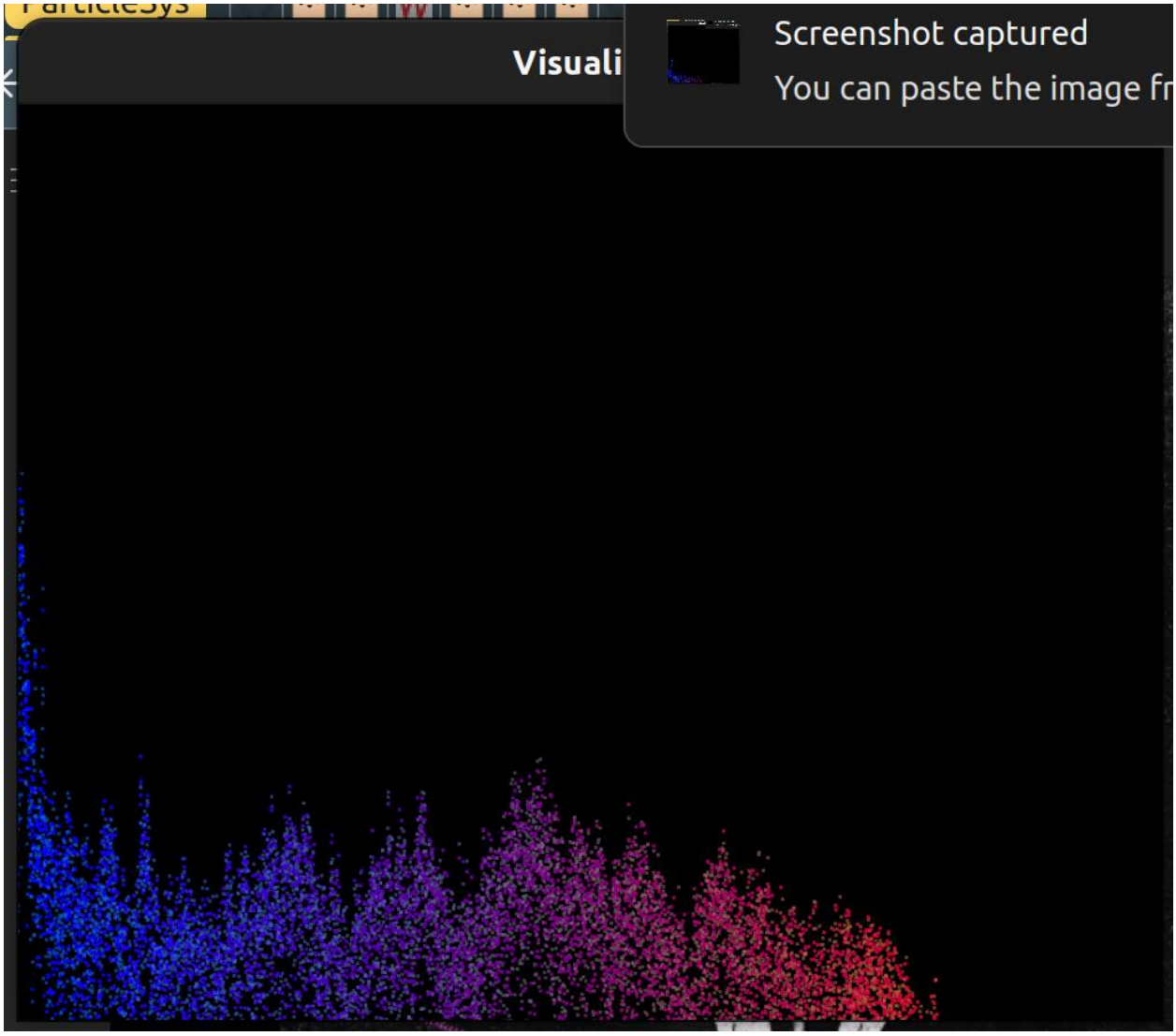

Рисунок 4. Спектр

8. Главный класс Visualizer

Назначение класса Visualizer - вызывать нужные моды отрисовки, держать открытым окно рендеринга и предоставлять пользователю возможность управления модами.

```
#pragma once

#include "../visualization_modes/AbstractMode.h"
#include "../audio_processing/PcmAudioProcessor.h"
#include "../audio_processing/FileAudioProcessor.h"
#include "../audio_processing/PulseAudioProcessor.h"

#include <SFML/Window.hpp>
#include <SFML/Graphics.hpp>

#define WIDTH 1000
#define HEIGHT 800
#define FPS 55

class Visualizer
{
private:
    sf::RenderWindow window;

    PulseAudioProcessor Reader;
public:
//в конструкторе инициализируется окно рендеринга и аудиопроцессор
    Visualizer();

//тут происходит контроль над модами и обработка действий пользователя
    void run();
};
```

9. Актуальность работы:

На первый взгляд аудио визуализаторы кажутся просто упражнением для программиста. На деле же программы визуализации звука имеют невероятно широкий спектр применения в прикладном искусстве

- повсеместное использование в колонках с экраном
- использование видеоряда для продвижения музыки и сохранения внимания слушателя на веб площадках
- использование в программах для музыкантов(Reaper, FL studio), для более наглядной и быстрой редактуры аудиодорожек
- использование огромных LED экранов на концертах любых масштабов

Актуальность приобретенных навыков:

Навыки, приобретенные в ходе разработки интерактивного аудио визуализатора являются фундаментальными для разработчиков высоконагруженных графических систем, таких как игровые движки или системы концертного оборудования. Язык C++ в этих индустриях принят как стандарт, высшая математика встречается в каждом проекте, навыки оптимизации кода и использование системных средств также обязательно для разработчиков движков и звуковиков.

Заключение

В ходе работы были выполнены все задачи и реализовано рабочее приложение.

Список источников:

1. https://www.fftw.org/fftw3_doc/ - документация по использованию библиотеки fftw
2. https://academy.yandex.ru/handbook/cpp - Основы C++
3. https://github.com/TheSalarKhan/ALSA.CPP - открытые ресурсы для использования ALSA
4. https://freedesktop.org/software/pulseaudio/doxygen/simple.html - документация по PulseAudio
5. https://github.com/sportdeath/audiorw - библиотека audiorw с документацией
6. https://www.sfml-dev.org/learn.php - документация sfml
7. https://www.calameo.com/books/005877599b3259ee0de3b - Алеся Болозя. «МАТЕМАТИЧЕСКИЕ МОДЕЛИ» 1 Способы визуализации звука и музыки
8. https://parallel.ru/fpga/fft- Основная схема быстрого преобразования Фурье