Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Магнитогорский государственный технический университет им. Г.И. Носова»
(ФГБОУ ВО «МГТУ им. Г.И. Носова»)

Институт энергетики и
автоматизированных систем
Кафедра бизнес-информатики
и информационных технологий
Направление подготовки
09.03.03 Прикладная информатика
(Информационные системы и
технологии в управлении ИТ-
проектами)

Допустить к защите
Заведующий кафедрой
_____ Г.Н. Чусавитина

15 июня 2024 г.

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

Обучающегося Егорова Михаила Игоревича

На тему: Разработка игрового движка с использованием кроссплатформенного API для 2D- и 3D-графики Vulkan

ВКР выполнена на xx страницах

Руководитель _____ 13.06.2024 доц. каф. БИиИТ, к.п.н., доц. Л.В. Курзаева

Консультант _____ 12.06.2024

Нормоконтроль и проверка
на антиплагиат выполнены.
Оригинальность текста _____ %.

_____ ___.06.2024 г. Белоусова И.Д.

Обучающийся гр. АПИб-20-2
_____М.И. Егоров
10.06. 2024 г.

Магнитогорск 2024

Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Магнитогорский государственный технический университет им. Г.И. Носова»
(ФГБОУ ВО «МГТУ им. Г.И. Носова»)

Кафедра бизнес-информатики и информационных технологий

УТВЕРЖДАЮ:
Заведующий кафедрой
_____ Г.Н. Чусавитина

31 января 2024 г.

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

ЗАДАНИЕ

Тема: Разработка игрового движка с использованием кроссплатформенного API для 2D- и 3D-графики Vulkan

Обучающемуся Егорову Михаилу Игоревичу

Тема утверждена приказом № 10-35/193 от 31.01.2024 г.

Срок выполнения 09.06.2024 г.

Исходные данные к работе:

1. Селлерс, Г. Vulkan. Руководство разработчика : руководство / Г. Селлерс ; перевод с английского А. В. Борескова. — Москва : ДМК Пресс, 2017. — 394 с. — ISBN 978-5-97060-486-1. — Текст : электронный // Лань : электронно-библиотечная система. — URL: https://e.lanbook.com/book/105835 (дата обращения: 05.04.2025). — Режим доступа: для авториз. пользователей.

2. OpenGL Documentation [Электронный ресурс] URL: https://www.opengl.org/Documentation/

3. Vulkan Documentation [Электронный ресурс] URL: https://docs.vulkan.org/

Перечень вопросов, подлежащих разработке в выпускной квалификационной работе:

1. Провести анализ проблем, связанных с разработкой игровых движков

1.1. Провести анализ текущих проблем, связанных с разработкой игровых движков

1.2. Определить потребности целевой аудитории при помощи методики CustDev

2. Осуществить постановку задачи на разработку игрового движка «LampyEngine» с использование API Vulkan.

3. Осуществить выбор и обоснования методов и средств разработки игрового движка «LampyEngine»

4. Спроектировать игровой движок

4.1. Осуществить концептуальное проектирование

4.2. Осуществить проектирования объектной модели движка

4.3. Спроектировать методы и средства рефлексивного поведения движка

4.4. Спроектировать систему рендеринга для игрового движка

5. Реализовать проектные решения по игровому движку «LampyEngine»

5.1. Разработать систему рендеринга для движка

5.2. Разработать интерфейсные формы для движка

5.3. Разработать систему рефлексии движка

6. Провести экспериментальное тестирование игрового движка «LampyEngine» и оценка его эффективности

Графическая часть:

1. UseCase диаграмма.

2. Иерархическая структура работ.

3. Диаграммы последовательностей.

4. Диаграмма активности.

5. Диаграммы классов.

6. Диаграммы компонентов

Руководитель	_____31.01.2024 г.	доц. каф. БИиИТ, к.п.н., доц. Л.В. Курзаева
Задание получил	_____31.01.2024 г.	студент группы АПИБ-21-22 М.И. Егоров

ОТЗЫВ

на выпускную квалификационную работу, выполненную обучающимся группы АПИб-21-22

Егоровым Михаилом Игоревичем

на тему: Разработка игрового движка с использованием кроссплатформенного API для 2D- и 3D-графики Vulkan

xx июхя 2025 г.

Руководитель выпускной квалификационной работы

_____ к.п.н., доцент каф. БИиИТ, к.п.н., доц. Л.В. Курзаева

Реферат

Выпускная квалификационная работа xx страниц, xx рисунков, x таблиц, xx источников.

Report

Final qualifying work xx pages, xx pictures, xx tables, xx sources.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Современные графические технологии играют ключевую роль в разработке программных решений, охватывающих такие области, как компьютерные игры, симуляторы, системы научной визуализации и образовательные приложения [5]. В центре этих решений находится графический движок — ядро, обеспечивающее визуализацию сцен, управление объектами, обработку физики и взаимодействие с пользователем. Создание собственного движка — это сложная инженерная задача, требующая глубокого понимания как архитектуры программных систем, так и графических API [8].

Одной из основных тенденций последних лет стало активное внедрение низкоуровневых API, таких как Vulkan, предоставляющих разработчику полный контроль над процессом визуализации и аппаратными ресурсами. В отличие от классических API вроде OpenGL, Vulkan позволяет достичь значительного прироста производительности за счёт таких механизмов, как предварительная запись командных буферов, эффективное управление памятью и минимизация переключений состояний [6]. Это особенно актуально при разработке сложных графических приложений, где важна максимальная производительность и масштабируемость.

Переход к использованию Vulkan требует переосмысления архитектурных решений в игровых движках. На практике выявляются ограничения традиционного объектно-ориентированного подхода к проектированию, в результате чего всё чаще применяется архитектурный паттерн Entity-Component-System (ECS), обеспечивающий гибкость, модульность и высокую степень переиспользования кода [3]. Такой подход упрощает управление большим количеством объектов в сцене и повышает эффективность параллельных вычислений на графических процессорах.

Актуальность разработки собственного игрового движка обусловлена как техническими, так и исследовательскими аспектами. Во-первых, это даёт возможность экспериментировать с современными графическими техниками, включая трассировку лучей, тесселяцию и прямой рендеринг аналитически заданных

поверхностей [1,15]. Во-вторых, создание собственного движка позволяет глубже понять процессы, лежащие в основе современных графических систем, и предложить новые решения существующих проблем, таких как поддержка векторных текстур или инкапсуляция Vulkan-визуализации в OpenGL-системы [2,5].

Кроме того, в условиях образовательной подготовки специалистов в области разработки игр и визуализации, наличие собственного движка открывает широкие возможности для учебных проектов, курсового и дипломного проектирования, а также проведения научных исследований [16]. Это позволяет студентам не только изучать готовые решения, но и самим участвовать в создании компонентов движка — от рендерера до редакторов сцен и систем взаимодействия.

Разработка игрового движка с использованием Vulkan API представляет собой актуальную и значимую задачу, направленную на создание гибкой, масштабируемой и производительной платформы для визуализации и разработки игр. Реализация такого движка позволит не только изучить современные подходы в области графических API и архитектуры движков, но и создать практический инструмент, пригодный для дальнейшего развития и применения в различных проектах.

Объектом исследования является процесс разработки игрового движка.

Предмет исследования – процесс проектирования и создания движка с использованием графических API.

Цель исследования – разработать эффективный и расширяемый инструмент для разработки видеоигр и медиаконтента.

В соответствии с поставленной целью необходимо решить следующие задачи:

1. Анализ современного состояния проблемы разработки игровых движков.

2. Осуществить постановку задачи на разработку игрового движка «LampyEngine» с использованием API Vulkan.

3. Осуществить выбор и обоснования методов и средств разработки игрового движка «LampyEngine».

4. Разработать проектные решения для игрового движка «LampyEngine».

5. Реализовать и протестировать игровой движок «LampyEngine».

6. Рассчитать экономические затраты на разработку игрового движка «LampyEngine».

В ходе работы применялись следующие методы исследования: изучение и анализ существующих решений, изучение опыта, календарное планирование, методы проектирования, а также использование программных продуктов для управления проектами Microsoft Project и проектирования PlantUML.

Практическая значимость заключается в разработанном игровом движке, отвечающем всем базовым требованиям к подобным инструментам. Дополнительно значимость достигается примененными в процессе разработке архитектурными решениями и подходами, которые могут быть полезны для образовательных проектов.

1 ТЕОРИТИЧЕСЕИЕ ОСНОВЫ ПРОЕКТИРОВАНИЯ И РАЗРАБОТКИ ИГРОВЫХ ДВИЖКОВ

1.1 Анализ современного состояния проблемы разработки игровых движков.

С начала становления игровой индустрии инструменты для их разработки появились почти сразу. Разработчиками создавались отдельные функциональные блоки для обобщения работы с конкретными элементами игры, например физика, искусственный интеллект, графика и многое другое. Такие блоки сначала содержали обычные библиотеки, объединенные для решения определенных задач при разработке игры.

Основной проблемой в то время являлось несовместимое друг с другом аппаратное обеспечение целевых платформ. К примеру с помощью AGI (Adventure Game Interpreter набор инструментов для разработки игр выпущенный в 1984 году) за 5 лет разработки было создано 14 игр для 7 разных платформ, одной из них была Manhunter 2: San Francisco выпущенная на MS-DOS, Amiga, Atari ST, Mac OS. Следовательно требование к кроссплатформенности появилось еще задолго до появления полновесных игровых движков.

Но с выходом в 1993 году шутера от первого лица Doom компании id Software и соответственно движка Doom engine ситуация на рынке изменилась. Doom engine представлял собой инструмент для создания псевдотрехмерных FPS игр. Но основной его особенностью стала модульность, в нем было несколько подсистем: системы физики, освещения, ресурсов и тд. Именно эта модульность сформировала современное определение игрового движка.

Таким образом «игровой движок – это инструмент для разработки интерактивных мультимедийных приложений и игр с графикой для разных платформ.»

В настоящее время создано большое количество игровых движков. Основными представителями, которых являются Unity и Unreal Engine. Так же существуют менее массовые только набирающие оборот движки такие как Godot. В

том числе игровые студии разрабатывают свои проприетарные движки к примеру id Tech 7, Frostbite и Red engine.

Современные движки это — набор множества модулей, подключаемых к общему центру-ядру. От ядра как точке входа движка исходят контексты редактора (инструмента для создания и описания игрового проекта) и самой игры. Важность для современной разработки движков в декомпозиции и разделении на модули привела к формированию исключительных архитектурных правил и концептов.

Из основных современных архитектурных правил можно выделить:

1. Модульность и расширяемость. Движки должны поддерживать возможность добавления и интеграции дополнительных модулей для расширения функционала;

2. Кроссплатформенность. Движок должен поддерживать множество платформ (ПК, консоли, мобильные устройства, VR/AR). Поддержка разных платформ реализуется за счет кроссплатформенных библиотек и абстрагирования графических API (Vulkan, OpenGL, DirectX, Metal) от основной логики движка;

3. Параллельность и многопоточность. Кроме того, что движок использует ресурсы видеопроцессора для рендера и каких-либо специфических задач, он должен разделять разными потоками работу независимых модулей, ради достижения комфортной работы с движком и высокой производительности;

4. Гибкость и настройка. Поскольку движок является большой высоконагруженной системой внесения даже маленьких изменений в нее может привести к частичной или даже полной перекомпиляции. Проблему конфигурирования решает конфигурационная система подгружающие требуемые настройки движка из файловых форматов XML, INI, JSON.

5. Рефлексия. Важно чтобы движок поддерживал не только гибкую настройку и конфигурирование, но и мог легко менять свое поведение и поведение игровых сценариев. Данная особенность достигается с помощью системы ре-

флексии — механизма (механизм ее работы представлен на рисунке 1), позволяющего программе анализировать и изменять собственную структуру во время выполнения. Основными способами в контексте языка c++ являются использование метапрограммирования, скриптовых языков (lua, python) и встроенных систем таких как c# mono.

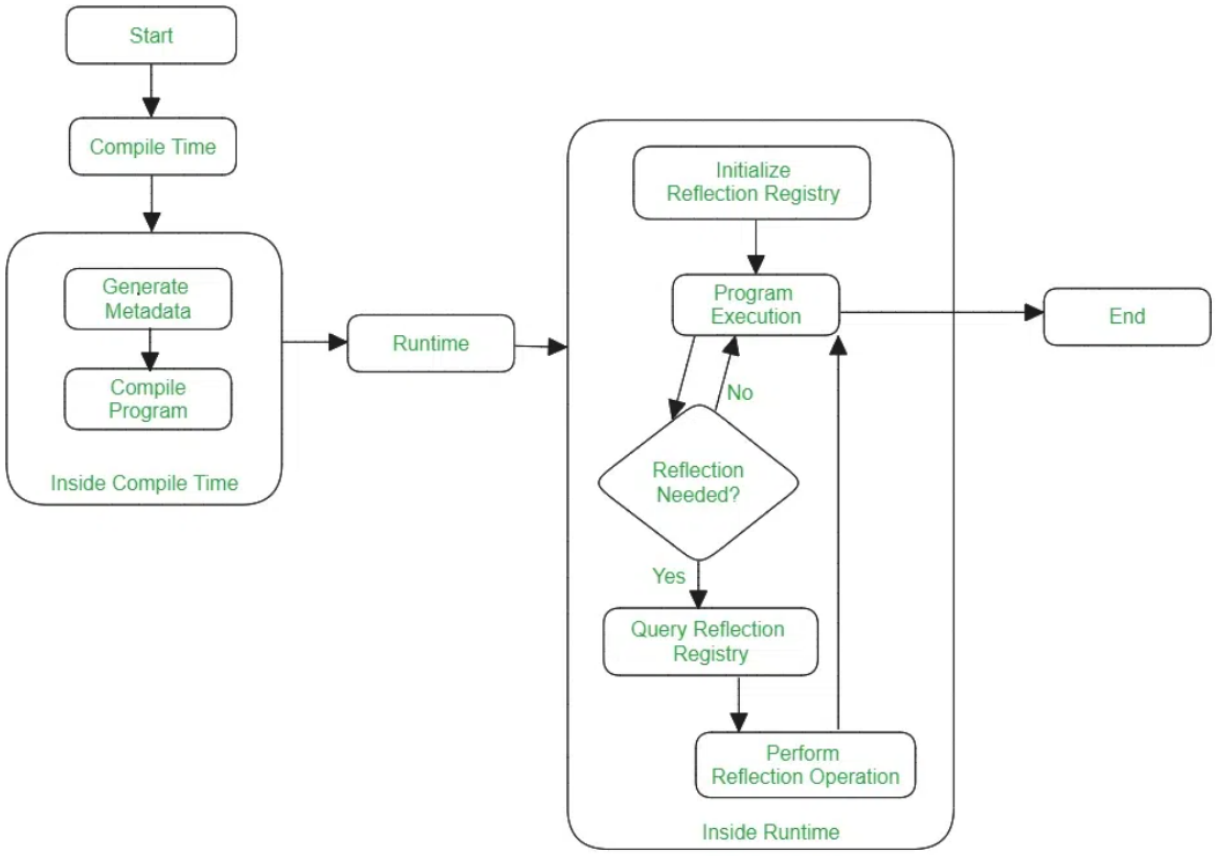


Рисунок 1 — Блок-схема работы рефлексии

Для подробного рассмотрения вышеописанных подходов, используемых в разных движках, обратимся к обзорной таблице 1 иллюстрирующей аспекты игровых движков.

В таблице приведена обзорная характеристика трёх широко используемых игровых движков: Unreal Engine, Unity и российской разработки Unigine. Таблица иллюстрирует основные архитектурные аспекты этих движков: подходы к модульности и расширяемости, кроссплатформенности, реализации параллельности и многопоточности, настройке системы и механизмах рефлексии. Для каж-

дого движка указаны конкретные методы и инструменты, применяемые при реализации обозначенных аспектов, что позволяет наглядно оценить особенности и отличия каждого из рассмотренных решений.

Таблица 1 – Сравнительная характеристика возможностей игровых движков

Движок	Модульность и расширяемость	Кроссплатформенность	Параллельность и многопоточность	Гибкость и настройка	Рефлексия
Unreal Engine	Система плагинов, модулей и слоёв абстракции. Модули могут быть подключены или отключены на этапе сборки.	• Windows/MacOs/Linux • Android/iOS • VR/AR	Многопоточность реализована на уровне движка: рендеринг, физика, загрузка ресурсов — всё работает в отдельных потоках.	Настройки хранятся в конфигурационных INI-файлах.	Используется UPROPERTY и UFUNCTION макросы, создающие метаданные для системы рефлексии. Поддержка сериализации и редактора.
Unity	Расширение осуществляется через Asset Store и пользовательские плагины на C#. Ядро закрытое, но доступны API-интерфейсы.	• Windows/MacOS/Linux • Android/iOS • VR/AR	Основные системы работают в основном потоке. Дополнительные задачи, такие как загрузка ассетов, могут выполняться асинхронно.	Настройка через XML-файлы конфигурации и редактор Sandbox.	Частичная рефлексия реализуется через экспортируемые переменные и типы. Нет полноценной поддержки RTTI на уровне C++.
Unigine(Россия)	Ядро построено по принципу архитектуры плагинов. Система позволяет расширять движок новыми модулями через C++ API.	• Windows/Linux • Android • VR/AR	Многопоточность встроена в систему ядра. Обработка сцены, рендер и физика выполняются параллельно.	Используется файл настроек проекта и конфигурационные XML/INI.	Рефлексия реализована через регистрацию классов и свойств в системе.

Разработка игровых движков сложный и многоплановый процесс. При старте проекта стоит учесть модульность, поддерживаемые платформы, возможность удобной разработки с помощью скриптов, поддержка современных графических API и другие частные аспекты.

1.2 Постановка задачи на разработку игрового движка «Lampy Engine» с использованием API Vulkan.

Движок должен быть построен с учетом модульности, позволяя подключать разные модули, такие как физика, рендеринг, скрипты, ввод и интерфейс. Так как модули могут зависеть друг от друга, важно обеспечить их загрузку в заданном порядке для корректной работы всей системы.

Необходимо предусмотреть поддержку как 2D, так и 3D графики, что позволит создавать разнообразные игровые миры и удовлетворять потребности различных жанров и стилей. Гибкость графических возможностей обеспечит разработчикам возможность создавать как простые двумерные игры, так и сложные трехмерные проекты.

Движок должен включать удобный инструмент для редактирования сцен, упрощающий процесс проектирования уровней. Такой инструмент должен иметь интуитивно понятный интерфейс, позволяющий размещать объекты, настраивать их свойства и организовывать взаимодействие между элементами сцены.

Система физики в движке должна быть тщательно проработана и включать поддержку коллайдеров, обнаружение столкновений и пересечений. Это обеспечит реалистичное поведение объектов в игровом мире, создавая динамичные и интерактивные окружения.

Для написания игровой логики необходим удобный инструмент для работы со скриптами на языке Lua. Интеграция Lua позволит разработчикам эффективно реализовывать различные механики и сценарии игры, упрощая процесс разработки и внесения изменений в игровой код.

Движок должен поддерживать интеграцию звуковых систем для воспроизведения звуковых эффектов, музыки и голосовых диалогов. Также должны быть

предусмотрены функции управления звуком, такие как регулировка громкости, пространственное звучание и поддержка различных аудиоформатов, что создаст атмосферу полного погружения для игроков.

Кроме того, движок должен включать продуманную систему управления ресурсами и встроенный файловый проводник. Система управления ресурсами должна эффективно организовывать и оптимизировать хранение игровых ассетов, таких как текстуры, модели, аудио и скрипты. Файловый проводник должен предоставлять удобный интерфейс для навигации по ресурсам, их поиска, импорта и управления, значительно упрощая разработку и сопровождение проекта.

Учитывая перечисленные функциональные и архитектурные требования к создаваемому игровому движку, а также ожидаемые сценарии его использования как со стороны разработчиков игр, так и со стороны разработчиков самого движка, необходимо определить ключевые ориентиры проекта на уровне бизнес-целей. Эти цели позволят сфокусировать разработку не только на технической реализации, но и на обеспечении практической ценности продукта для конечных пользователей, обеспечив его конкурентоспособность и применимость в реальных условиях разработки мультимедийных приложений.

Бизнес-цели:

1. Создать высокопроизводительный движок для разработки мультимедийных приложений который поддерживает разработку и сборку проектов на системах с Windows или Linux.
2. Реализовать поддержку движком Vulkan, OpenGL позволяя разработчикам выбирать наиболее подходящее API.
3. Предоставить инструменты для упрощенной интеграции скриптов, редактора сцен и физических симуляций.

Критерии успеха:

1. Расширение аудитории потенциальных разработчиков и пользователей.
2. Увеличение гибкости движка и его инструментов.

3. Сокращение времени разработки игр, что сделает движок привлекательным для инди-разработчиков и небольших студий.

В рамках проекта игрового движка предполагается существование двух функциональных ролей: разработчик игр, разработчик движка.

Исходя из требований разрабатываемый игровой движок будет:

1. Кроссплатформенным (Поддерживать как работу на Linux, так и на Windows системах)
2. Поддерживать написание скриптов на языке Lua
3. Иметь несколько графических API (OpenGL и Vulkan) для совместимости с большинством устройств
4. Базироваться на базе ECS системы для достижения максимальной производительности
5. Иметь модуль физики с интеграцией через ECS.
6. Поддерживать расширения с помощью встраивания сторонних модулей
7. Иметь открытый исходный код.

Разрабатываемый игровой движок не будет:

1. Поддерживать сборку проекта на консоли или мобильные устройства.
2. Иметь технологии для удаленной разработки.
3. Поддерживать разработку многопользовательских игр.
4. Поддерживать редактирование BIM моделей.
5. Иметь сборку под мобильные приложения.
6. Иметь поддержку проприетарных графических API таких как Direct3D(Windows и XBox) и Metal(MacOS).

Разработчик игр использует конечное приложение как инструмент для создания и отладки игровых проектов. Он взаимодействует с разными инструментами и системами движка, включая редактор сцен и систему скриптов.

Возможности разработчика игр:

- Создание сцен через специфичный редактор.

- Написание игровой и системной логики на Lua
- Работа с ECS для организации объектов
- Использование встроенной физической модели.
- Подключение расширений и сторонних библиотек.
- Сборка и тестирование игрового проекта на Windows и Linux.
- Импорт ресурсов в виде 3D-моделей текстур, звуков в поддерживаемых форматах.

Разработчик движка занимается улучшением функционала, оптимизацией работы рендеринга физики, ECS, системы ресурсов и других внутренних механизмов движка.

Возможности разработчика движка:

- Разработка и оптимизация графического API (OpenGL/Vulkan)
- Расширение и поддержка ECS-архитектуры.
- Интеграция и поддержка Lua-скриптинга включая добавления новых взаимодействий с ядром или модулей движка.
- Разработка новых редакторов и инструментов.
- Поддержка кроссплатформенности (Windows/Linux).
- Ведение документации и примеров использования.

Основные функции представлены в виде диаграммы прецедентов на рисунке 1.

Рисунок 2 – Диаграмма прецендентов игрового движка.

Таким образом были определены возможности пользователей в рамках игрового движка, обозначены бизнес-цели и критерии успеха.

1.3 Выбор и обоснование методов и средств разработки игрового движка «LampyEngine».

Создание игрового движка для 2D и 3D графики с использованием кроссплатформенных API требует тщательного подхода к выбору инструментов и технологий. Каждый компонент стека разработки должен быть выбран с учетом таких факторов, как производительность, гибкость, кроссплатформенность и

сложность реализации. В этом разделе рассматриваются возможные варианты для каждой части стека и приводятся обоснования для окончательного выбора.

В первую очередь нужно определится с языком программирования, на котором будет написанная корневая логика, системы модули и т.п. От данного выбора будет зависеть многие аспекты такие как сборка, управление проектом и кроссплатформенные зависимости. Обозначим несколько претендентов на статус проектного языка в таблице 1.

Таблица 1 – Характеристика языков в контексте разработки игровых движков

Язык	Преимущества	Недостатки
C++	• Имеет несравненную производительность и полный контроль над системой. • Несомненно, подходит для крупных игровых движков.	• Довольно сложен в изучении. • Накладываются риски утечек памяти.
Rust	• Безопасное управление памятью, современный синтаксис и высокая производительность.	• В наличии мало готовых библиотек. • Нет единого «де-факто» языкового стандарта.
C#	• Прост в освоении • Быстрая разработка • Подходит для кроссплатформенных решений.	• Не подходит для низкоуровневой оптимизации. • Зависимость от среды исполнения (.NET CLR, Mono/IL2CPP).
Python	• Максимально простоя для старта. • Быстрое прототипирование • Большое сообщество разработчиков	• Низкая производительность. • Не подходит для сложной графики.

Исходя из данной таблицы и соответствующего опыта разработки был выбран язык c++. Данный выбор был обоснован производительностью, кроссплатформенностью и наличием большего количества библиотек для работы с графическими API, окнами, аудиосистемами и т.д.

Невозможно начинать проект такого масштаба без проверенной временем системы сборки. Данные системы позволяют облегчать процесс сборки проекта, автоматизируют управление сложными конфигурациями сборки и интеграцию сторонних библиотек. Из представленных сборочных систем можно выделить CMake, Make и Meson. Рассмотрим их в сравнительной таблице 2.

Таблица 2 – Сравнение систем сборки C++

Система	Приемущества	Недостатки
CMake	Поддерживается всеми крупными IDE и CI, легко работает с кроссплатформенными проектами, умеет генерировать файлы под Make	Синтаксис неинтуитивен, отладка CMakeLists может быть сложной, требует времени на настройку больших проектов
Make	Простой и понятный механизм сборки, полный контроль над процессом, работает "из коробки" в Unix-средах	Нет встроенной поддержки кроссплатформенности, не управляет зависимостями, плохо масштабируется в больших проектах
Meson	Современный синтаксис, высокая скорость сборки с Ninja, упрощённая кроссплатформенность, удобная работа с зависимостями	Меньшая популярность в индустрии, хуже поддерживается IDE, требует установки дополнительных инструментов

Не смотря на простоту Make и высокая скорость работы Meson окончательным выбором стал CMake. В контексте игрового движка этот инструмент раскроет весь свой потенциал, в виде кроссплатформенной генерации, управлению зависимостями, интеграции с внешними библиотеками и что не мало важно поддержку многоцелевых и модульных сборок.

В рамках языка C++ несомненной проблемой до недавнего времени являлся усложненный поиск библиотек и устранение проблем с зависимостями всего решения. Но сравнительно недавно был разработан специальный инстру-

мент для поиска и организации зависимостей Conan. Conan имеет удобный инструмент упрощающий добавление и обновление библиотек, что особенно полезно в крупных проектах. Хотя vcpkg и Hunter так же являются хорошими альтернативами, но они не предоставляют такого же уровня гибкости и удобства, как Conan.

Выбор технологического стека для разработки игрового движка стал одним из наиболее значимых этапов проектирования, требующим комплексного анализа требований к производительности, кроссплатформенности, модульности и удобству сопровождения кода. В таблице 3 представлены три базовых стека, каждый из которых обладает сильными сторонами и применяется в современных игровых решениях. Так, стек WinAPI + DirectXMath + DirectX 11/12 обеспечивает высокую производительность в среде Windows, но лишён кроссплатформенности. Вариант SDL + OpenGL упрощает процесс разработки благодаря интеграции рендеринга, ввода и звука, но менее гибок в части настройки и масштабирования. Комбинация GLFW, Vulkan, GLM и OpenAL предлагает современный низкоуровневый доступ к ресурсам, при этом обеспечивая кроссплатформенность и модульность архитектуры.

Таблица 3 – Сравнение c++ стеков для разработки движков

Стек	Преимущества	Недостатки
WinAPI + DirectXMath + DirectX 11/12	• Высокая производительность • Полный контроль за ресурсами • Нативный для Windows	• Поддержка исключительно Windows систем • Управление окнами и вводом через WinAPI
SDL + OpenGL	• Кроссплатформенность • Сочетает в себе единый доступ к управлению окнами и звуком	• Ограниченные возможности по звуку • Не подходит для сложной графики

GLFW + OpenGL + GLM + OpenAL	• Прост в использовании • Отличен для обучения и небольших проектов	• OpenGL устаревает, ограниченный доступ к низкоуровневым GPU-возможностям
GLFW + Vulkan + OpenGL + OpenAL	• Максимальная кроссплатформенность • Соответствие лучшим практикам	• Высокая сложность настройки работы двух API.

Несмотря на то, что все рассмотренные стеки из таблицы 3 обладают высокой практической ценностью, в рамках настоящей разработки был выбран стек OpenGL + Vulkan + GLFW + GLM + OpenAL. Такое решение обеспечило необходимый баланс между производительностью, гибкостью и переносимостью. Vulkan используется как современный графический API для задач низкого уровня, OpenGL — как инструмент быстрой отладки и визуализации, GLFW — для управления окнами и пользовательским вводом, а OpenAL — для интеграции пространственного звука. Ключевым элементом математической части системы выступает библиотека **GLM**, полностью совместимая с синтаксисом GLSL и подходящая для работы как с OpenGL, так и с Vulkan. Выбранный стек позволяет реализовать архитектуру игрового движка с возможностью масштабирования, модульного расширения и поддержки нескольких целевых платформ.

Окончательный выбор стека технологий включает C++ в качестве языка программирования, CMake как систему сборки, Conan для управления зависимостями, Vulkan и OpenGL как графический API, GLM для математических операций и GLFW для работы с окнами и вводом. Этот набор инструментов обеспечивает высокую производительность, гибкость и кроссплатформенность, что делает его идеальным выбором для разработки игрового движка с использованием Vulkan.

ВЫВОДЫ ПО РАЗДЕЛУ 1

В результате проведённого анализа были выявлены ключевые особенности и проблемы, связанные с проектированием и разработкой современных игровых движков. Особое внимание было уделено историческому развитию индустрии, эволюции архитектурных принципов и изменению требований к функциональности подобных систем. Модульность, кроссплатформенность, многопоточность, гибкость в конфигурации и поддержка рефлексии — все эти характеристики стали неотъемлемыми атрибутами современных решений, используемых как в коммерческой, так и в образовательной среде. На основе проведённого анализа сформировалось понимание того, каким требованиям должен соответствовать игровой движок нового поколения.

Для устранения существующих ограничений и создания универсального инструмента была поставлена задача разработки собственного игрового движка «LampyEngine». Он должен обеспечить поддержку современных графических API, таких как Vulkan и OpenGL, включать систему обработки 2D и 3D графики, встроенную поддержку физики, редактор сцен и инструменты для скриптинга на языке Lua. Ключевым требованием также стала модульная архитектура с возможностью масштабирования, подключения внешних библиотек и расширения функциональности. Движок ориентирован на использование как конечными разработчиками игр, так и техническими специалистами, занимающимися расширением и адаптацией его ядра.

Для реализации поставленных задач был проведён выбор инструментов и технологий. В качестве основного языка программирования выбран C++ благодаря его высокой производительности, контролю над ресурсами и широкому набору совместимых библиотек. Система сборки CMake позволяет гибко управлять проектной структурой и обеспечивать кроссплатформенную компиляцию. Управление зависимостями организовано через инструмент Conan. Для реализации графической подсистемы использованы Vulkan и OpenGL, что обеспечивает одновременно современный подход и совместимость. GLM применяется для математических вычислений, GLFW — для управления окнами и вводом, а OpenAL

— для работы со звуком. Такое сочетание компонентов обеспечивает гибкость, масштабируемость и соответствие современным индустриальным стандартам.

Таким образом, в рамках первого этапа разработки были сформулированы цели, задачи и архитектурные требования к игровому движку, а также обоснован выбор технологического стека. Предложенное решение обеспечивает основу для создания гибкой, расширяемой и кроссплатформенной платформы для разработки мультимедийных приложений. Полученные результаты служат фундаментом для дальнейшего проектирования, реализации и тестирования «LampyEngine», направленного на поддержку современных подходов и упрощение процесса создания игр как для начинающих, так и для профессиональных разработчиков.

2 РАЗРАБОТКА ИГРОВОГО ДВИЖКА «LAMPY ENGINE»

2.1 Разработка проектных решений для игрового движка «LampyEngine»

Исходя из функциональных требований к движку была организована модульная структура, разделяющая обязанности между различными системами.

Рассмотрение архитектуры проекта стоит начать от общего к частному ради достижения полной ясности и прозрачности его внутреннего устройства. Диаграмма компонентов архитектуры приложения представлена на рисунке 2.

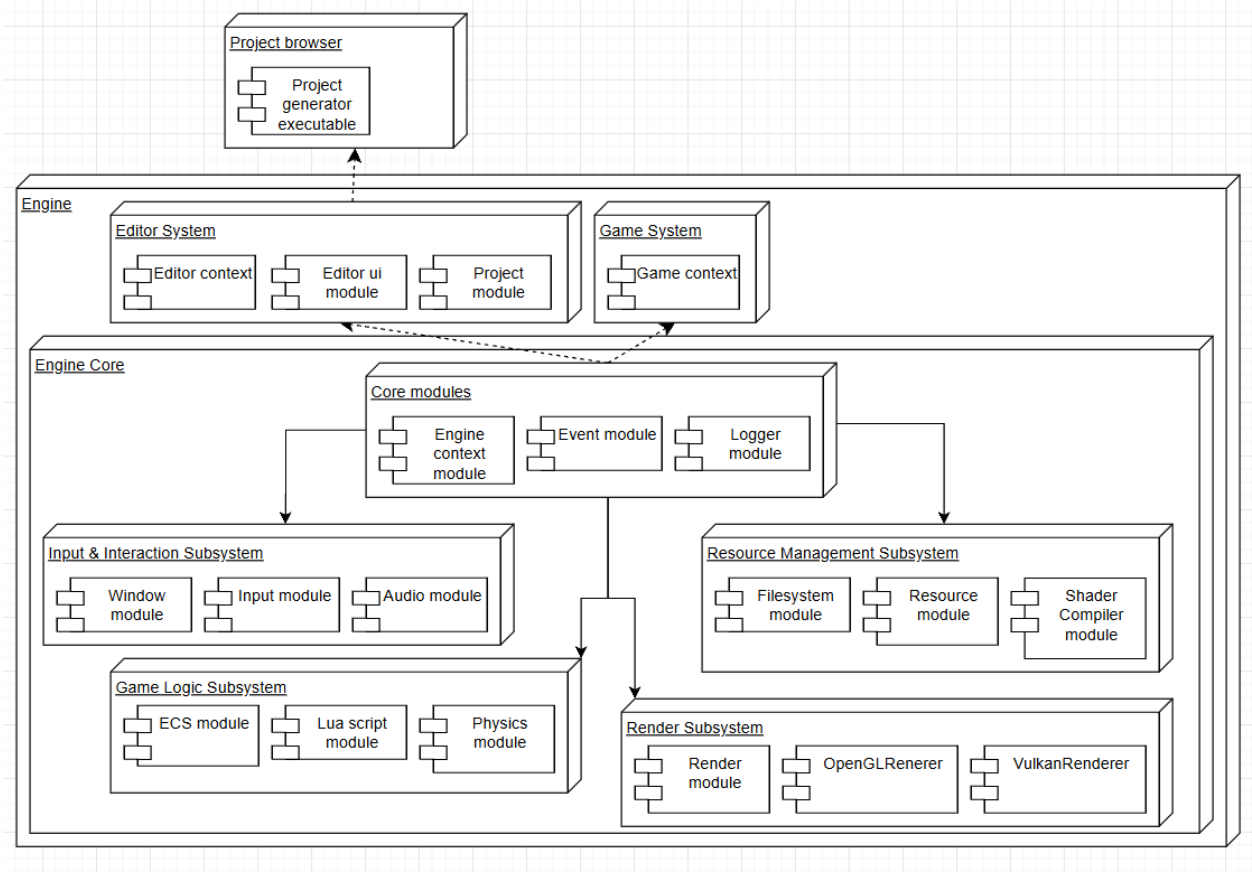


Рисунок 2 – Диаграмма компонентов игрового движка

Верхний срез архитектуры состоит из нескольких ключевых блоков:

1. Engine – главный исполняемый файл движка использующий все основные системы и модули.

2. Engine context – абстракция для создания определенных конфигурируемых режимов запуска движка. Нужный контекст создается при запуске приложения и конструируется на основе параметров запуска.

3. Editor system – система-надстройка над основным решением расширяющая основной функционал движка до полноценного редактора.

4. Project browser – отдельный исполняемый файл выполняющий задачу создания и конфигурирования проектов.

5. Game system – аналогичная Editor system настройка которая в свою очередь модифицирует основное решение до запуска определенного проекта в режиме конечной сборки.

Порядок запуска объектов верхнего уровня изображен на диаграмме последовательности.

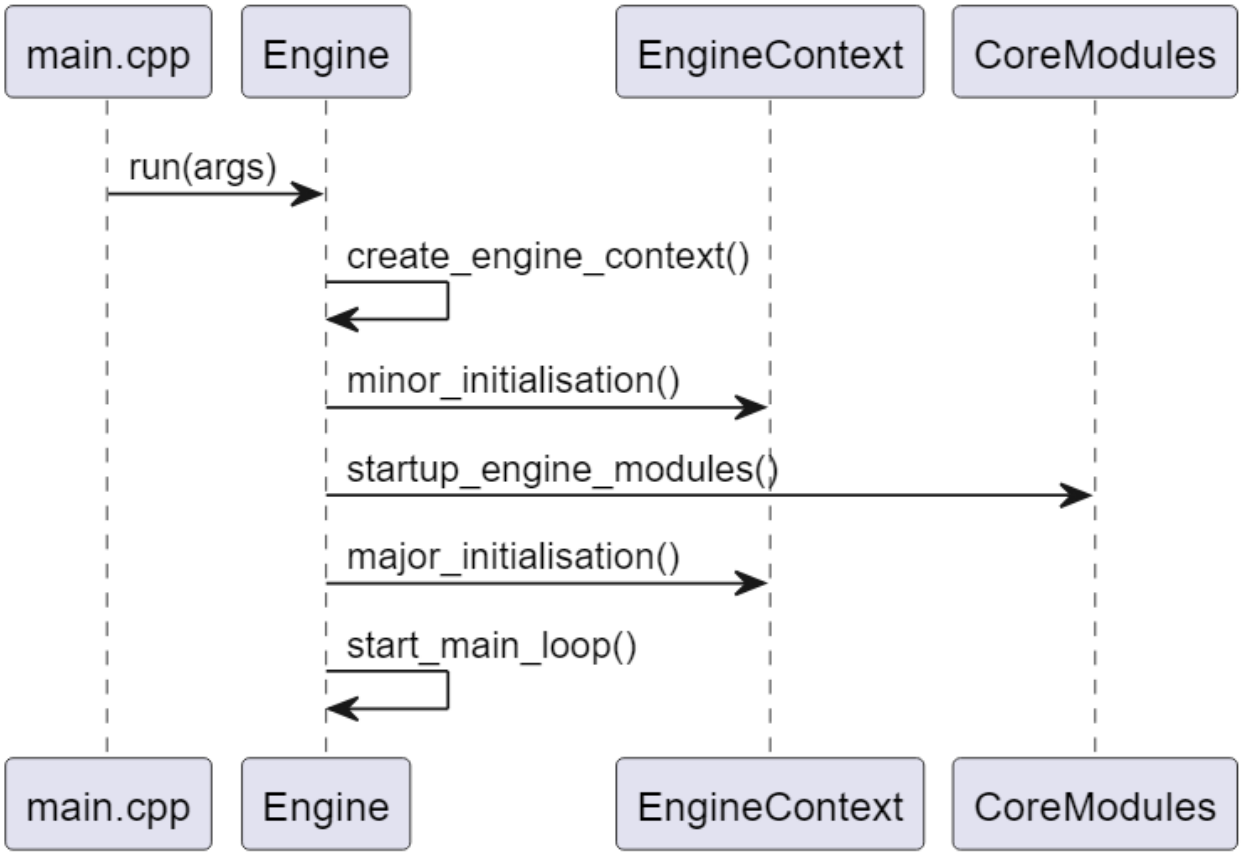


Рисунок 3 – Диаграмма последовательности запуска движка.

На рисунке 3 явно виден процесс запуска движка, где от входной точки исполнения программы поток поднимается к основному классу Engine который инициализирует контекст движка и модули ядра (Core Modules – на диаграмме последовательности запуска это обобщение над всеми основными системами движка).

Для полного понимания архитектуры движка следует рассмотреть блоки по отдельности.

Ядро движка (Engine core) служит связующим звеном для всех систем и модулей которое организует корректную последовательность инициализации подсистем приложения.

Корневые модули ядра (Core modules), организующие внутреннюю логику движка:

- Модуль контекста (Engine context module) является абстракцией для создания конфигурируемых режимов запуска. В зависимости от параметров, заданных при запуске, движок создает нужный контекст и благодаря этому способен функционировать как редактор или исполнятся как игра. Это позволяет системе сохранять модульность и модифицируемость.
- Модуль событий (Event module) основной инструмент для передачи данных между системами и модулями, он содержит в себе логику для создания различных событий, изменения передаваемых параметров и соблюдения потокобезопасности. Необходимость создания собственной реализации системы событий обусловлена контекстом языка C++ и в том числе требованием регистрации собственных событий в Lua-скриптах.
- Модуль логирования (Logger module) модуль, без которого не может существовать ни одна комплексная программа. Этот модуль предоставляет интерфейс для регистрации и сохранения сообщений во внутреннюю консоль и до записи в отдельный log файл. Модуль так же предоставляет различные категории логирования такие как verbose, debug, warning, error и fatal.

Теперь можем перейти к рассмотрению частных систем, включенных в ядро движка (Engine core), реализующих различный функционал.

Для обработки ввода и инициализации контекста взаимодействия с операционной системой существует система «Input & Interaction system» которая объединяет в себе модули окна, ввода и аудио. Эти модули напрямую взаимодействуют с целевой платформой для получения доступа к созданию окон, считыва-

ния ввода с клавиатуры, мыши или других пользовательских устройств. В искомую систему так же вынесен модуль для работы со звуками, хотя такая группировка не совсем очевидна, но в подробном рассмотрении обнаруживается, что популярные аудиобиблиотеки диктуют необходимость получения устройств воспроизведения, и в некоторых частных случаях управления параметрами конкретного аудиоустройства.

Поскольку в движке подразумевается поддержка множества API рендеринга и взаимодействия с видеокартой, подсистема рендеринга вынесена в обособленный модуль, в котором OpenGL и Vulkan реализуют и расширяют интерфейс базового класса. Так же вынесение в отдельную подсистему (Render Subsystem) было продиктовано современными требованиями к организации подобных решений.

Для создания логики работы с ресурсами движка организована подсистема управления ресурсами (Resource Management Subsystem). В данную систему включены модули для работы с файловой системой (Filesystem module), ресурсами движка (Resource module) и компилятор шейдеров (Shader compiler module). Собственная реализация методов для работы с файловой системой требуется для обеспечения кроссплатформенности и ясности. Модуль ресурсов служит публичным интерфейсом для управления ресурсами. Модуль компиляции шейдеров использует программы-компиляторы для преобразования шейдеров с языка GLSL в промежуточный язык для параллельных вычислений SPIR-V.

За игровую логику, описание объектной модели движка, а также поддержку рефлексивного поведения отвечает подсистема игровой логики (Game Logic Subsystem), объединяющая в себе модуль ECS (Entity component system), скриптов (Lua script module) и физики (Physics module). Объектная модель (ECS) регламентирует использование компонентов и сущностей мира, следствием чего является соответствующая компоновка модулей физики и скриптов.

Таким образом, ядро движка (Engine Core) собирает в себе весь функционал, где каждая подсистема предоставляет публичный интерфейс для использования в разных контекстах и формах, таких как редактор или игра.

Сам по себе движок не являет собой полезную программу или конечное решение, а лишь предоставляет инструменты для взаимодействия с миром игры, ресурсами, вводом и т.д. Основную полезную работу с помощью интерфейсов движка выполняют контексты-расширения, которые включаются в инициализацию и процесс исполнения программы как отдельные абстрактные объекты. Рассмотрим функциональные расширения в виде системы редактора (Editor System) и игры (Game System).

Система редактора (Editor System) – контекст реализующий интерактивный инструмент, использующий корневой функционал движка. Редактор содержит два ключевых модуля:

1. Проектный модуль (Project module) – реализует функционал хранения основной информации о проекте и его структуре. Для создания или открытия проектов используется отдельное приложение – проектный браузер (Project browser).

2. Модуль редакторского интерфейса (Editor UI module) предлагает набор инструментов для взаимодействия с функциями движка. Среди основных инструментов можно выделить: инспектор сцены, настройки сущностей и файловый менеджер.

Система игры (Game system) – включает в себя исключительно процесс запуска конечного проекта созданного в редакторе с учетом расположения ресурсов и файлов проекта в упакованном виде.

Архитектура движка «LampyEngine» обеспечивает явные разграничения функционала между подсистемами и позволяет адаптировать ядро под различные сценарии использования как в среде редактора, так и при запуске финального игрового проекта. Такое структурное решение дает возможность системе масштабироваться для повторного использования и интеграции новых модулей и систем. В изначальной реализации представлены основные подсистемы ядра и две системы, наделяющие движок расширяемыми вариантами исполнения.

2.2 Реализация и тестирование игрового движка «Lampy Engine».

Перейдем к рассмотрению частных реализаций крупных подсистем модулей параллельно определяя их значимость для всего решения в целом.

Реализация объектной модели движка

В процессе проектирования игрового движка одним из самых щепетильных вопросов для разработчиков является выбор объектной модели движка, от которой будет зависеть структура будущих проектов и архитектуры в целом. Существует несколько популярных объектных моделей:

1. Классическая объектно-ориентированная модель диктует жесткую иерархию объектов, исполняющих как задачу хранения данных, так и исполнения внутренней логики. Этот подход интуитивен и прост в освоении позволяя программистам быстрее вникать в архитектуру того или иного проекта, но и накладывает ограничения по масштабированию и общей гибкости.

2. Современная компонентная модель, завоевавшая свою популярность вместе с движками Unity и Godot, дает возможность собирать объекты, используя набор независимых компонентов. Компоненты модели отвечают за отдельный аспект поведения или состояния. Несмотря на возможное возникновения путаницы в связях между компонентами, этот подход дает больше контроля над структурой проекта.

3. ECS (Entity Component System) – сравнительно молодая объектная система, задачи которой разделены между тремя понятиями, где сущность (Entity) удерживает компоненты-данные (Components), обрабатываемые системами (Systems). Такой подход позволяет легко отделить логику от параметров, облегчая процесс компоновки объектов и сериализацию.

4. Прототипно-ориентированные модели дают возможность собирать варианты объектов, так как сущности создаются путем копирования и расширения объекта-прототипа. Так можно легко создавать вариации, но это может ухудшать читаемость архитектуры.

С учетом всех преимуществ по производительности, гибкости и масштабируемости была выбрана модель ECS. Однако выбор конкретной реализации

ECS играет немаловажную роль, так как требуется не только эффективность, но и удобство разработки. Среди существующих решений, таких как EnTT, specs, Bevy ECS и других, особенно удачным представляется Flecs. Flecs имеет лаконичную структуру и демонстрирует высокую производительность. Стоит отметить, что Flecs написана на C и предоставляет высокоуровневый интерфейс для C++, что делает ее оптимальной для проекта.

Теперь следует рассмотреть, каким образом Flecs ECS интегрирована в движок. Обратим внимание на модуль ECS из подсистемы игровой логики (Рисунок 4).

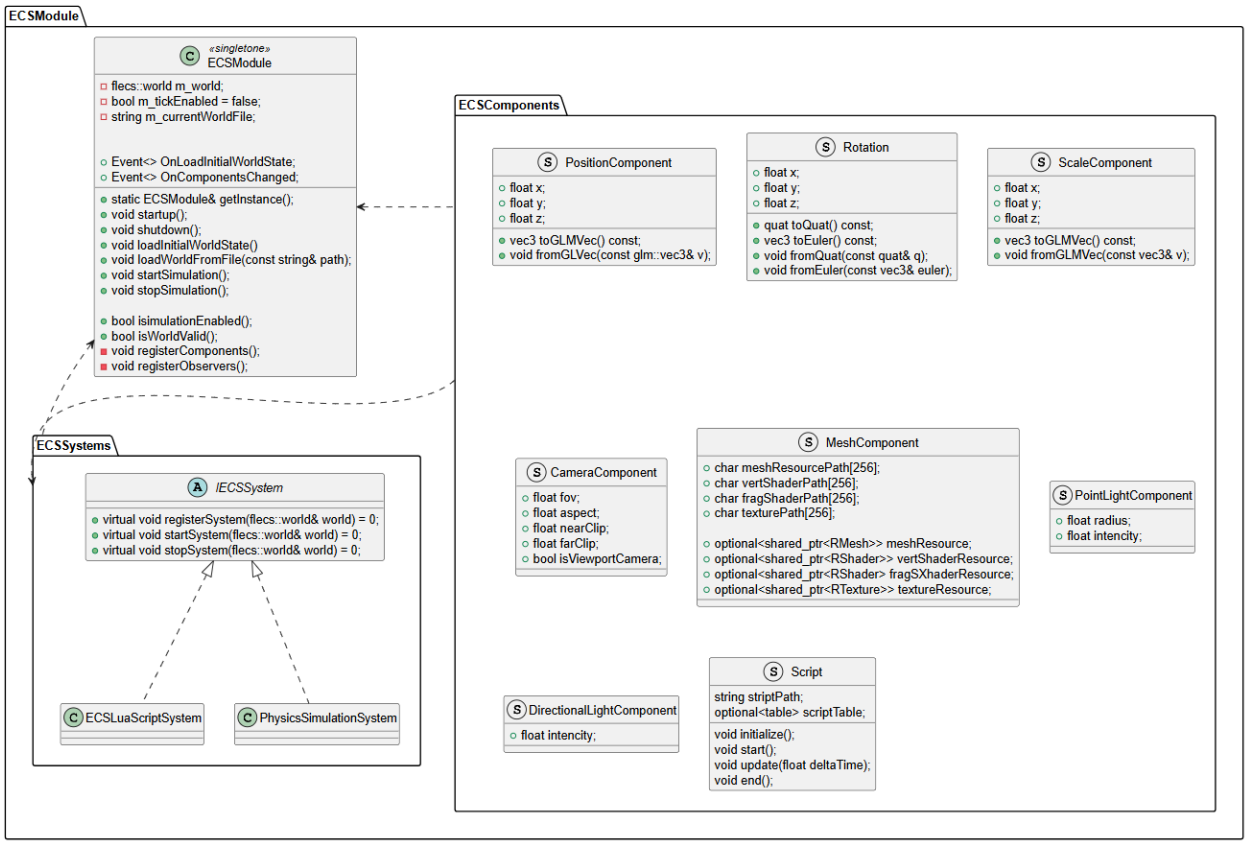


Рисунок 5 – Диаграмма классов модуля ECS

Модуль инкапсулирует в себе работу с системой ECS. Мир содержится в ECSModule классе. Компоненты регистрируются как структуры содержащие простые типы (POD – Plain old data). Для организации логики компонентов, в базовой реализации, основные системы обобщены интерфейсом IECSSystem:

- ECSLuaScriptSystem – обрабатывает логику Lua-скриптов (Script – структура-компонент, содержащая путь до файла скрипта), вызывает основные

события старта, обновления и завершения симуляции. Эта система напрямую работает с модулем Lua-скриптов.

- PhysicsSimulationSystem – глубоко интегрированная в модуль физики (Bullet) система. Она регистрирует компоненты статичных или динамичных тел в физическом мире и напрямую взаимодействует с соответствующими объектами.

В отдельном блоке диаграммы определен пакет ECSComponents, в нем расположены базовые компоненты трансформации, света, сеток и скриптов.

В процессе разработки игрового проекта инструментарий редактора (функции движка и Lua-расширения), позволяет расширять ECS системы и компоненты.

Реализация системы рефлексии

При разработке подобных инструментов, важную роль выполняет система рефлексии – механизм, позволяющий программе анализировать и изменять собственную структуру и поведение во время выполнения. Это полезно для тяжеловесных приложений, требующих перекомпиляции. В игровых движках рефлексия дает возможность управлять сущностями, параметрами компонентов, а также расширять логику игровых сценариев.

Исходя из существующих решений для организации рефлексивного поведения, каждое из которых имеет свои преимущества, представим наиболее распространённые из них в сравнительной таблице 1.

Подход	Примеры	Преимущества	Ограничения
1. Статическая генерация кода	EnTT meta, macros, codegen	Высокая производительность, прямой контроль	Требует генерации и пересборки, сложно масштабировать

2. RTTI и typeid	typeid, std::any, кастомные реестры	Использует стандартные возможности языка C++	Ограничено типами, плохо интегрируется с внешними системами
3. Встроенная мета-система	Unreal Engine (UCLASS, UFUNCTION)	Гибкая, мощная, поддерживает инструменты редактора	Высокий порог входа, сложно реализовать с нуля
4. Интеграция скриптового языка	Lua, Python, Squirrel	Изменение поведения на лету, максимальная гибкость	Дополнительный интерпретатор, требуется интеграция API

Основываясь на масштабах проекта и характеристик этих решений, выбор пал на использование скриптового языка Lua (библиотека sol2). Обусловлено это простотой интеграции с C/C++ через стандартный API и компактности ядра Lua.

Перейдем к рассмотрению интеграции Lua в движок. Представление модуля Lua (рисунок 6) состоит всего из двух компонентов: интерпретатора в RAII обертке и публичного singleton класса.

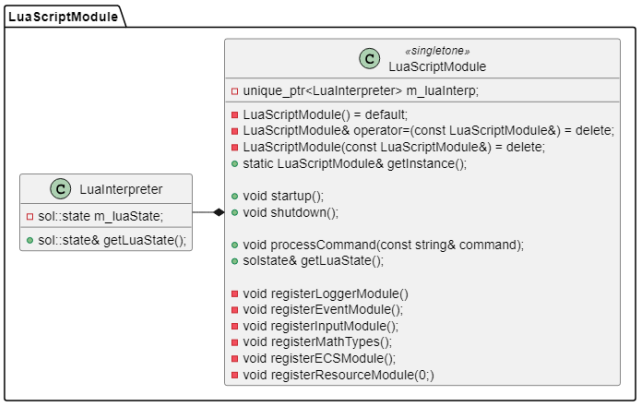


Рисунок 6 – Диаграмма классов модуля Lua

В момент инициализации LuaScriptModule создает объект интерпретатора и регистрирует методы основных подсистем и модулей. Под конец инициализации сущность интерпретатора будет иметь все основные функции и методы с привязками к их полной реализации в c++.

Но просто пробросить компоненты движка в среду Lua недостаточно, поскольку в реальной разработке программист игровой логики должен легко ориентироваться в функциях и методах предоставляемых движком. Для решения подобной проблемы необходимо грамотно подойти к написанию документации и что не мало важно предоставить автоматически генерируемый API в виде большего файла lua подключаемого к игровому проекту.

После регистрации типов и методов движка LuaScriptModule создает отдельный объект engine_api.lua представляющий собой список объявлений с документацией. Фрагмент кода генерируемого API предоставлен на рисунке 6

```lua
--- Print verbose message to log
--- @param message string
function LogVerbose(message) end

--- Print info message to log
--- @param message string
function LogInfo(message) end

--- Print debug message to log
--- @param message string
function LogDebug(message) end

--- Print warning message to log
--- @param message string
function LogWarning(message) end

--- Print error message to log
--- @param message string
function LogError(message) end

--- Print fatal message to log
--- @param message string
function LogFatal(message) end

--- Represents an event system that allows subscribing, unsubscribing, and invoking handlers.
--- @class Event
local Event = {}

--- Subscribes a function to the event.
--- @param handler fun(...):void The function to be called when the event is invoked.
--- @return number A unique subscription ID.
function Event:subscribe(handler) end

--- Unsubscribes a function from the event by its ID.
--- @param id number The subscription ID to remove.
function Event:unsubscribe(id) end

--- Invokes all subscribed functions with the given arguments.
--- @vararg any Arguments to pass to the subscribed functions.
function Event:invoke(...) end

--- Creates a new Event object.
--- @return Event A new event instance.
function Event.new() end

--- Represents the global event for key actions.
--- @type Event
OnKeyAction = Event;
```

Рисунок 7 – Диаграмма классов верхнего уровня рендеринга

Из фрагмента видны задокументированные сущности корневой подсистемы движка (модуль логирования и модуль событий). Приведена документация в виде комментариев к каждому элементу кода. Таким образом разработчику достаточно подключить файл API движка, чтобы начать комфортно работать над игровыми проектами.

Реализация структуры проектов редактора

Проекты движка и их структуру описывает система редактора и проектный модуль. При активации проектного модуля запрашивается приложение браузера проектов (рисунок 4).

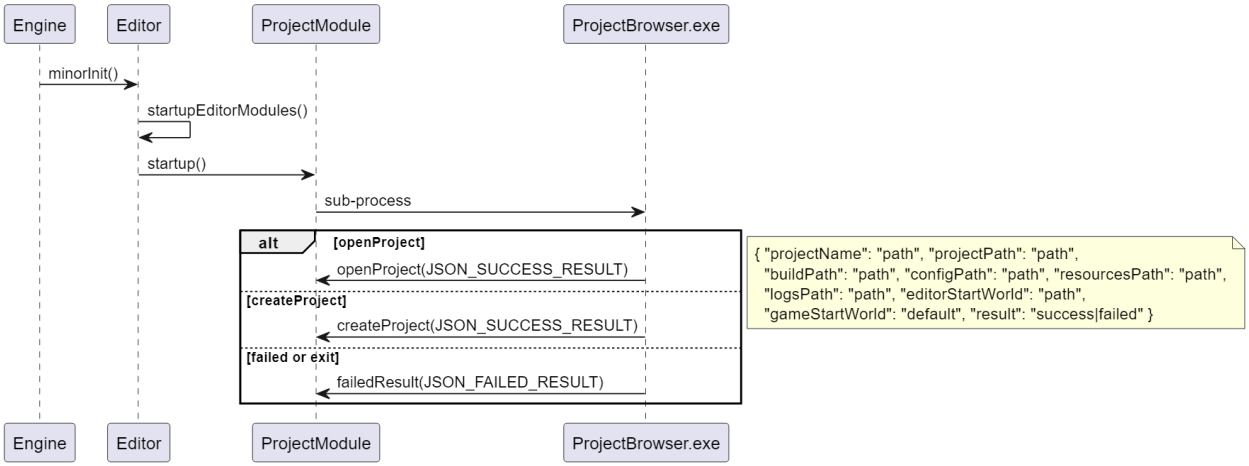


Рисунок 4 – Диаграмма последовательности выбора и создания проекта.

На диаграмме видно, что запуск браузера проекта происходит во время «минорной» инициализации контекста-редактора, и как результат он возвращает JSON структуру проекта. В JSON описаны служебные значения, относительные пути до ключевых директорий проекта таких как папки ресурсов, сборки, конфигурации и логов. Так же стоит отметить, что информация о текущих мирах движка и игры упраздняет необходимость вручную реализовывать их поиск.

Рендеринг и используемые API

В контексте нашего проекта подсистема рендеринга поддерживает отрисовку в двух API (OpenGL и Vulkan). Такое решение было принято с опорой на структуру игрового движка Godot, который следует использованию разных API для специфичных задач, где OpenGL (режим Compatibility) отвечает за совместимость с большинством устройств, а Vulkan (режим Forward+) за получение максимальной производительности и контроля над рендерингом.

Разделение между несколькими API требует организации грамотной прослойки между движком требующим отрисовку интерфейса или мира игры и конкретной реализацией рендеринга. Основной сложностью разделения OpenGL и

Vulkan является разная философия этих графических интерфейсов. OpenGL является высокоуровневой и во многих аспектах абстрактной системой, а Vulkan ближе к низкому уровню и требует практически полного контроля над основными процессами рендеринга.

Стоит ненадолго заострить внимание на работе с шейдерами в OpenGL и Vulkan, поскольку OpenGL использует GLSL формат, проводя шейдеры через этап фронтенда (компиляции на лету), а Vulkan перекомпилированный SPIR-V. Эта проблема решается использованием расширений OpenGL (например, GL_ARB_gl_spirv) позволяет использовать формат SPIR-V без передачи GLSL кода, что позволит компилировать шейдеры перед запуском конечного приложения. Однако такой подход требует поддержки со стороны драйвера и код SPIR_V должен быть совместим с OpenGL.

В текущей реализации игрового движка было принято решение построить прослойку рендеринга (IRenderer на рисунке 6) на основе абстракции объектов используемых для отрисовки на низком уровне, то есть верхний слой рендеринга повторяет структуру низкого уровня схожей с Vulkan. В такой структуре с более высокоуровневыми API, такими как OpenGL, достаточно следовать инструкциям верхнего уровня для корректной отрисовки.

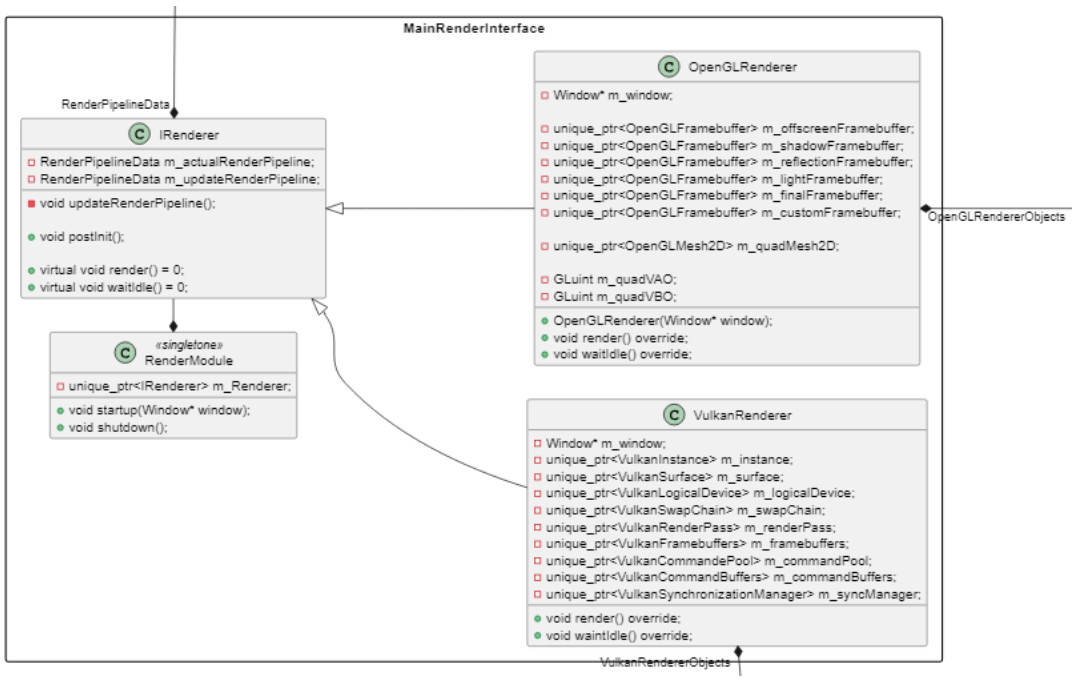


Рисунок 6 – Диаграмма классов верхнего уровня рендеринга.

Диаграмма иллюстрирует главные объекты подсистемы рендеринга, в ней публичным интерфейсом выступает singleton класс RenderModule, который содержит базовый класс-прослойку IRenderer. IRenderer в свою очередь может быть расширен до конкретной реализации OpenGL (OpenGLRenderer) и Vulkan (VulkanRenderer)

Следом стоит рассмотреть процесс передачи визуального состояния из логики движка в подсистему рендеринга для того, чтобы узнать, как актуализируется информация и отслеживаются изменения.

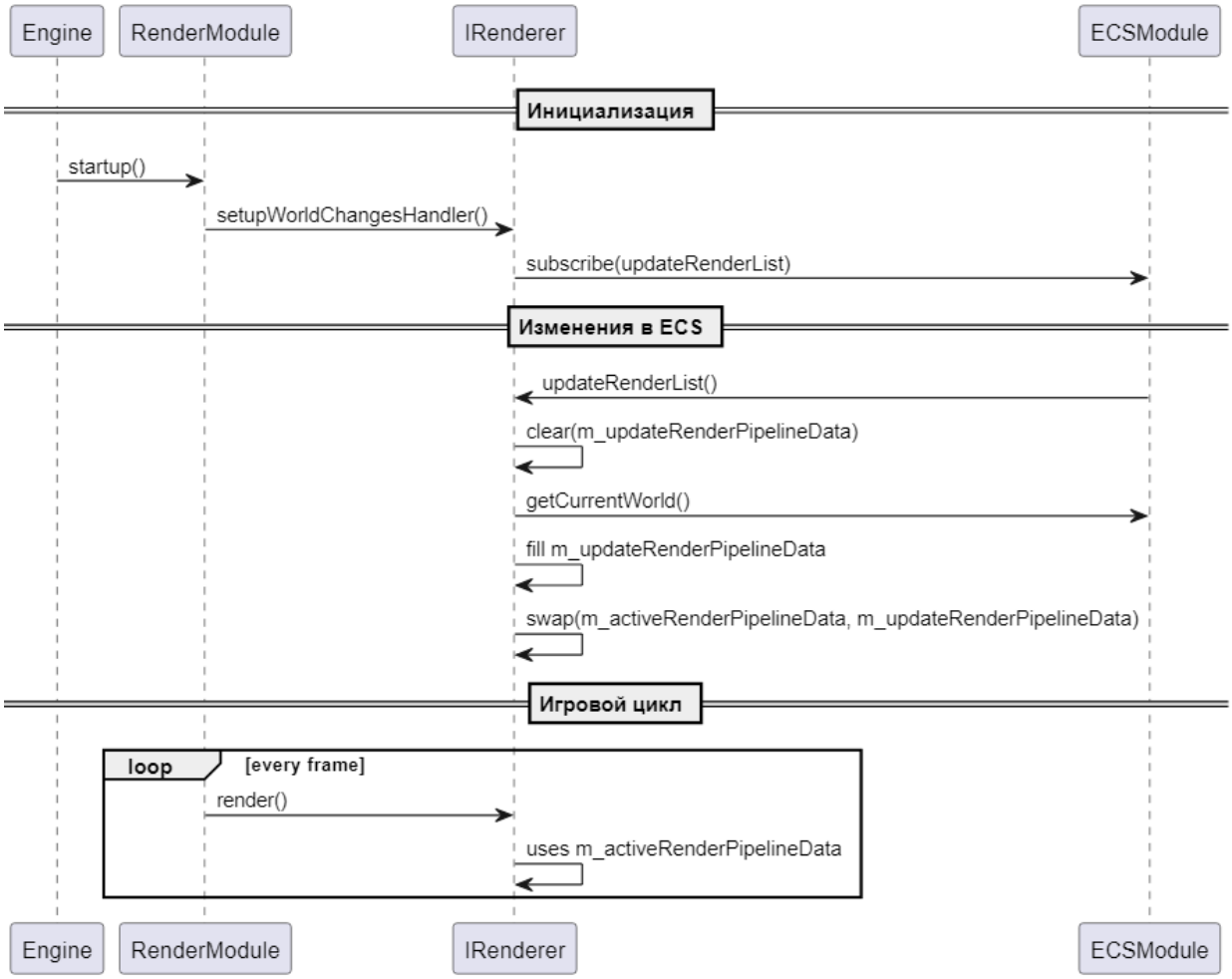


Рисунок 7 – Диаграмма последовательности запуска рендеринга и обработки состояния сцены.

На диаграмме последовательности (рисунок 7) отображен процесс синхронизации логики и рендера. Можно заметить, что при инициализации рендеринга

обобщенный класс IRenderer подписывается на изменения мира ECS. Когда состояние мира изменяется, происходит вызов метода, который подменяет текущее состояние прослойки (m_activeRenderPipelineData), используемое в конкретной реализации рендерера, на актуальное (m_updateRenderPipelineData).

Для предметного понимания того, что генерирует прослойка, разберем диаграммы классов представленных на рисунках 8 и 9.

На первой диаграмме отображены объекты, требуемые для рендеринга. Текстуры, шейдеры, сетки и материалы. Они являются абстрактными классами, где под конкретные API есть свои наследники, реализующие специфичную логику для их работы. Так же эти сущности предоставляют публичный интерфейс для работы на верхнем уровне. Каждый из этих объектов отражает структуру и состав проходов рендеринга и может быть создан с помощью фабричных классов генераторов, которые используют объекты-ресурсы и на их основе конструируют соответствующие объекты-рендера в зависимости от выбранного API.

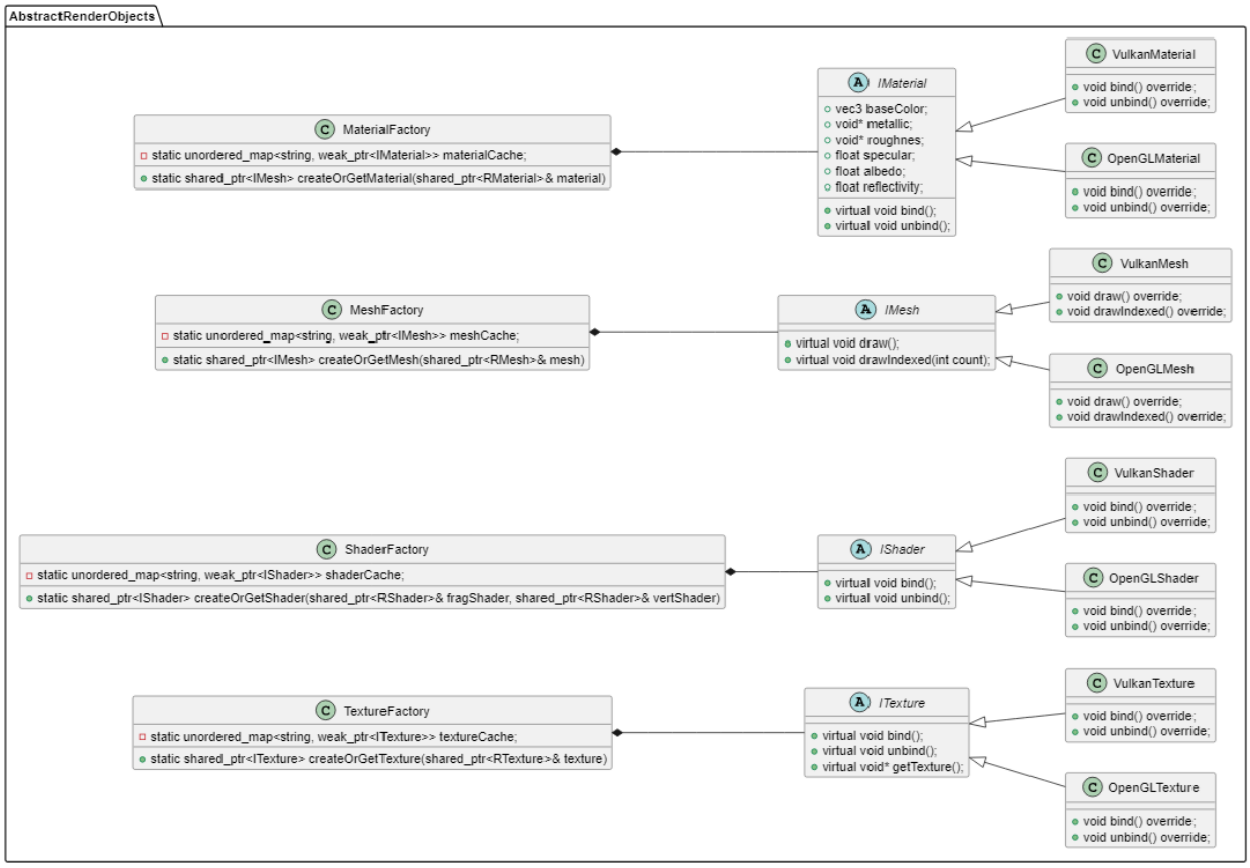


Рисунок 8 – Диаграмма классов обобщенных объектов рендеринга.

На следующей диаграмме отображены объекты прослойки. Видно, что единицей обобщения для рендеринга является структура RenderObject описывающая один вызов отрисовки и содержит в себе модельную матрицу и опциональные настройки отображения. Схожие со структурой Vulkan проходы (RenderPassData) содержат объекты для отрисовки структурированные в виде «батчей» (пакетов, содержащих от 1 до 256 объектов с одинаковой сеткой и разными модельными матрицами) и тип прохода. Конвейер отрисовки (RenderPipelineData) собирает всю информацию для отрисовки одного кадра. В ней содержится 5 рендер проходов: тени, свет, отражения, финальный и пользовательский проходы. В том числе в конвейере содержатся важные для отрисовки по типу структур камеры и объектов освещения.

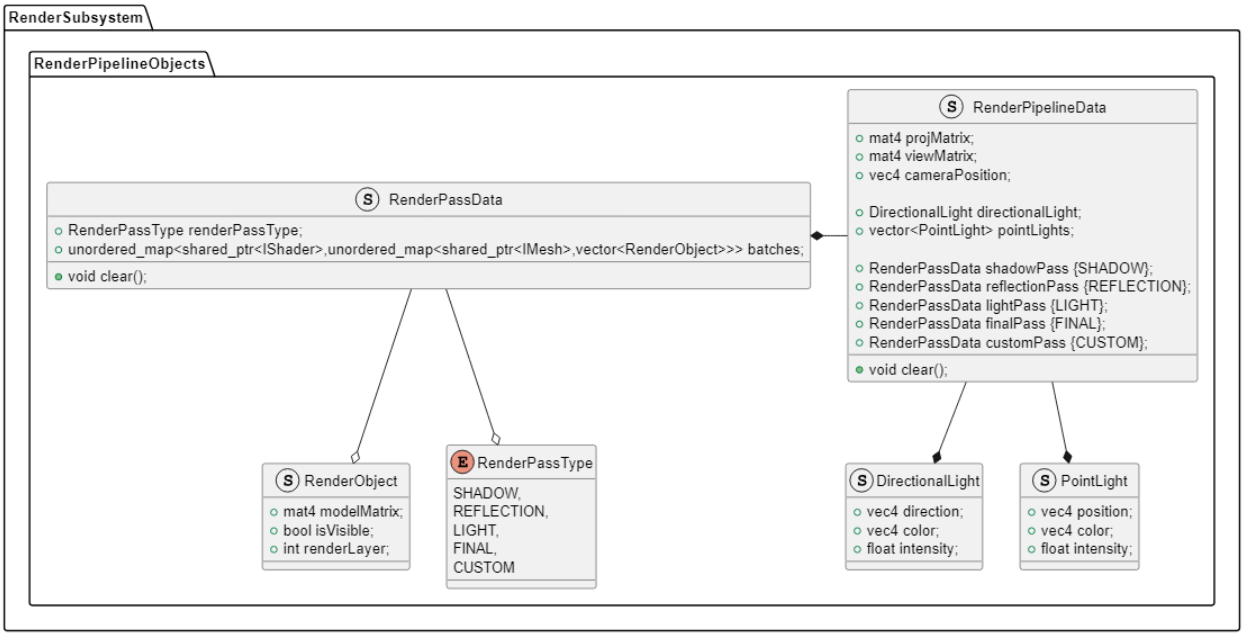


Рисунок 9 – Диаграмма классов данных рендеринга.

Теперь основываясь на информации, генерируемой на уровне прослойки и процессе ее подмена в момент изменения состояния мира, следует перейти к рассмотрению конкретных реализаций OpenGL и Vulkan.

В Vulkan особое внимание стоит уделить управлению объектами и этапам их инициализации. В базовой реализации Vulkan рендеринг в окно не подразумевается, и разработчикам требуются расширения, позволяющие использовать

цепочку кадров (swap chain) и поверхность (surface). К тому же в некоторых случаях может потребоваться внеэкранный рендеринг (offscreen renderer), для отображения сцен в отдельных окнах редактора (viewports) или генерации текстур постобработки.

Несмотря на возможность в Vulkan использования объектов сразу и последовательной их инициализации друг за другом, было принято логичное решение, исходя из большего объема кода и актуальных практик разработки, как с использованием Vulkan, так и c++ в целом, сделать RAII обертки для основных объектов. На рисунке 10 отображены основные объекты рендеринга Vulkan, такие как экземпляр API (VulkanInstance), логическое устройство (LogicalDevice), поверхность отрисовки (VulkanSurface) и другие.

Для удобства была реализована цепочка наследования RAII оберток над ключевыми буферами. От базового класса IVulkanBuffer принимающего абстрактные данные и выделяющего память логического устройства к дочерним VulkanBuffer (для передачи плоских данных в память gpu) и VulkanUniformBuffer (специализируется на формализации данных и последующей передачи в шейдерную программу). VulkanBuffer уже расширяется до VulkanIndexBuffer(хранилище данных индексов вершин) и VulkanVertexBuffer(хранилище данных вершин).

Таким образом происходит передача структур и свойств в память GPU тем самым упрощаются рутинные процессы создания буферов и ручной их настройки.

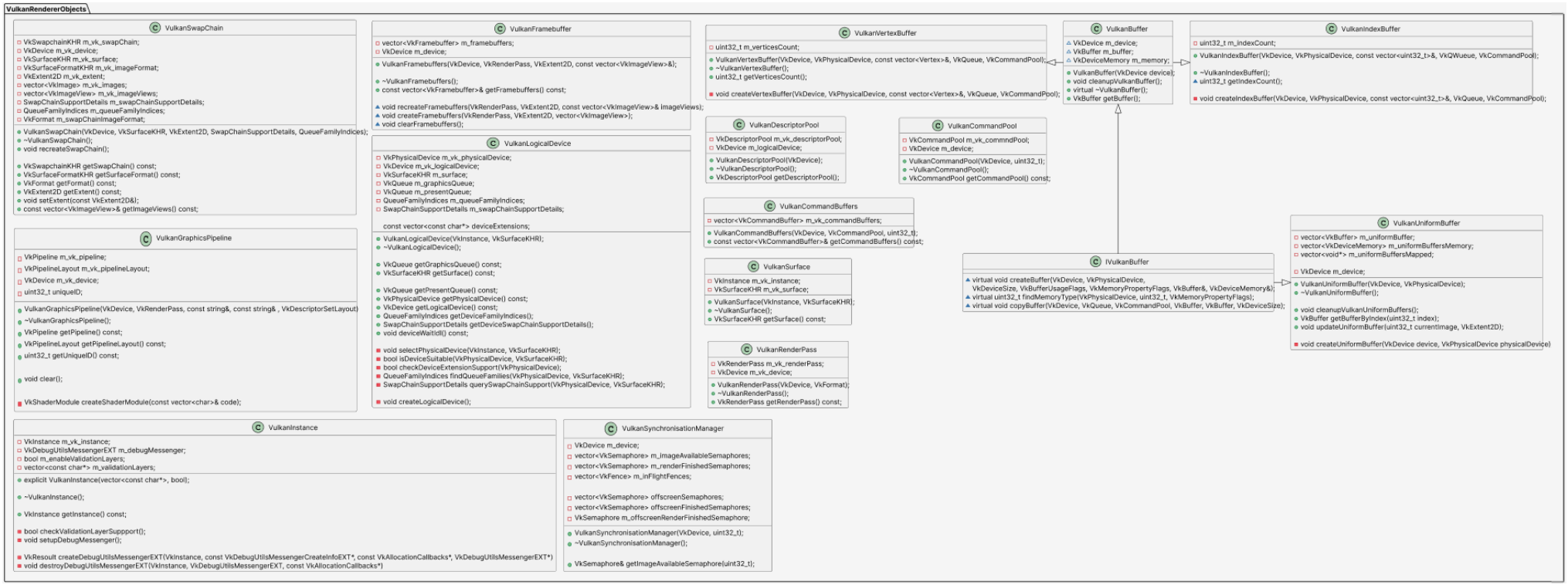


Рисунок 10 – Состав RAII-классов Vulkan.

Теперь рассмотрим последовательность инициализации объектов и их роль в рендеринге Vulkan (Рисунок 11).

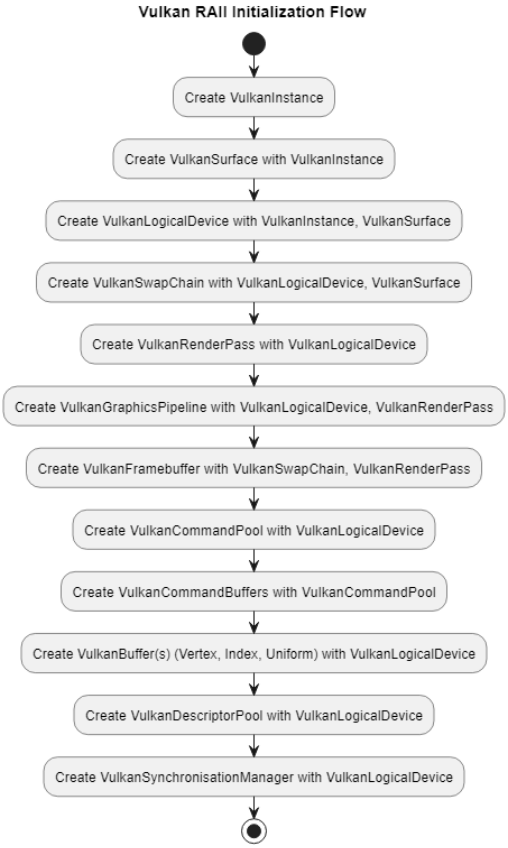


Рисунок 11 – Последовательность инициализации RAII объектов Vulkan.

Пройдем эти этапы по порядку:

1. VulkanInstance – контекст Vulkan API, с помощью которого создаются другие объекты и проверяется поддержка расширений.

2. VulkanSurface – предоставляет платформо-зависимую поверхность вывода (Win32, X11, Wayland) и выступает связующим звеном между рендерингом и оконной системой.

3. VulkanLogicalDevice – отвечает за выбор физического и создание логического устройства, через которое будут проходить команды для рендеринга.

4. VulkanSwapChain – цепочка изображений рендеринга. Количество изображений в цепочке ограничено устройством и может быть изменено через RenderConfig.

5. VulkanRenderPass – Описание структуры отрисовки, определяет какие этапы будут проходить в процессе рендера. Проходы могут быть созданы во множественном экземпляре с разными параметрами в зависимости от организации прослойки.

6. VulkanGraphicsPipline – Содержит шейдеры, входные значения вершин и описание их структуры. Объект может создаваться как на этапе инициализации, так и на этапе создания новых шейдеров.

7. VulkanFramebuffer – объект, описывающий набор целеив изображений (attachments) участвующих в процессе рендеринга, включая буферы цвета и глубины.

8. VulkanCommandPool / VulkanCommandBuffers – Выделяют память для командных буферов, где сами буферы хранят инструкции для исполнения на GPU.

9. VulkanBuffer (VertexBuffer, IndexBuffer, UniformBuffer) – Обертки, управляющие памятью GPU. Используются для передачи вершин, индексов, структур в шейдер, текстур и т.д.

10. VulkanDescriptorPool – Источник, генерирующий связи между шейдерами и uniform-буферами.

11. VulkanSynchronisationManager – Объект синхронизации, который обеспечивает правильную подачу и завершения команд между CPU и GPU.

Во время непосредственного рендеринга Vulkan следует инструкциям, созданным прослойкой, используя отдельные проходы (VulkanRenderPass) для освещения, теней и прочего. В процессе отрисовки обеспечивается передача шейдеров и вершин через VulkanGraphicsPipeline и VulkanVertexBuffer объекты.

Далее переходим к структуре OpenGL. Как было подмечено в начале главы, OpenGL поддерживает высокоуровневую архитектуру, которая легко подстраивается под данные, генерируемые на верхнем уровне. Аналогично организации RAII-оберток Vulkan определенны соответствующие классы, инкапсулирующие работу с буферами кадров, мешами, шейдерами, буферами и текстурами (Рисунок 12).

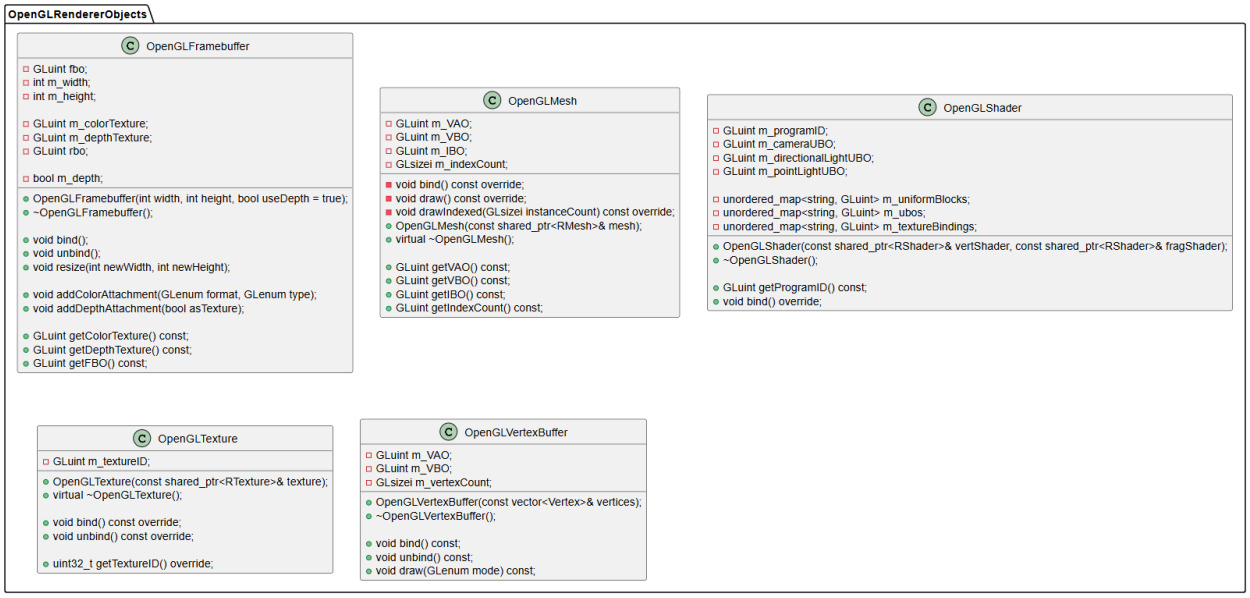


Рисунок 12 – Состав RAII-классов OpenGL.

Поскольку сам процесс отрисовки, порядок исполнения проходов и алгоритм передачи команд на отрисовку в Vulkan-реализации предметно рассмотрены не были, сделаем это в контексте обсуждения OpenGL-реализации.

На диаграмме последовательности (рисунок 13) иллюстрируются этапы обработки проходов рендеринга в OpenGL:

1. Очистка предыдущих результатов отрисовки;
2. Последовательная привязка фрейм-буферов, которые выполняют роль проходов;
    a. Запись прохода теней (ShadowPass);
    b. Запись прохода отражений (ReflectionPass);
    c. Запись прохода света (LightPass);
    d. Запись пользовательского прохода (CustomPass);
3. Отрисовка системных объектов и интерфейса редактора.
4. Завершение отрисовки и смена буферов.

Алгоритм обработки проходов использует передаваемые в метод renderPass пакеты (batches). Обработка структуры команд отрисовки отображена на рисунке 12. В цикле итерируется хеш-таблица с ключом-шейдером и значением в виде меша. Меши в свою очередь относятся к структуре RenderObject содержа-

щей модельную матрицу. Такая организация позволяет рендерить объекты пакетами используя индексированную отрисовку (drawIndexed в OpenGL). В процессе перебора для каждого «батча» активируется шейдерная программа, заполняются uniform-структуры, включая данные трансформации, и вызывается команда отрисовки.

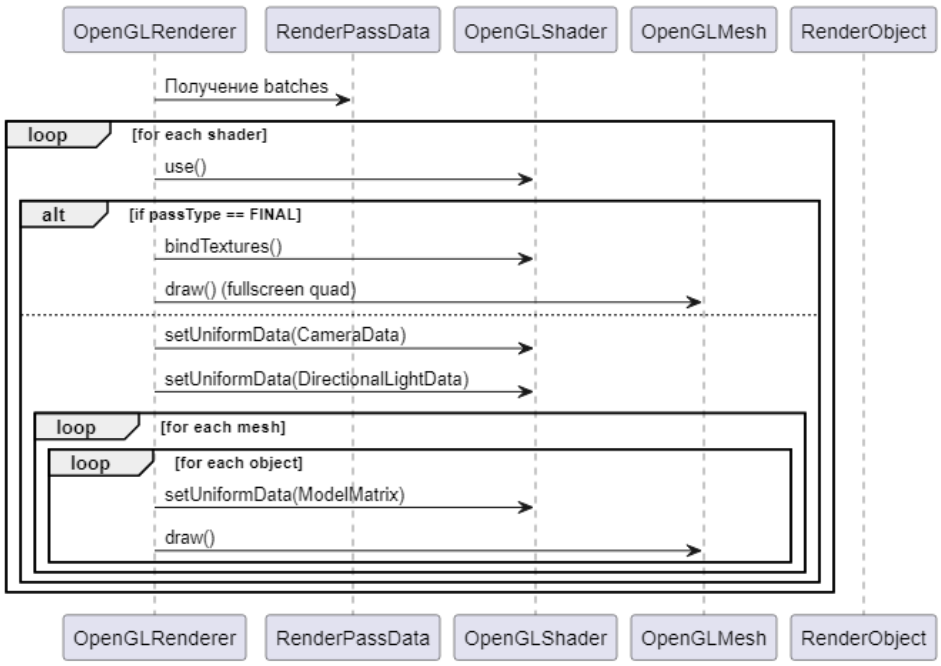


Рисунок 12 – Алгоритм исполнения команд отрисовки.

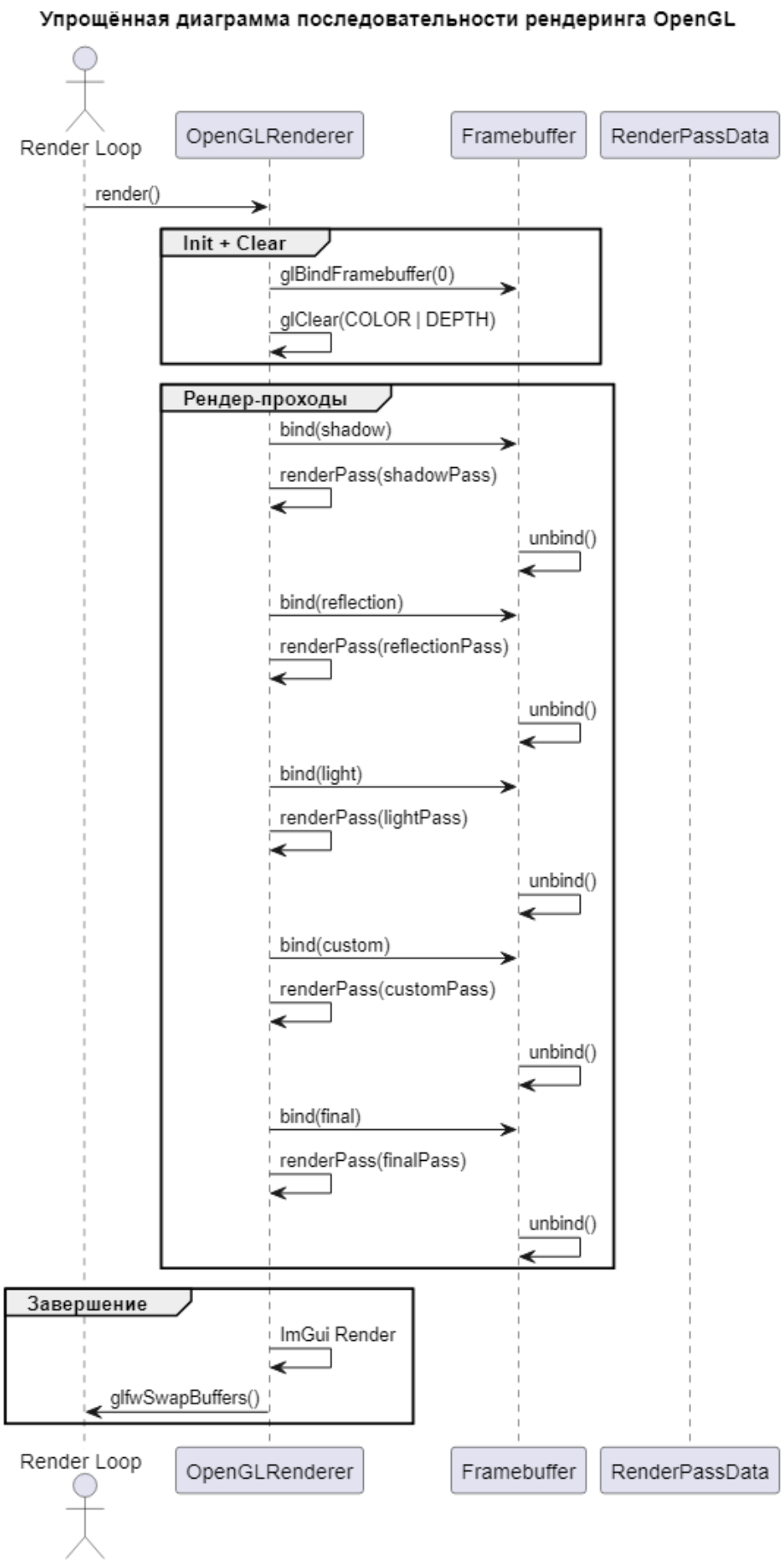
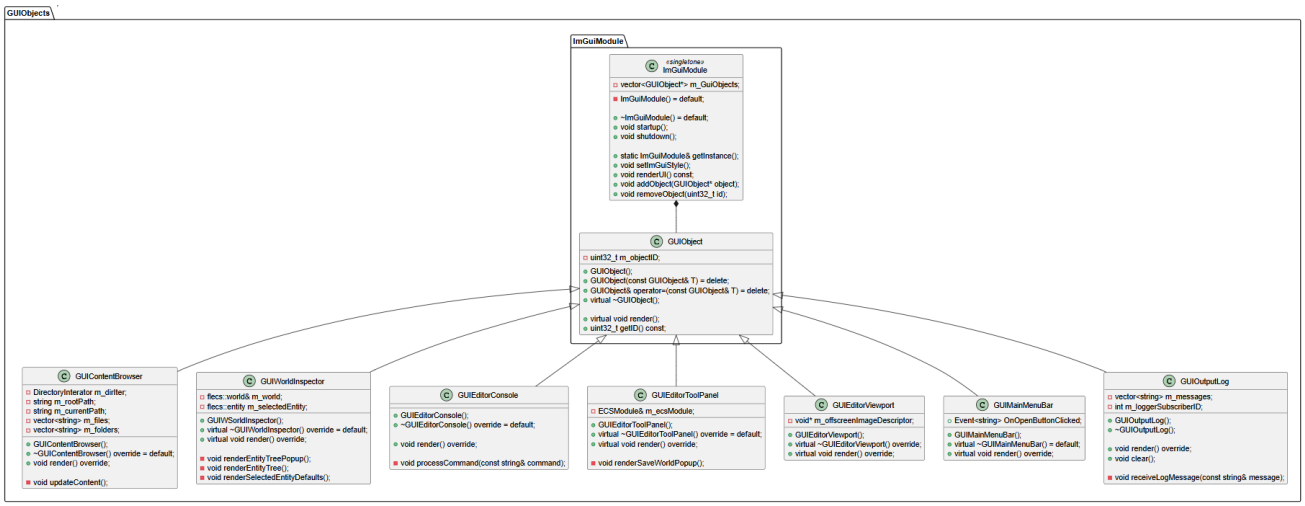


Рисунок 13 – Диаграмма последовательности рендера OpenGL.

Таким образом OpenGLRenderer следует инструкциям, составленным на верхнем уровне.

В заключение раздела можно сказать, что архитектура на основе высокоуровневой абстракции, позволяет поддерживать систему рендеринга актуальной и легко расширяемой, вплоть до добавления новых графических API. Для конфигурирования и модификации процесса рендеринга, например добавления пост-обработки или технологии трассировки лучей достаточно изменить состав проходов прослойки, не внося изменения в конкретную реализацию.

Реализация пользовательского интерфейса редактора. Интерфейсные формы.



2.3 Экспериментальное тестирование игрового движка и оценка его эффективности.

В данной главе рассматривается процесс расчета затрат игрового движка «LampyEngine». Экономические расчёты позволяют оценить рентабельность и финансовую целесообразность разработки приложения. Они помогают заранее определить затраты, потенциальную прибыль и срок окупаемости проекта.

Основными таблицами затрат будут представлены следующие:

– Рассчет совокупной стоимости владения (ССВ);

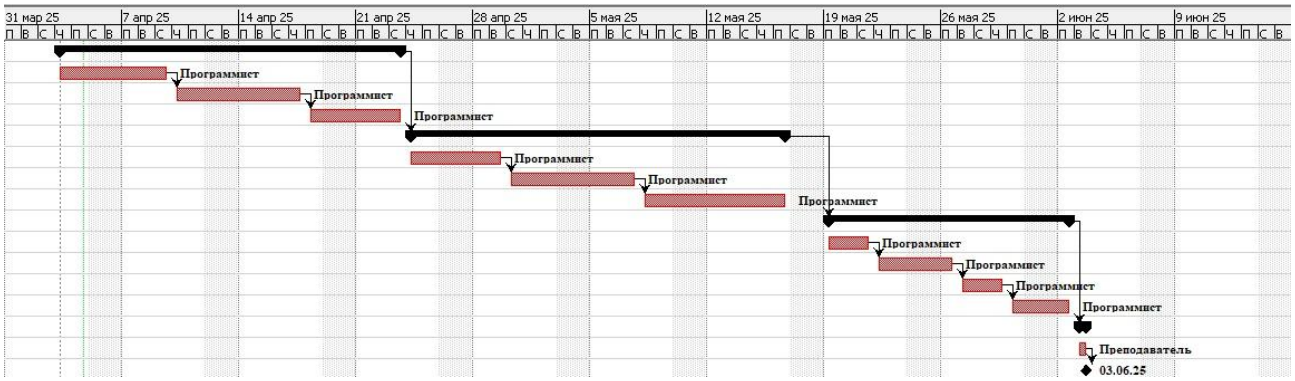
– Бюджет проекта;

– Денежные потоки;

– Экономический эффект эксплуатации приложения

Более подробно экономические расчеты продемонстрированны в Приложении В.

	Название	Продолжи...	Начало	Окончание	Название ресурса	Предшествующие
1	**Проектирование игрового движка**	15 дней	**03.04.25 8:00**	**23.04.25 17:00**	**Программист**	
2	Проектирование ядра движка	5 дней	03.04.25 8:00	09.04.25 17:00	Программист	
3	Проектирование модулей движка	6 дней	10.04.25 8:00	17.04.25 17:00	Программист	2
4	Проектирование системы рендеринга	4 дней	18.04.25 8:00	23.04.25 17:00	Программист	3
5	**Разработка игрового движка**	17 дней	**24.04.25 8:00**	**16.05.25 17:00**	**Программист**	**1**
6	Разработка ядра движка	4 дней	24.04.25 8:00	29.04.25 17:00	Программист	
7	Разработка модулей движка	6 дней	30.04.25 8:00	07.05.25 17:00	Программист	6
8	Разработка системы рендера	7 дней	08.05.25 8:00	16.05.25 17:00	Программист	7
9	**Тестирование игрового движка**	11 дней	**19.05.25 8:00**	**02.06.25 17:00**	**Программист**	**5**
10	Тестирование ядра движка	3 дней	19.05.25 8:00	21.05.25 17:00	Программист	
11	Тестирование модулей движка	3 дней	22.05.25 8:00	26.05.25 17:00	Программист	10
12	Тестирование системы рендера	3 дней	27.05.25 8:00	29.05.25 17:00	Программист	11
13	Оценка качества рендеринга	2 дней	30.05.25 8:00	02.06.25 17:00	Программист	12
14	Эксплуатация	1 день	**03.06.25 8:00**	**03.06.25 17:00**	**Программист**	**9**
15	Обучение студентов с использованием игр	1 день	03.06.25 8:00	03.06.25 17:00	Преподаватель	
16	Сопровождение системы	0 дней	03.06.25 17:00	03.06.25 17:00	Программист	15



ВЫВОДЫ ПО РАЗДЕЛУ 2

ЗАКЛЮЧЕНИЕ

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Вяткин, С. И. Прямой рендеринг трехмерных объектов на основе функций возмущения с использованием графических процессоров / С. И. Вяткин, Б. С. Долговесов // Программные системы и вычислительные методы. – 2023. – № 1. – С. 42-50. – DOI 10.7256/2454-0714.2023.1.38263. – EDN IWRNCU.

2. Sanzharov, V. V. Vector Textures Implementation in Photorealistic Rendering System on GPU / V. V. Sanzharov, V. A. Frolov, V. A. Galaktionov // Proceedings of the International Conference on Computer Graphics and Vision “Graphicon”. – 2025. – No. 32. – P. 15-25. – EDN CAHPAG.

3. Саловаров, А. Е. Использование паттерна ECS для решения проблем проектирования архитектуры игровых движков / А. Е. Саловаров, Р. С. Долгих // Кулагинские чтения: техника и технологии производственных процессов : сборник статей XIX Международной научно-практической конференции. В. 3 ч., Чита, 28–30 ноября 2019 года. Том Часть 1. – Чита: Забайкальский государственный университет, 2019. – С. 77-81. – EDN NBJOZU.

4. Мельников, В. А. Процесс разработки движка для 2D игр и интерфейсов Sad Lion Engine / В. А. Мельников // Вестник Сыктывкарского университета. Серия 1: Математика. Механика. Информатика. – 2019. – № 4(33). – С. 21-37. – EDN UJTLXQ.

5. Свидетельство о государственной регистрации программы для ЭВМ № 2023680880 Российская Федерация. Программная оболочка (капсула) для встраивания Vulkan-визуализации в OpenGL-комплексы (ПО "Инкапсулятор Vulkan-визуализации") : № 2023669870 : заявл. 26.09.2023 : опубл. 06.10.2023 / М. В. Михайлюк, П. Ю. Тимохин ; заявитель Федеральное государственное учреждение «Федеральный научный центр Научно-исследовательский институт системных исследований Российской академии наук». – EDN BHQDVE.

6. Мазовка, Д. И. Эффективная организация процесса рендеринга на графическом конвейере / Д. И. Мазовка, В. В. Краснопрошин // Международный конгресс по информатике: информационные системы и технологии : материалы

международного научного конгресса, Минск, 24–27 октября 2019 года / С. В. Абламейко (гл. редактор). – Минск: Белорусский государственный университет, 2019. – С. 968-972. – EDN XWMVTR.

7. Гонахчян, В. И. Модель производительности графического конвейера для однопроходной схемы рендеринга динамических трехмерных сцен / В. И. Гонахчян // Труды Института системного программирования РАН. – 2020. – Т. 32, № 4. – С. 53-72. – DOI 10.15514/ISPRAS-2020-32(4)-4. – EDN ZWCNWR.

8. Григорьева, Д. Ф. Программные комплексы для создания сцен освещения и 3D визуализаций / Д. Ф. Григорьева // Формирование и реализация стратегии устойчивого экономического развития Российской Федерации : сборник статей XIII Международной научно-практической конференции, Пенза, 08–09 декабря 2023 года. – Пенза: Пензенский государственный аграрный университет, 2023. – С. 135-137. – EDN CJDNSX.

9. Лоттер, Р. Blender: новый уровень мастерства : руководство / Р. Лоттер ; перевод с английского И. Л. Люско. — Москва : ДМК Пресс, 2023. — 452 с. — ISBN 978-5-93700-164-1. — Текст : электронный // Лань : электронно-библиотечная система. — URL: https://e.lanbook.com/book/348074 (дата обращения: 16.01.2025). — Режим доступа: для авториз. пользователей.

10. Морозов, В. П. Сравнительный обзор инструментов для дизайна пользовательского интерфейса / В. П. Морозов // Повышение управленческого, экономического, социального и инновационно-технического потенциала предприятий, отраслей и народно-хозяйственных комплексов : Сборник статей XV Международной научно-практической конференции, Пенза, 22–23 мая 2024 года. – Пенза: Пензенский государственный аграрный университет, 2024. – С. 216-219. – EDN CCCDUM.

11. Практическое применение нотации визуального моделирования UML в бизнес процессах : учебное пособие / Д. В. Шлаев, С. Г. Шматко, Ю. В. Орел, А. А. Сорокин. — Ставрополь : СтГАУ, 2022. — 72 с. — Текст : электронный // Лань : электронно-библиотечная система. — URL:

https://e.lanbook.com/book/323537 (дата обращения: 16.01.2025). — Режим доступа: для авториз. пользователей.

12. Сидоров, А. А. Процесс создания и визуализации объектов в 3D Max : учебное пособие / А. А. Сидоров. — Иваново : ИГЭУ, 2021. — 72 с. — Текст : электронный // Лань : электронно-библиотечная система. — URL: https://e.lanbook.com/book/296258 (дата обращения: 16.01.2025). — Режим доступа: для авториз. пользователей.

13. Сравнительный анализ платформ для разработки игр / А. А. Разживин, A. Razzhivin, Н. И. Лиманова [и др.] // Бюллетень науки и практики. — 2023. — № 7. — С. 250-252. — ISSN 2414-2948. — Текст : электронный // Лань : электронно-библиотечная система. — URL: https://e.lanbook.com/journal/issue/334985 (дата обращения: 16.01.2025). — Режим доступа: для авториз. пользователей.

14. Французов, А. М. Сравнительный анализ графических движков Unreal Engine 4 и Unreal Engine 5 / А. М. Французов, И. М. Камалутдинов // Научные открытия: междисциплинарные аспекты : Сборник статей Международной научно-практической конференции, Саратов, 20 января 2024 года. – Москва: Издательство "Доброе слово и Ко", 2024. – С. 243-249. – EDN CDDOQK.

15. Gil, I. Performance Improvement Methods for Hardware Accelerated Graphics Using Vulkan API / I. Gil // 2022 6th International Conference on Information Technologies in Engineering Education, Inforino 2022 - Proceedings : 6, Moscow, 12–15 апреля 2022 года. – Moscow, 2022. – DOI 10.1109/Inforino53888.2022.9782991. – EDN GGAWYS.

16. Del Gallego, N. P. Constructing a game engine: A proposed game engine architecture course for undergraduate students / N. P. Del Gallego // Entertainment Computing. – 2024. – Vol. 50. – P. 100657. – DOI 10.1016/j.entcom.2024.100657. – EDN WWHKAV.

ПРИЛОЖЕНИЯ

Прилоложение А
Календарный график выполнения ВКР

Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное
учреждение высшего образования
«Магнитогорский государственный технический университет им. Г.И. Носова»
(ФГБОУ ВО «МГТУ им. Г.И Носова»)
Институт энергетики и автоматизированных систем
Кафедра бизнес-информатики и информационных технологий

Календарный график

выполнения выпускной квалификационной работы

Обучающегося: Егорова Михаила Игоревича

Тема ВКР: Разработка игрового движка с использованием кроссплатформенного API для 2D- и 3D-графики Vulkan

п/п	Этапы выполнения ВКР	Дата (срок) выполнения		Отметка руководителя ВКР или заведующего кафедрой о выполнении
		план	факт	
1.	Разработка структуры ВКР. Проведение литературного обзора			
2.	Сбор фактического материала. Подготовка 1 главы рукописи ВКР (Написать название главы)			
3.	Подготовка 2 главы рукописи ВКР (Написать название главы)			
4.	Представление руководителю полностью готовой ВКР			
5.	Доработка текста ВКР в соответствии с замечаниями руководителя			
6.	Предварительная защита ВКР на кафедре			
7.	Ознакомление с отзывом руководителя и рецензией			
8.	Подготовка доклада и презентационного материала			

Обучающийся _____М.И. Егоров

Руководитель ВКР _____Л.В. Курзаева

Заведующий кафедрой БИиИТ _____Г.Н. Чусавитина

00.00.2025 г.

Приложение Б
Описание вариантов использования

Таблица Б.1 – Описание варианта использования «Создание сцен через специфичный редактор»

№ Варианта использования	Вариант использования 1
Название	Создание сцен через специфичный редактор
Автор	Егоров М.И.
Дата создания	4.01.2024
Действующее лицо	Разработчик игр
Описание	Создание сцены и объектов через встроенный редактор
Предварительные условия	Проект инициализирован
Выходные условия	Сцена добавлена и сохранена
Нормальные условия	Открытие редактора, добавление и настройка объектов
Альтернативное направление	Ошибка загрузки редактора или ресурсов
Приоритет	Высокий
Частота использования	20 раз в месяц
Особые требования	Наличие сцены и ресурсов
Замечания и вопросы	Нет

Таблица Б.2 – Описание варианта использования «Написание игровой и системной логики на Lua»

№ Варианта использования	Вариант использования 2
Название	Написание игровой и системной логики на Lua
Автор	Егоров М.И.
Дата создания	4.01.2024
Действующее лицо	Разработчик игр
Описание	Программирование логики поведения объектов и систем с помощью Lua
Предварительные условия	Загружен проект
Выходные условия	Рабочий Lua-скрипт
Нормальные условия	Открытие скрипта, написание логики, сохранение
Альтернативное направление	Ошибки выполнения скрипта
Приоритет	Высокий
Частота использования	20 раз в месяц
Особые требования	Поддержка Lua в движке
Замечания и вопросы	Нет

Таблица Б.3 – Описание варианта использования «Работа с ECS для организации объектов»

№ Варианта использования	Вариант использования 3
Название	Работа с ECS для организации объектов
Автор	Егоров М.И.
Дата создания	4.01.2024
Действующее лицо	Разработчик игр
Описание	Организация игровой сцены через сущности, компоненты и системы
Предварительные условия	Создан проект
Выходные условия	Сцена с ECS-объектами
Нормальные условия	Настройка сущностей, компонентов, логики
Альтернативное направление	Некорректные связи между сущностями
Приоритет	Средний
Частота использования	10 раз в месяц
Особые требования	ECS-поддержка в движке
Замечания и вопросы	Нет

Таблица Б.4 – Описание варианта использования «Использование встроенной физической модели»

№ Варианта использования	Вариант использования 4
Название	Использование встроенной физической модели
Автор	Егоров М.И.
Дата создания	4.01.2024
Действующее лицо	Разработчик игр
Описание	Применение физики к игровым объектам
Предварительные условия	Сцена содержит объекты
Выходные условия	Объекты реагируют на физику
Нормальные условия	Назначение масс, коллизий, запуск симуляции
Альтернативное направление	Физика работает некорректно
Приоритет	Средний
Частота использования	8 раз в месяц
Особые требования	Физика активирована
Замечания и вопросы	Нет

Таблица Б.5 – Описание варианта использования «Подключение расширений и сторонних библиотек»

№ Варианта использования	Вариант использования 5
Название	Подключение расширений и сторонних библиотек
Автор	Егоров М.И.
Дата создания	4.01.2024
Действующее лицо	Разработчик игр
Описание	Добавление дополнительных возможностей через библиотеки
Предварительные условия	Настроен Lua-интерфейс
Выходные условия	Библиотека работает
Нормальные условия	Подключение, регистрация в скрипте
Альтернативное направление	Конфликт библиотек, ошибки интеграции
Приоритет	Средний
Частота использования	6 раз в месяц
Особые требования	Совместимость с Lua API
Замечания и вопросы	Нет

Таблица Б.6 – Описание варианта использования «Сборка и тестирование игрового проекта (Windows/Linux)»

№ Варианта использования	Вариант использования 6
Название	Сборка и тестирование игрового проекта (Windows/Linux)
Автор	Егоров М.И.
Дата создания	4.01.2024
Действующее лицо	Разработчик игр
Описание	Компиляция проекта, проверка стабильности и работы
Предварительные условия	Проект завершён
Выходные условия	Создан исполняемый файл
Нормальные условия	Запуск сборки, выполнение на целевых ОС
Альтернативное направление	Сборка завершилась с ошибками
Приоритет	Высокий
Частота использования	12 раз в месяц
Особые требования	Установленные компиляторы и SDK
Замечания и вопросы	Нет

Таблица Б.7 – Описание варианта использования «Импорт ресурсов»

№ Варианта использования	Вариант использования 7
Название	Импорт ресурсов
Автор	Егоров М.И.
Дата создания	4.01.2024
Действующее лицо	Разработчик игр
Описание	Добавление 3D-моделей, текстур и звуков
Предварительные условия	Проект загружен
Выходные условия	Ресурсы импортированы
Нормальные условия	Использование менеджера ресурсов
Альтернативное направление	Неподдерживаемый формат или ошибка импорта
Приоритет	Средний
Частота использования	10 раз в месяц
Особые требования	Поддержка нужных форматов
Замечания и вопросы	Нет

Таблица Б.8 – Описание варианта использования «Разработка и оптимизация графического API (OpenGL/Vulkan)»

№ Варианта использования	Вариант использования 8
Название	Разработка и оптимизация графического API (OpenGL/Vulkan)
Автор	Егоров М.И.
Дата создания	4.01.2024
Действующее лицо	Разработчик движка
Описание	Настройка и оптимизация рендеринга с использованием OpenGL и Vulkan
Предварительные условия	Имеется сцена или тестовый пример
Выходные условия	Стабильная и производительная отрисовка
Нормальные условия	Инициализация API, реализация рендер-пайплайна
Альтернативное направление	Падение производительности или сбой инициализации
Приоритет	Высокий
Частота использования	10 раз в месяц
Особые требования	Поддержка Vulkan и OpenGL драйверами
Замечания и вопросы	Необходима отладка на разных системах

Таблица Б.9 – Описание варианта использования «Расширение и поддержка ECS-архитектуры»

№ Варианта использования	Вариант использования 9
Название	Расширение и поддержка ECS-архитектуры
Автор	Егоров М.И.
Дата создания	4.01.2024
Действующее лицо	Разработчик движка
Описание	Добавление новых компонентов, систем и улучшение производительности ECS
Предварительные условия	ECS-ядро реализовано
Выходные условия	Обновлённая ECS с новым функционалом
Нормальные условия	Реализация новых сущностей, систем, профилирование
Альтернативное направление	Конфликты между компонентами
Приоритет	Высокий
Частота использования	8 раз в месяц
Особые требования	Стандартизация формата компонентов
Замечания и вопросы	Необходимы юнит-тесты компонентов

Таблица Б.10 – Описание варианта использования «Интеграция и поддержка Lua-скриптинга»

№ Варианта использования	Вариант использования 10
Название	Интеграция и поддержка Lua-скриптинга
Автор	Егоров М.И.
Дата создания	4.01.2024
Действующее лицо	Разработчик движка
Описание	Обеспечение связи между ядром движка и скриптовой частью
Предварительные условия	Подключена Lua-библиотека
Выходные условия	Возможность писать сценарии поведения
Нормальные условия	Создание биндингов, регистрация функций и событий
Альтернативное направление	Скрипт не выполняется из-за ошибки движка
Приоритет	Средний
Частота использования	6 раз в месяц
Особые требования	Lua 5.x / совместимая версия
Замечания и вопросы	Желательна документация по API

Таблица Б.11 – Описание варианта использования «Разработка новых редакторов и инструментов»

№ Варианта использования	Вариант использования 11
Название	Разработка новых редакторов и инструментов
Автор	Егоров М.И.
Дата создания	4.01.2024
Действующее лицо	Разработчик движка
Описание	Создание пользовательских инструментов для работы со сценами и ассетами
Предварительные условия	Базовый интерфейс редактора
Выходные условия	Новый функциональный инструмент интегрирован
Нормальные условия	Разработка UI, привязка к системам движка
Альтернативное направление	Ошибки взаимодействия с внутренним API
Приоритет	Средний
Частота использования	4 раза в месяц
Особые требования	Совместимость с архитектурой редактора
Замечания и вопросы	Нет

Таблица Б.12 – Описание варианта использования «Поддержка кроссплатформенности (Windows/Linux)»

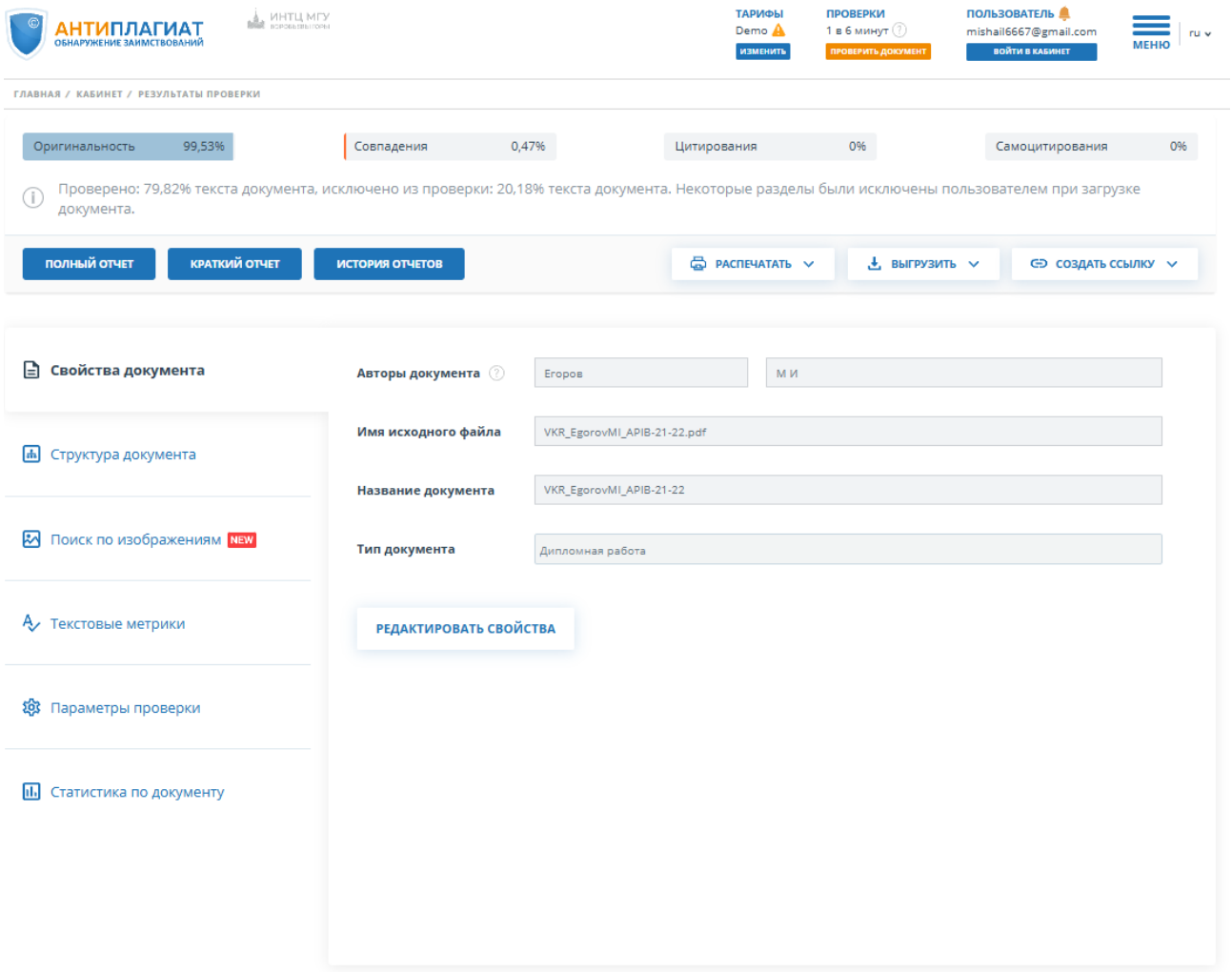
№ Варианта использования	Вариант использования 12
Название	Поддержка кроссплатформенности (Windows/Linux)
Автор	Егоров М.И.
Дата создания	4.01.2024
Действующее лицо	Разработчик движка
Описание	Обеспечение сборки и корректной работы движка на разных ОС
Предварительные условия	Наличие проекта и систем сборки
Выходные условия	Исполняемый файл под обе ОС
Нормальные условия	Конфигурация CMake, сборка под Windows и Linux
Альтернативное направление	Ошибки сборки на одной из платформ
Приоритет	Средний
Частота использования	6 раз в месяц
Особые требования	Среда сборки и библиотеки под обе ОС
Замечания и вопросы	Желательна CI-интеграция

Таблица Б.13 – Описание варианта использования «Ведение документации и примеров использования»

№ Варианта использования	Вариант использования 13
Название	Ведение документации и примеров использования
Автор	Егоров М.И.
Дата создания	4.01.2024
Действующее лицо	Разработчик движка
Описание	Создание примеров, обучающих материалов и описания API
Предварительные условия	Реализованы основные модули движка
Выходные условия	Пользователь может разобраться в архитектуре
Нормальные условия	Написание README, туториалов, комментирование кода
Альтернативное направление	Устаревшие или неполные описания
Приоритет	Средний
Частота использования	4 раза в месяц
Особые требования	Ясность и полнота материалов
Замечания и вопросы	Нет

Приложение В

Справка о проверке на антиплагиат



ПРИЛОЖЕНИЕ Г.

Экономические расчеты

Таблица Г.
1 - Расходы на оборудование

Наименование оборудования	Кол-во единиц	Стоимость, руб./ед.	Общая стоимость, руб.	Срок полезного использования, мес.	Период ввода, мес.	Суммарная установочная мощность, Квт/ч.	Характеристика
Этап разработки							
1. Оборудование, в т.ч.							
1.1 ПК	2,00	75000,00	150000,00	60,00	(ранее введен, срок экспл. 24 мес.)	0,30	Модель процессора - Ryzen 9 5950x Материнская плата - MSI MPG AM4 Оперативная память - DDR 4 32 GB Видеокарта - RTX 3060 Ti
Этап эксплуатации							
1. Оборудование, используемое системой, в т. ч.							
1.1 ПК	10,00	25000,00	250000,00	60,00	(ранее введен, срок экспл. 24 мес.)	1,50	Модель процессора - Ryzen 3 3500 Материнская плата - MSI MPG AM4 Оперативная память - DDR 4 16 GB Видеокарта - RTX 3050
2. Общесистемное оборудование, в т. ч.							
2.1 Роутер	1,00	2200,00	2200,00	60,00		0,02	WiFi роутер Kenetic Viva

Таблица Г.2 - Расчет амортизационных отчислений оборудования

Наименование оборудования	Кол-во единиц	Стоимость, руб./ед.	Срок полезного использования	Амортизация, руб.	
				месяц	год
Этап разработки					
1. Оборудование, в т.ч.					
1.1.ПК	2,00	75000,00	60,00	2500,00	30000,00
Этап эксплуатации					
1. Оборудование, используемое системой, в т. ч.					
1.1 ПК	10,00	25000,00	60,00	4166,67	50000,00
2. Общесистемное оборудование, в т. ч.					
2.1 Роутер	1,00	2200,00	60,00	36,67	440,00

Таблица Г.3 - Расходы на эксплуатацию оборудования

Наименование расходов	Этап разработки	Этап эксплуатации	
		Оборудование, используемое системой	Общесистемное оборудование
1. Стоимость расходных материалов, руб./мес.	0,00	0,00	0,00
2. Коэфициент использования электроустановок	0,90	0,90	0,90
3. Суммарная установочная мощность, Квт	0,30	1,50	0,02
4. Трудоемкость выполнения работы системой, ч. в мес.	132,00	6,00	6,00
5. Цена одного Квт/ч, руб.	7,20	7,20	7,20
Итого:	256,61	58,32	0,58

Таблица Г.4 - Расходы на программное обеспечение

Наименование ПО	Стоимость ПО, руб.	Кол-во, ед.	Общая стоимость, руб.	Период ввода в эксплуатацию	Срок использования, мес.
Этап разработки					
1. ПО, в т.ч.					
1.4 RenderDoc	0,00	0,00	0,00	0,00	60,00
1.5 MS Visual Studio 2022 Community edition	0,00	0,00	0,00	0,00	60,00
Этап эксплуатации					
1 ПО, используемое системой, в т.ч					
-	-	-	-	-	-

Таблица Г.5 - Расчет списания стоимости ПО

Наименование ПО	Стоимость, руб./ед.	Кол-во единиц	Списание стоимости ПО, руб.		Срок использования, мес.
			месяц	год	
Этап разработки					
1. ПО, в т.ч.					
1.4 RenderDoc	0,00	2,00	0,00	0,00	60,00
1.5 MS Visual Studio 2022 Community edition	0,00	2,00	0,00	0,00	60,00

Таблица Г.6 - Расчет списания стоимости ПО

Наименование ПО	Арендные платежи, руб.		Договор, арендодатель
	месяц	год	
Этап разработки			
1. ПО, в т.ч.			
Этап эксплуатации			
1. ПО, используемое системой, в т. ч.	-	-	-
2. Общесистемное ПО, в т. ч.	-	-	-

Таблица Г.7 – Продолжительность выполнения работ (Согласно диаграмме Ганта)

Этап жизненного цикла - исполнительное лицо	Вид работ	Продолжительность, дней	Продолжительность, час/мес.
1. Этап разработки - Программист	Проектирование ядра движка	5,00	33,00
	Проектирование модулей движка	6,00	39,60
	Проектирование системы рендеринга	4,00	26,40
	Разработка ядра движка	4,00	26,40
	Разработка модулей движка	6,00	39,60
	Разработка системы рендера	7,00	46,20
	Тестирование ядра движка	3,00	19,80
	Тестирование системы рендера	3,00	19,80
	Тестирование модулей движка	3,00	19,80
	Оценка качества рендера	2,00	13,20
Итого:		43,00	283,80
2. Этап эксплуатации			
2.1 Работа с системой			
Преподаватель	Обучение студентов с использованием игрового движка	1,00	8,00
Программист	Сопровождение системы	На всем ЖЦ проекта	8,00
2.2 Общесистемные расходы			
2.2.1 Общесистемное оборудование		1,00	8,00
Роутер		1,00	8,00

Таблица Г.8 - Штатное расписание

Этап ЖЦ проекта	Должность	Кол-во штатных единиц	Оклад, руб./мес.	Часовая тарифная ставка
Этап разработки				
Проектирование ядра движка	Программист	1,00	24816,00	188,00
Проектирование модулей движка				
Проектирование системы рендеринга				
Разработка ядра движка				
Разработка модулей движка				
Разработка системы рендера				
Тестирование ядра движка				
Тестирование системы рендера				
Тестирование модулей движка				
Оценка качества рендера				
Разработка проекта с использованием игрового движка				
Оценка качества рендера				
Экспорт в игровой движок				
Итого:				
Этап эксплуатации				
1.Работа с системой				
Эксплутация	Преподаватель	10,00	30000,00	227,27
Сопровождение	Программист	1,00	24816,00	188,00
2. Общесистемный персонал				
-	-	-	-	-

Таблица Г.10 – Расчет затрат на этапе разработки

Должность	Количество штатных единиц, чел	Оклад, руб/мес	Часовая тарифная ставка, руб/час	Трудоемкость, час/мес	Расходы на оплату труда, руб	Страховые взносы в социальные фонды (30,2%)	Итого РОТ, руб/мес
Этап разработки (1 месяц)							
Программист	1,00	24816,00	188,00	132,00	24816,00	7494,43	32310,43
Итого 1 месяц:				132,00	24816,00	7494,43	32310,43
Этап разработки (2 месяц)							
Программист	1,00	24816,00	188,00	132,00	24816,00	7494,43	32310,43
Итого 2 месяц:				132,00	24816,00	7494,43	32310,43
Этап разработки (3 месяц)							
Программист	1,00	3741,20	188,00	19,90	3741,20	1129,84	4871,04
Итого 3 месяц:				19,90	3741,20	1129,84	4871,04
Итого:					53373,20	16118,71	69491,91
Этап эксплуатации (помесячно)							
1. Персонал, работающий с системой, в т.ч.							
Преподаватель	10,00	1818,18	227,27	8,00	18181,82	5490,91	23672,73
Программист	1,00	1504,00	188,00	8,00	1504,00	454,21	1958,21
2. Общисистемный персонал, в т.ч.							
Итого:							25630,94

Таблица Г.11 – Расчет расходов на этапе эксплуатации - Расчет условно-прямых расходов

Наименование	Всего	в т.ч. по периодам эксплуатации системы										2 –ой год	3-ий год
		Итого за 1 год	4 мес.	5 мес.	6 мес.	7 мес.	8 мес.	9 мес.	10 мес.	11 мес.	12 мес.		
Условно-прямые расходы, руб.	985245,42	268703,30	29855,92	29855,92	29855,92	29855,92	29855,92	29855,92	29855,92	29855,92	29855,92	358271,06	358271,06
1. Расходы на оборудование, руб.	139424,56	38024,88	4224,99	4224,99	4224,99	4224,99	4224,99	4224,99	4224,99	4224,99	4224,99	50699,84	50699,84
1.1. Расходы на закупку оборудования, руб.	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
1.2. Сумма амортизации капитальных вложений, руб.	137500,00	37500,00	4166,67	4166,67	4166,67	4166,67	4166,67	4166,67	4166,67	4166,67	4166,67	50000,00	50000,00
1.3. Расходы на эксплуатацию оборудования, руб.	1924,56	524,88	58,32	58,32	58,32	58,32	58,32	58,32	58,32	58,32	58,32	699,84	699,84
2. Расходы на программное обеспечение (ПО), руб.	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
3.Административные расходы, руб.	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
3.1. Расходы на оплату труда пользователей системы, руб.	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
3.2 Расходы на операции ИС, руб.	845820,86	230678,42	25630,94	25630,94	25630,94	25630,94	25630,94	25630,94	25630,94	25630,94	25630,94	307571,22	307571,22
4. Расходы на связь и коммуникации, руб.	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00

Таблица Г.12 – Расчет расходов на этапе эксплуатации -Расчет условно-косвенных расходов

Наименование	Всего	Всего (9 месяцев)	1-ый год									2-ой год	3-ий год
			4 мес	5 мес	6 мес	7 мес	8 мес	9 мес	10 мес	11 мес	12 мес		
Условно-косвенные расходы по подсистеме (системе) с учетом коэффициента участия(0,001), руб.	100,23	27,34	3,04	3,04	3,04	3,04	3,04	3,04	3,04	3,04	3,04	36,45	36,45
Условно-косвенные расходы по подсистеме (системе), руб.	100229,25	27335,25	3037,25	3037,25	3037,25	3037,25	3037,25	3037,25	3037,25	3037,25	3037,25	36447,00	36447,00
1. Косвенные расходы на общесистемное оборудование, руб.	1229,25	335,25	37,25	37,25	37,25	37,25	37,25	37,25	37,25	37,25	37,25	447,00	447,00
1.1. Сумма амортизации капитальных вложений в общесистемное оборудование, руб.	1210,00	330,00	36,67	36,67	36,67	36,67	36,67	36,67	36,67	36,67	36,67	440,00	440,00
1.2. Расходы на эксплуатацию общесистемного оборудования, руб	19,25	5,25	0,58	0,58	0,58	0,58	0,58	0,58	0,58	0,58	0,58	7,00	7,00
2. Косвенные расходы на общесистемное программное обеспечение (ПО), руб.	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
3. Косвенные административные расходы, руб.	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
4. Косвенные расходы на операции ИС, руб.	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
5. Косвенные расходы на связь и коммуникации для рассматриваемой подсистемы (системы), руб.	99000,00	27000,00	3000,00	3000,00	3000,00	3000,00	3000,00	3000,00	3000,00	3000,00	3000,00	36000,00	36000,00

Таблица Г.13 – Расчет совокупной стоимости владения

Наименование	Всего	1-ый год													2-ой год	3-ий год
		Всего(1 год)	1 мес	2 мес	3 мес	4 мес	5 мес	6 мес	7 мес	8 мес	9 мес	10 мес	11 мес	12 мес		
Общая сумма расходов (ССВ)	1062889,66	346274,64	35067,04	35067,04	7409,92	29858,96	29858,96	29858,96	29858,96	29858,96	29858,96	29858,96	29858,96	29858,96	358307,51	358307,51
1.Расходы на этапе разработки	77544,00	77544,00	35067,04	35067,04	7409,92	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
1.1 Расходы на оборудование, руб.	8052,10	8052,10	2756,61	2756,61	2538,88	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
1.2 Расходы на программное обеспечение , руб.	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
1.3 Административные расходы, руб.	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
1.4 Расходы на операции ИС, руб.	69491,91	69491,91	32310,43	32310,43	4871,04	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
1.5 Расходы на связь и коммуникации, руб.	1800,00	1800,00	600,00	600,00	600,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
2.Расходы на этапе эксплуатации	985345,65	268730,63	0,00	0,00	0,00	29858,96	29858,96	29858,96	29858,96	29858,96	29858,96	29858,96	29858,96	29858,96	358307,51	358307,51
2.1.Условно-прямые расходы	985245,42	268703,30	0,00	0,00	0,00	29855,92	29855,92	29855,92	29855,92	29855,92	29855,92	29855,92	29855,92	29855,92	358271,06	358271,06
2.2.Условно-косвенные расходы	100,23	27,34	0,00	0,00	0,00	3,04	3,04	3,04	3,04	3,04	3,04	3,04	3,04	3,04	36,45	36,45

Таблица Г.14 – Расчёт трудоёмкости работ преподавателя

	Операция	Трудоемкость, час/мес		
		До внедрения продукта	После внедрения продукта	Отклонение
1 сотрудник	Обучение студентов	16,00	8,00	8,00
10 сотрудников		160,00	80,00	80,00

Таблица Г.15 - Эгод (прям)

Отклонение 10 сотрудников	Часов в год на обучение студентов(4 занятия в месяц по 2 часа)	Тсм	Эгод	Эмес
80,00	96,00	227,27	218181,82	18181,82

Таблица Г.15.1 косвенные затраты на 1 преподавателя (мес)

Статья затрат	Оценочная стоимост, руб./мес
Командировочные расходы	1700,00
Печатные материалы	300,00
Контроль и проверка заданий вручную	1000,00
Потери времени на повторяющиеся вводные	500,00
Участие в промежуточной аттестации	700,00
Итог:	4200,00

Таблица Г.15.2 – Эгод (косв)

Расходы на обучение студентов	До внедрения продукта, руб	После внедрения продукта, руб	Отклонение
1 сотрудник	4200,00	2100,00	-2100,00
10 сотрудников	42000,00	21000,00	-21000,00
Годовая экономия на обучении студентов, руб/год	-252000,00		

Таблица Г.16 – Эгод

Эгод (прям)	Эгод (косв)	Эгод
218181,82	252000,00	470181,82

Таблица Г.17 – Бюджет проекта

Наименование	Всего	1-ый год													2-ой год	3-ий год
		Всего (1 год)	1 мес	2 мес	3 мес	4 мес	5 мес	6 мес	7 мес	8 мес	9 мес	10 мес	11 мес	12 мес		
1. Доходы, в т. ч.	1293000	352636	0	0	0	39182	39182	39182	39182	39182	39182	39182	39182	39182	470182	470182
1.1 Годовая экономия (прямые факторы)	600000	163636	0	0	0	18182	18182	18182	18182	18182	18182	18182	18182	18182	218182	218182
1.2 Годовая экономия (косвенные факторы)	693000	189000	0	0	0	21000	21000	21000	21000	21000	21000	21000	21000	21000	252000	252000
2. Расходы, в т. ч.	1062890	346275	35067	35067	7410	29859	29859	29859	29859	29859	29859	29859	29859	29859	358308	358308
2.1 Расходы, связанные с разработкой ИТ проекта	77544	77544	35067	35067	7410	0	0	0	0	0	0	0	0	0	0	0
2.2 Эксплуатационные расходы	985346	268731	0	0	0	29859	29859	29859	29859	29859	29859	29859	29859	29859	358308	358308
3. Профицит (+) или дефицит (-) бюджета проекта	230110	6362	-35067	-35067	-7410	9323	9323	9323	9323	9323	9323	9323	9323	9323	111874	111874

Таблица Г.18.1 – Денежные потоки

Наименование	Всего	1-ый год													2-ой год	3-ий год
		Всего(1 год)	1 мес	2 мес	3 мес	4 мес	5 мес	6 мес	7 мес	8 мес	9 мес	10 мес	11 мес	12 мес		
Притоки (Доходы), руб.	1293000	352636	0	0	0	39182	39182	39182	39182	39182	39182	39182	39182	39182	470182	470182
Оттоки (расходы), руб.	1062890	346275	35067	35067	7410	29859	29859	29859	29859	29859	29859	29859	29859	29859	358308	358308
Чистый денежный поток	230110	6362	-35067	-35067	-7410	9323	9323	9323	9323	9323	9323	9323	9323	9323	111874	111874
Чистый денежный поток нарастающем итогом, руб.	230110	6362	-35067	-70134	-77544	-68221	-58898	-49575	-40253	-30930	-21607	-12284	-2961	6362	118236	230110

Таблица Г.18.2 – Денежные потоки

Наименование	Всего	1-ый год												2-ой год	3-ий год
		Всего(1 год)	1 мес	2 мес	3 мес	4 мес	5 мес	6 мес	7 мес	8 мес	9 мес	10 мес	11 мес	12 мес	
Коэффициент дисконтирования 1 / (1+r) ^ t, r = 25%	-	1													0,8	0,64
Дисконтированные денежные потоки, руб. (NPV)	167461	6362													89499	71600
Дисконтированые денежные потоки нарастающем итогом, руб. (NPV)	269684	6362	-35067	-70134	-77544	-68221	-58898	-49575	-40253	-30930	-21607	-12284	-2961	6362	95861	167461

Таблица 19 – показатели эффективности ИТ-проектов

Наименование	Единицы измерения	Значение
Чистая приведенная стоимость (NPV)	руб.	269684
Внутренняя норма доходности (IRR)	%	94%
Рентабельность (PI)	индекс	1,21
Дисконтированный срок окупаемости (DPP)	лет	0.94

Таблица 11 – Расчет расходов на этапе эксплуатации - Расчет условно-прямых расходов

Наименование	Всего	в т.ч. по периодам эксплуатации системы										2 –ой год	3-ий год
		Итого за 1 год	4 мес.	5 мес.	6 мес.	7 мес.	8 мес.	9 мес.	10 мес.	11 мес.	12 мес.		
Условно-прямые расходы, руб.	985245,42	268703,30	29855,92	29855,92	29855,92	29855,92	29855,92	29855,92	29855,92	29855,92	29855,92	358271,06	358271,06
1. Расходы на оборудование, руб.	139424,56	38024,88	4224,99	4224,99	4224,99	4224,99	4224,99	4224,99	4224,99	4224,99	4224,99	50699,84	50699,84
1.1. Расходы на закупку оборудования, руб.	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
1.2. Сумма амортизации капитальных вложений, руб.	137500,00	37500,00	4166,67	4166,67	4166,67	4166,67	4166,67	4166,67	4166,67	4166,67	4166,67	50000,00	50000,00
1.3. Расходы на эксплуатацию оборудования, руб.	1924,56	524,88	58,32	58,32	58,32	58,32	58,32	58,32	58,32	58,32	58,32	699,84	699,84
2. Расходы на программное обеспечение (ПО), руб.	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
3. Административные расходы, руб.	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
3.1. Расходы на оплату труда пользователей системы, руб.	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
3.2 Расходы на операции ИС, руб.	845820,86	230678,42	25630,94	25630,94	25630,94	25630,94	25630,94	25630,94	25630,94	25630,94	25630,94	307571,22	307571,22
4. Расходы на связь и коммуникации, руб.	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00

Таблица 12 – Расчет расходов на этапе эксплуатации -Расчет условно-косвенных расходов

Наименование	Всего	Всего (9 месяцев)	1-ый год									2-ой год	3-ий год
			4 мес	5 мес	6 мес	7 мес	8 мес	9 мес	10 мес	11 мес	12 мес		
Условно-косвенные расходы по подсистеме (системе) с учетом коэффициента участия(0,001), руб.	100,23	27,34	3,04	3,04	3,04	3,04	3,04	3,04	3,04	3,04	3,04	36,45	36,45
Условно-косвенные расходы по подсистеме (системе), руб.	100229,25	27335,25	3037,25	3037,25	3037,25	3037,25	3037,25	3037,25	3037,25	3037,25	3037,25	36447,00	36447,00
1. Косвенные расходы на общесистемное оборудование, руб.	1229,25	335,25	37,25	37,25	37,25	37,25	37,25	37,25	37,25	37,25	37,25	447,00	447,00
1.1. Сумма амортизации капитальных вложений в общесистемное оборудование, руб.	1210,00	330,00	36,67	36,67	36,67	36,67	36,67	36,67	36,67	36,67	36,67	440,00	440,00
1.2. Расходы на эксплуатацию общесистемного оборудования, руб	19,25	5,25	0,58	0,58	0,58	0,58	0,58	0,58	0,58	0,58	0,58	7,00	7,00
2. Косвенные расходы на общесистемное программное обеспечение (ПО), руб.	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
3. Косвенные административные расходы, руб.	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
4. Косвенные расходы на операции ИС, руб.	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
5. Косвенные расходы на связь и коммуникации для рассматриваемой подсистемы (системы), руб.	99000,00	27000,00	3000,00	3000,00	3000,00	3000,00	3000,00	3000,00	3000,00	3000,00	3000,00	36000,00	36000,00

Таблица 13 – Расчет совокупной стоимости владения

Наименование	Всего	1-ый год													2-ой год	3-ий год
		Всего(1 год)	1 мес	2 мес	3 мес	4 мес	5 мес	6 мес	7 мес	8 мес	9 мес	10 мес	11 мес	12 мес		
Общая сумма расходов (ССВ)	1062889,66	346274,64	35067,04	35067,04	7409,92	29858,96	29858,96	29858,96	29858,96	29858,96	29858,96	29858,96	29858,96	29858,96	358307,51	358307,51
1.Расходы на этапе разработки	77544,00	77544,00	35067,04	35067,04	7409,92	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
1.1 Расходы на оборудование, руб.	8052,10	8052,10	2756,61	2756,61	2538,88	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
1.2 Расходы на программное обеспечение , руб.	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
1.3 Административные расходы, руб.	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
1.4 Расходы на операции ИС, руб.	69491,91	69491,91	32310,43	32310,43	4871,04	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
1.5 Расходы на связь и коммуникации, руб.	1800,00	1800,00	600,00	600,00	600,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
2.Расходы на этапе эксплуатации	985345,65	268730,63	0,00	0,00	0,00	29858,96	29858,96	29858,96	29858,96	29858,96	29858,96	29858,96	29858,96	29858,96	358307,51	358307,51
2.1 Условно-прямые расходы	985245,42	268703,30	0,00	0,00	0,00	29855,92	29855,92	29855,92	29855,92	29855,92	29855,92	29855,92	29855,92	29855,92	358271,06	358271,06
2.2 Условно-косвенные расходы	100,23	27,34	0,00	0,00	0,00	3,04	3,04	3,04	3,04	3,04	3,04	3,04	3,04	3,04	36,45	36,45

Таблица 17 – Бюджет проекта

Наименование	Всего	1-ый год													2-ой год	3-ий год
		Всего(1 год)	1 мес	2 мес	3 мес	4 мес	5 мес	6 мес	7 мес	8 мес	9 мес	10 мес	11 мес	12 мес		
1. Доходы, в т. ч.	1293000	352636	0	0	0	39182	39182	39182	39182	39182	39182	39182	39182	39182	470182	470182
1.1 Годовая экономия (прямые факторы)	600000	163636	0	0	0	18182	18182	18182	18182	18182	18182	18182	18182	18182	218182	218182
1.2 Годовая экономия (косвенные факторы)	693000	189000	0	0	0	21000	21000	21000	21000	21000	21000	21000	21000	21000	252000	252000
2. Расходы, в т. ч.	1062890	346275	35067	35067	7410	29859	29859	29859	29859	29859	29859	29859	29859	29859	358308	358308
2.1 Расходы, связанные с разработкой ИТ проекта	77544	77544	35067	35067	7410	0	0	0	0	0	0	0	0	0	0	0
2.2 Эксплуатационные расходы	985346	268731	0	0	0	29859	29859	29859	29859	29859	29859	29859	29859	29859	358308	358308
3. Профицит (+) или дефицит (-) бюджета проекта	230110	6362	-35067	-35067	-7410	9323	9323	9323	9323	9323	9323	9323	9323	9323	111874	111874

Таблица 18 – Денежные потоки

Наименование	Всего	1-ый год													2-ой год	3-ий год
		Всего(1 год)	1 мес	2 мес	3 мес	4 мес	5 мес	6 мес	7 мес	8 мес	9 мес	10 мес	11 мес	12 мес		
Притоки (Доходы), руб.	1293000	352636	0	0	0	39182	39182	39182	39182	39182	39182	39182	39182	39182	470182	470182
Оттоки (расходы), руб.	1062890	346275	35067	35067	7410	29859	29859	29859	29859	29859	29859	29859	29859	29859	358308	358308
Чистый денежный поток	230110	6362	-35067	-35067	-7410	9323	9323	9323	9323	9323	9323	9323	9323	9323	111874	111874
Чистый денежный поток нарастающем итогом, руб.	230110	6362	-35067	-70134	-77544	-68221	-58898	-49575	-40253	-30930	-21607	-12284	-2961	6362	118236	230110
Коэффициент дисконтирования 1 / (1+r) ^ t, r = 25%	-	1													0,8	0,64
Дисконтированные денежные потоки, руб. (NPV)	167461	6362													89499	71600
Дисконтированые денежные потоки нарастающем итогом, руб. (NPV)	269684	6362	-35067	-70134	-77544	-68221	-58898	-49575	-40253	-30930	-21607	-12284	-2961	6362	95861	167461