

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

ОТЧЕТ

по учебной – научно-исследовательской работе

Исполнитель: Егоров Михаил Игоревич студент 4 курса, группы АПИб-21-22

Руководитель Курзаева Любовь Викторовна, доц. каф. БИиИТ, к.п.н., доц
практики:

Отчет «____» ____ 20____ г. с оценкой _____
защищен
(оценка) (подпись)

Магнитогорск, 2025

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

РАБОЧИЙ ПЛАН-ГРАФИК

учебной – научно-исследовательской работы

в период с 02.09.2024 по 14.01.2025

Обучающемуся Егорову Михаилу Игоревичу

группы АПИб-21-22

<i>№</i>	<i>Этапы практики по выполнению программы практики и индивидуального задания</i>	<i>Срок исполнения</i>
1	Организационно-подготовительный этап	2.09.2024 - 9.09.2024
2	Основной этап	12.09.2024 - 22.12.2024
3	Отчетный этап	23.12.2024 - 14.01.2025

Руководитель практики от МГТУ им. Г.И. Носова

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

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

ИНДИВИДУАЛЬНОЕ ЗАДАНИЕ
на учебную – научно-исследовательскую работу

Обучающемуся Егорову Михаилу Игоревичу

группы АПИБ-21-22

09.03.03 Прикладная информатика (разработка компьютерных игр и приложений виртуальной/дополненной реальности)

1. Период практики: с 02.09.2024 по 14.01.2025
2. Место прохождения практики
Кафедра БИиИТ ФГБОУ ВО «МГТУ им. Г.И. Носова»

№ п/п	Содержание индивидуального задания (перечень задач, подлежащих выполнению)
1	Организационно-подготовительный этап
1.1	Участие в установочном собрании по организации практики. Получение индивидуального задания.
1.2	Вводный инструктаж представителя кафедры БИиИТ обучающимся по Правилам ТБ, производственной и противопожарной безопасности
2	Основной этап
2.1	Пройти онлайн-курс. Академическое русское письмо https://openedu.ru/course/spbu/ACADRU/?session=spring_2021
2.2	Выполнить подборку базы апробации результатов исследования (конференции для участия, журналы для публикаций)
2.3	Подготовить к публикации обзорную статью по теме исследования, ориентируясь на теоретические аспекты проблемы исследования
2.4	Провести анализ современного состояния проблемы разработки игровых движков.
2.5	Осуществить постановку задачи на разработку игрового движка «LampyEngine» с использованием API Vulkan.
2.6	Провести сравнительный анализ и выбрать оптимальные средства разработки
3	Отчетный этап
3.1	Подготовка и защита отчета по практике

Руководитель практики
от МГТУ им. Г.И. Носова _____ / Курзаева Л.В. /

Обучающийся _____ / Егоров М.И. /

Дата выдачи 02.09.2024

Дневник прохождения практики

Студента Егорова Михаила Игоревича

Группы АПИБ-21-22

курса 4

Направления 09.03.03 Прикладная информатика (разработка компьютерных игр и приложений виртуальной/дополненной реальности)

Сроки практики: с 02.09.2024 по 14.01.2025 г.

Дата	Краткое содержание выполненной работы	Отметка о выполнении
1. Организационно-подготовительный этап		
02.09.2024	1.1 Участие в установочном собрании по организации практики. Получение индивидуального задания.	Выполнено
02.09.2024	1.2 Вводный инструктаж представителя кафедры БИиИТ обучающимся по Правилам ТБ, производственной и противопожарной безопасности	Выполнено
2. Основной этап		
12.09.2024 – 22.12.2024	2.1 Пройден онлайн-курс. Академическое русское письмо https://openedu.ru/course/spbu/ACADRU/?session=spring_2021	Выполнено
13.09.2024 – 27.09.2024	2.2 Выполнена подборка базы апробации результатов исследования (конференции для участия, журналы для публикаций)	Выполнено
21.10.2024 – 01.12.2024	2.3 Подготовлена к публикации обзорная статья по теме исследования, которая ориентирована на теоретические аспекты проблемы исследования	Выполнено
30.09.2024 – 18.10.2024	2.4 Проведен анализ современного состояния проблемы разработки игровых движков.	Выполнено
21.10.2024 – 18.11.2024	2.5 Осуществлена постановка задачи на разработку игрового движка «LampyEngine» с использованием API Vulkan.	Выполнено
19.11.2024 – 20.12.2024	2.6 Осуществлен выбор и обоснование методов и средств разработки игрового движка «LampyEngine».	Выполнено
3. Отчетный этап		
23.12.2024- 14.01.2025	Подготовка отчета по практике Отправка отчета на проверку Выступление на отчетной конференции	Выполнено

Руководитель практики
от МГТУ им. Г.И. Носова

_____ / Курзаева Л.В. /

СОДЕРЖАНИЕ

ВВЕДЕНИЕ	6
1 Анализ современного состояния проблемы разработки игровых движков....	8
2 Постановка задачи на разработку игрового движка «LampyEngine» с использованием API Vulkan	13
3 Выбор и обоснование методов и средств разработки игрового движка «LampyEngine»	17
ЗАКЛЮЧЕНИЕ	22
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ.....	24
ПРИЛОЖЕНИЯ	27
Приложение А Документ о вариантах использования	27
Приложение Б Обзорная статья по теме исследования	40
Приложение В Задание и содержание ВКР	49
Приложение Г Справка о проверке на антиплагиат.....	52

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 разработки 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 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 рефлексии — 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.

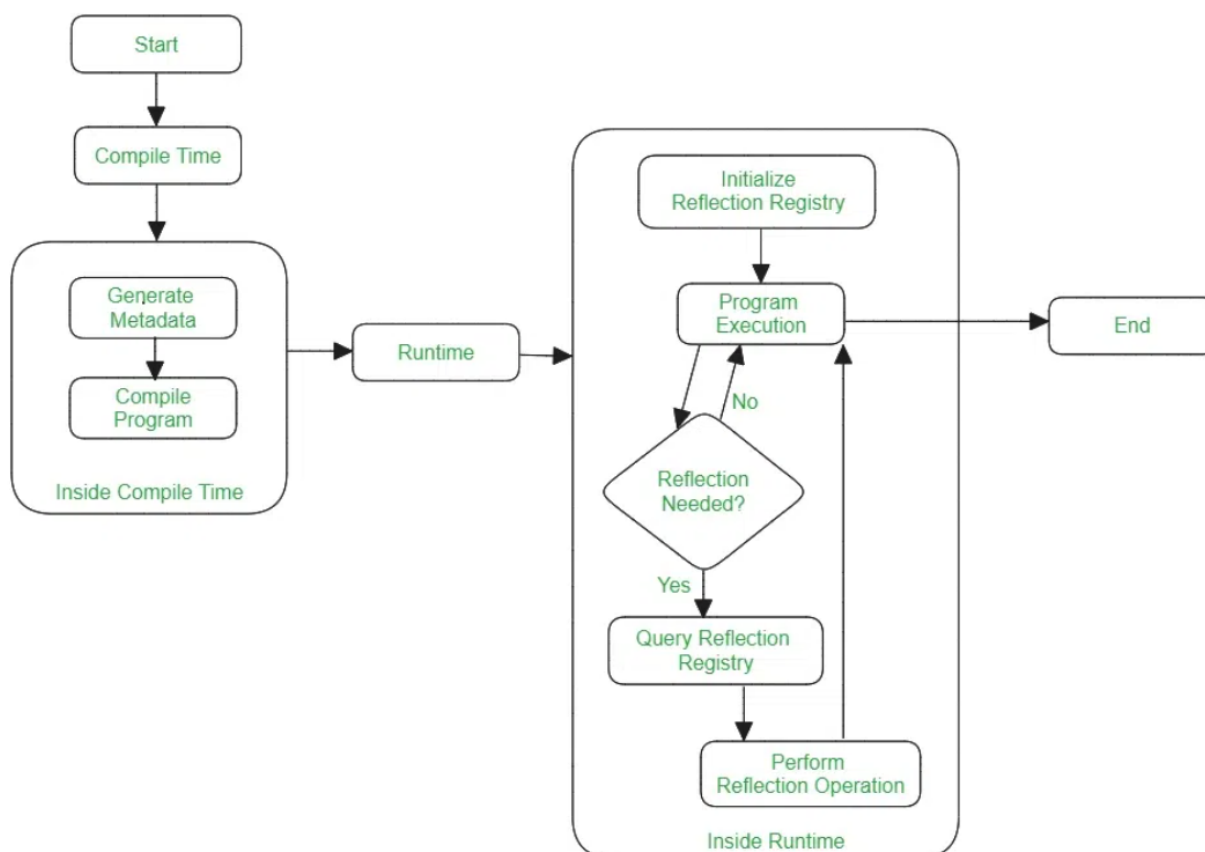


Рисунок 1 — Блок-схема работы рефлексии

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 каждого 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.	<ul style="list-style-type: none"> 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.	<ul style="list-style-type: none"> 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.	<ul style="list-style-type: none"> 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.

2 Постановка задачи на разработку игрового движка «LampyEngine» 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 прецедентов игрового движка.

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

3 Выбор и обоснование методов и средств разработки игрового движка «LampryEngine»

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 определиться 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++	<ul style="list-style-type: none"> • Имеет несравненную производительность и полный контроль над системой. • Несомненно, подходит для крупных игровых движков. 	<ul style="list-style-type: none"> • Довольно сложен в изучении. • Наклаываются риски утечек памяти.
Rust	<ul style="list-style-type: none"> • Безопасное управление памятью, современный синтаксис и высокая производительность. 	<ul style="list-style-type: none"> • В наличии мало готовых библиотек. • Нет единого «де-факто» языкового стандарта.
C#	<ul style="list-style-type: none"> • Прост в освоении • Быстрая разработка • Подходит для кроссплатформенных решений. 	<ul style="list-style-type: none"> • Не подходит для низкоуровневой оптимизации. • Зависимость от среды исполнения (.NET CLR, Mono/IL2CPP).
Python	<ul style="list-style-type: none"> • Максимально простая для старта. • Быстрое прототипирование • Большое сообщество разработчиков 	<ul style="list-style-type: none"> • 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	Преимущества	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 инструмент 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 с++ стеков для разработки движков

Стек	Преимущества	Недостатки
WinAPI + DirectXMath + DirectX 11/12	<ul style="list-style-type: none"> • Высокая производительность • Полный контроль за ресурсами • Нативный для Windows 	<ul style="list-style-type: none"> • Поддержка исключительно Windows систем • Управление окнами и вводом через WinAPI
SDL + OpenGL	<ul style="list-style-type: none"> • Кроссплатформенность • Сочетает в себе единый доступ к управлению окнами и звуком 	<ul style="list-style-type: none"> • Ограниченные возможности по звуку • Не подходит для сложной графики

GLFW + OpenGL + GLM + OpenAL	<ul style="list-style-type: none"> • Прост в использовании • Отличен для обучения и небольших проектов 	<ul style="list-style-type: none"> • OpenGL устаревает, ограниченный доступ к низкоуровневым GPU-возможностям
GLFW + Vulkan + OpenGL + OpenAL	<ul style="list-style-type: none"> • Максимальная кроссплатформенность • Соответствие лучшим практикам 	<ul style="list-style-type: none"> • 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.

ЗАКЛЮЧЕНИЕ

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 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.

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 CАНРАG.
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, А. 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

Приложение А Документ о вариантах 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, tutorиалов, 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 Б

Обзорная статья по теме исследования

ПРОЕКТИРОВАНИЕ СИСТЕМЫ РЕНДЕРИГА ДЛЯ ИГРОВОГО ДВИЖКА «LAMPY ENGINE»

¹Егоров М.И.

¹ФГБОУ ВО «Магнитогорский государственный университет»

Цель работы – разработка универсальной, масштабируемой и расширяемой системы рендеринга для игрового движка с поддержкой различных графических API. В работе представлена архитектура подсистемы рендеринга, основанная на абстрактной прослойке, обеспечивающей единый интерфейс взаимодействия с графической системой. Такой подход позволяет отделить логику рендеринга от конкретной реализации и упростить поддержку различных технологий отображения. Рассматриваются ключевые компоненты архитектуры, включая обобщённые графические объекты, структуры данных для описания рендер-процессов и механизм управления отрисовкой сцены. Показан подход к организации рендеринга через последовательные этапы и обработку объектов с возможностью их группировки и оптимизации.

Ключевые слова: рендеринг, графическая архитектура, игровой движок, прослойка, графический программный интерфейс, визуализация.

Введение

Современные игровые движки требуют гибких и масштабируемых решений в области рендеринга для обеспечения совместимости с различными устройствами и поддержки новейших графических технологий. Использование нескольких графических API в рамках одного проекта позволяет достичь баланса между производительностью и универсальностью. При этом особую сложность представляет организация архитектуры, способной эффективно взаимодействовать как с высокоуровневыми, так и с низкоуровневыми графическими интерфейсами.

Разработка собственной системы рендеринга требует создания абстрактного слоя, способного скрыть различия между API и предоставить единый интерфейс для отрисовки объектов сцены. Такой подход позволит добиться повторного использования кода, упрощения логики рендеринга на уровне движка и повышения модульности всей системы.

В рамках проекта создания игрового движка «LampyEngine» была поставлена задача реализовать универсальную подсистему рендеринга с поддержкой нескольких API. В качестве основы архитектуры был выбран подход, аналогичный применяемому в ряде современных движков, ориентированный на разделение логики рендеринга и конкретной реализации. Настоящая работа посвящена описанию построенной архитектуры, принципов взаимодействия компонентов подсистемы и особенностей реализации одного из графических интерфейсов.

Актуальность исследования

В последние годы значительно возрос интерес к разработке кроссплатформенных мобильных приложений, способных оперативно взаимодействовать с внешними источниками данных в реальном времени, такими как финансовые сервисы и государственные информационные системы. Это обусловлено стремлением обеспечить высокую точность и актуальность предоставляемой информации при минимальных затратах на поддержку и сопровождение приложений на разных платформах.

Согласно публикациям в научных журналах, одной из наиболее востребованных областей применения таких приложений является мониторинг и отображение актуальных валютных курсов, предоставляемых центральными банками различных стран. В частности, использование официальных XML- и JSON-сервисов позволяет повысить надёжность и скорость получения данных, а также унифицировать процесс интеграции с приложениями различного назначения [1]. Применение современных платформ разработки, таких как Unity, обеспечивает значительное ускорение процесса создания интерфейсов и снижает издержки при переносе приложений между мобильными и десктопными системами [2].

Существенное внимание в исследованиях также уделяется проблемам парсинга данных из внешних источников и эффективной обработке получаемой информации. Современные методы обработки XML и JSON, реализуемые средствами языков программирования высокого уровня, таких как C#, позволяют упростить и ускорить процесс получения и отображения данных, что особенно актуально в контексте мобильных приложений, предъявляющих повышенные требования к производительности и скорости отклика [3].

Таким образом, разработка мобильного приложения «Биржа360fps», основанного на платформе Unity и официальном XML-сервисе Центрального банка Российской Федерации, является актуальной задачей. Она направлена на устранение существующих проблем оперативности и достоверности информации о курсах валют, выявленных в предыдущих исследованиях, и способствует дальнейшему развитию мобильных финансовых приложений.

Материал и методы исследования

В качестве основного объекта исследования в данной работе выступает подсистема рендеринга, разрабатываемая в рамках проекта игрового движка «LampyEngine». Целью исследования является создание универсальной и масштабируемой архитектуры рендеринга, способной работать с несколькими графическими API, в частности OpenGL и Vulkan. Для достижения поставленной цели были использованы как теоретические, так и практические методы проектирования, включая анализ существующих решений, построение абстракций, а также реализация и тестирование конкретных модулей рендеринга.

Материалы исследования включают в себя:

- Игровой движок «LampyEngine», спроектированный и разработанный с нуля для обеспечения гибкости архитектуры и поддержки модульного подключения компонентов рендеринга.
- Графические API Vulkan и OpenGL, выбранные как основные средства визуализации, что позволило провести сравнительный анализ подходов к реализации рендеринга на высоком и низком уровнях абстракции.
- Инструменты разработки: компилятор C++, система сборки CMake, система управления зависимостями Conan, отладочные и профилировочные средства, а также система контроля версий Git.

Методы исследования включают в себя следующие этапы:

1. Анализ архитектурных решений, реализованных в популярных игровых движках (в том числе Godot и Unreal Engine), с целью выявления подходов к абстрагированию графических интерфейсов и построению универсальных подсистем рендеринга.
2. Проектирование абстрактного слоя рендеринга, предполагающего создание общего интерфейса взаимодействия с подсистемой визуализации. Архитектура разработана на основе объектно-ориентированных принципов и разделена на высокоуровневые и низкоуровневые компоненты.
3. Моделирование ключевых компонентов, таких как текстуры, шейдеры, материалы, рендер-проходы и объекты отрисовки. Каждый из компонентов реализован в виде абстрактных классов с возможностью конкретизации под выбранный API.
4. Разработка прослойки IRenderer, реализующей общий контракт рендеринга. Далее эта прослойка расширяется в зависимости от выбранного API: OpenGLRenderer или VulkanRenderer. Такой подход обеспечивает единообразие обработки рендер-команд на верхнем уровне движка и упрощает переключение между графическими интерфейсами.
5. Организация управления рендер-процессами через реализацию структур данных RenderPassData и RenderPipelineData, обеспечивающих пакетную обработку объектов, настройку рендер-проходов и упрощённое добавление новых этапов визуализации.
6. Реализация и отладка прототипов, включающих последовательную отрисовку сцены с использованием системы батчей, индексированной отрисовки и управления состоянием GPU, а также проверка работоспособности архитектуры на тестовых сценах с различным числом объектов.

В совокупности применённые методы обеспечили реализацию гибкой и масштабируемой системы рендеринга, обладающей возможностью адаптации под различные графические интерфейсы и способной обеспечить высокую производительность визуализации в рамках движка «LampyEngine».

Результаты исследования

В рамках данного исследования была разработана система рендеринга для игрового движка «LampyEngine», поддерживающая работу с графическими API OpenGL и Vulkan. Основной целью было создание абстрактного слоя, обеспечивающего единый интерфейс для взаимодействия с различными графическими API.

Материалы исследования:

- Игровой движок «LampyEngine»: разработан с нуля для обеспечения гибкости и расширяемости системы рендеринга.
- Графические API: использованы OpenGL и Vulkan для реализации рендеринга, что позволило оценить их особенности и производительность.
- Инструменты разработки: применялись современные среды разработки и системы контроля версий для обеспечения эффективного процесса разработки.

Методы исследования:

1. Анализ существующих решений: проведен обзор существующих подходов к интеграции различных графических API в игровые движки, что позволило определить лучшие практики и возможные проблемы.
2. Проектирование архитектуры рендеринга: разработана модульная архитектура, разделяющая логику рендеринга и специфичные реализации для каждого API. Это обеспечило гибкость и возможность расширения системы.
3. Реализация абстрактного слоя: создан интерфейс, скрывающий различия между OpenGL и Vulkan, что упростило процесс разработки и тестирования.

Данные методы позволили создать эффективную и гибкую систему рендеринга, соответствующую целям исследования.

Результаты исследования

В контексте проекта «LampyEngine» 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.

В текущей реализации игрового движка было принято решение построить прослойку рендеринга (IRenderer на рисунке 1) 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.

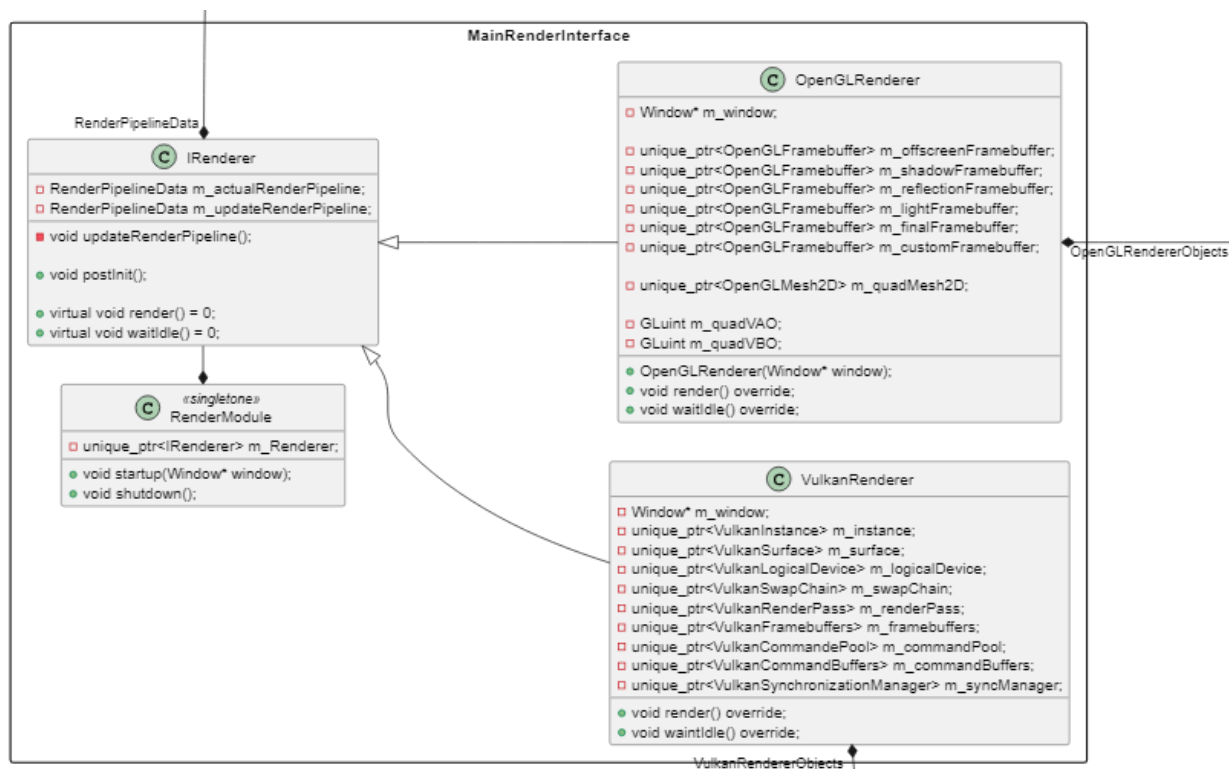


Рисунок 1 – Диаграмма классов верхнего уровня рендеринга.

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)

Для предметного понимания того, что генерирует прослойка, разберем диаграммы классов представленных на рисунках 2 и 3.

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.

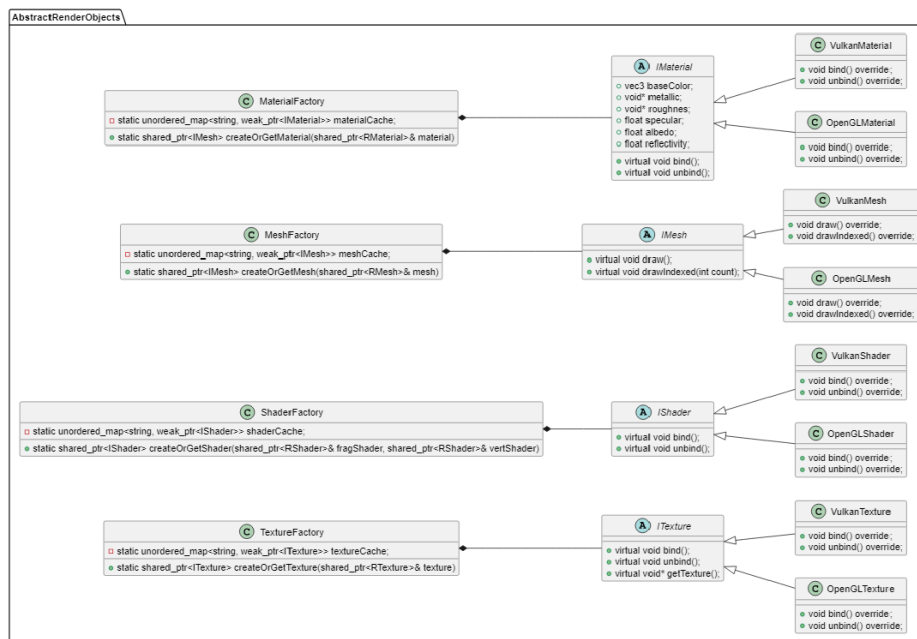


Рисунок 2 – Диаграмма классов обобщенных объектов рендеринга.

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.

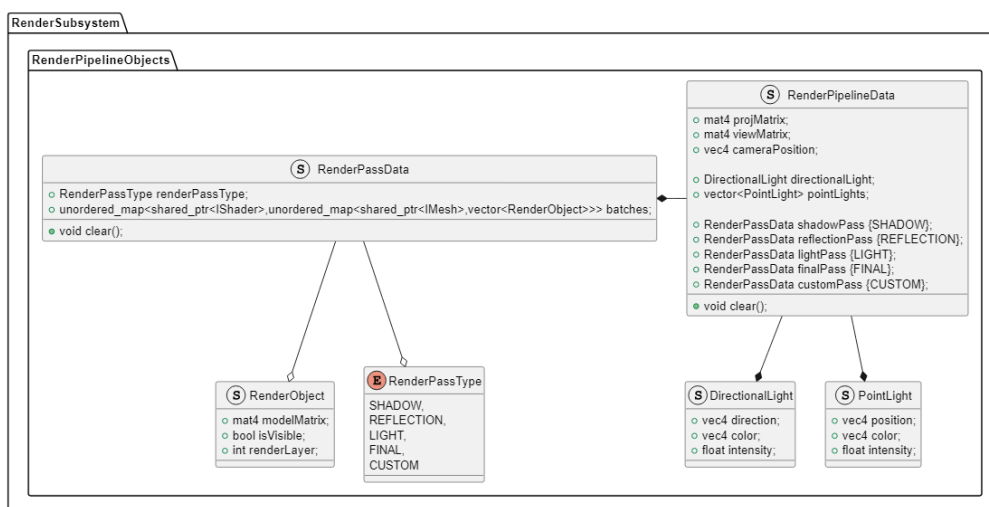


Рисунок 3 – Диаграмма классов данных рендеринга.

Теперь основываясь на информации, генерируемой на уровне прослойки и процессе ее подмена в момент изменения состояния мира, следует перейти к рассмотрению в конкретной реализации OpenGL. 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 4. В цикле итерируется хеш-таблица с ключом-шейдером и значением в виде меша. Меша в свою очередь относятся к структуре RenderObject содержащей 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.

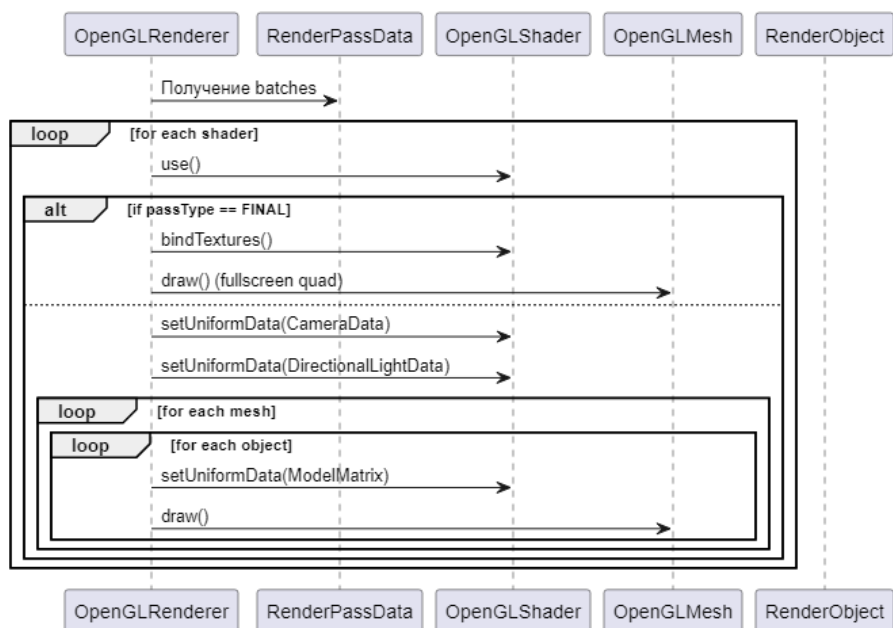


Рисунок 4 – Алгоритм исполнения команд отрисовки.

Упрощённая диаграмма последовательности рендеринга OpenGL

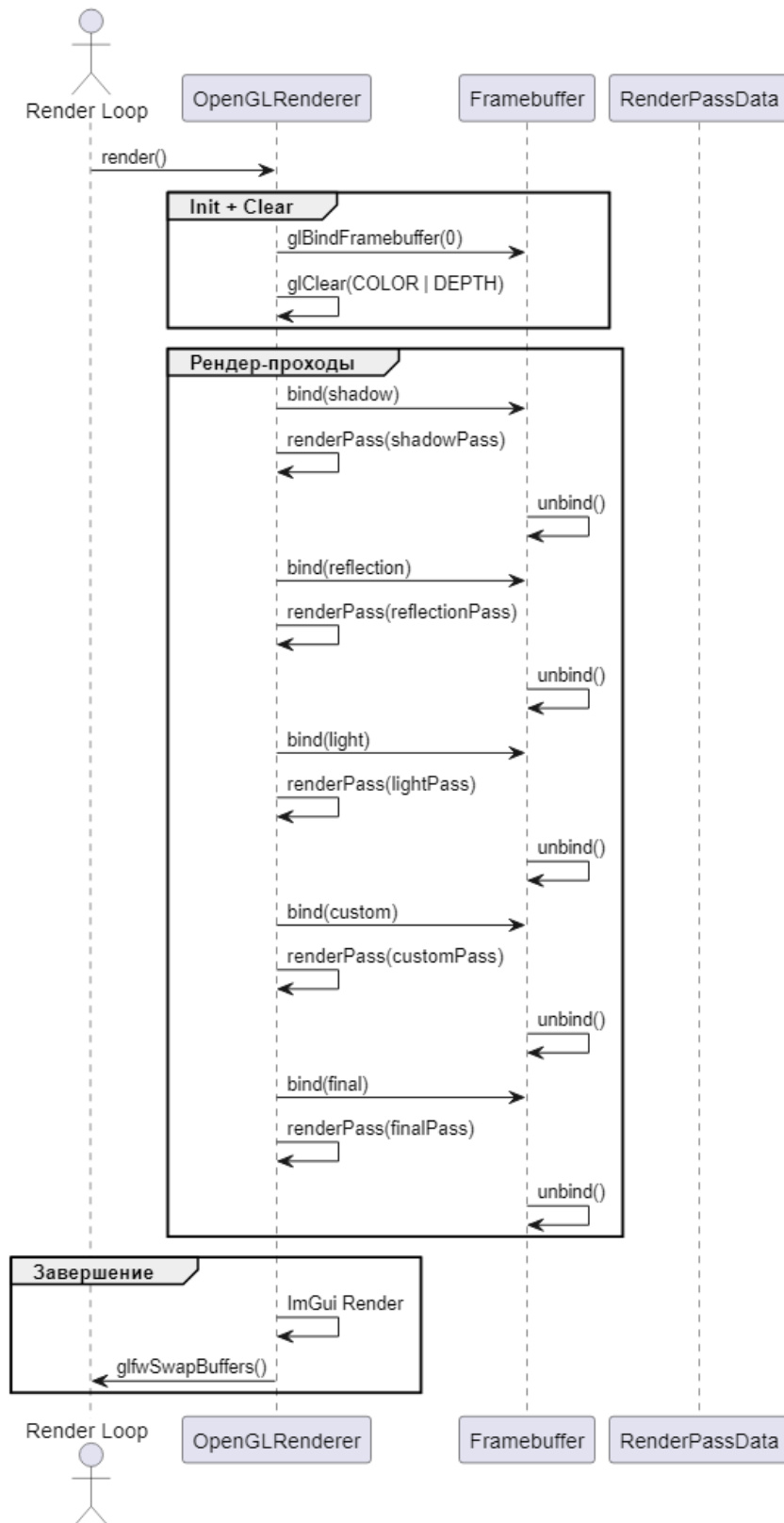


Рисунок 5 – Диаграмма последовательности рендера OpenGL.

Таким образом поддерживается архитектура на основе высокоуровневой прослойки IRenderer. OpenGL достаточно следовать инструкциям верхнего уровня и отрисовка будет соответствовать ожиданиям разработчиков.

Вывод

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.

Список использованных источников

1. Гуляев, Н. А. О структурах организации графических данных в прямой объёмной визуализации с регулируемыми параметрами рендеринга / Н. А. Гуляев, В. В. Селянкин // Информационные технологии, системный анализ и управление (ИТСАУ-2018) : сборник трудов XVI Всероссийской научной конференции молодых ученых, аспирантов и студентов: в 3 томах, Ростов-на-Дону - Таганрог, 05–07 декабря 2018 года / Южный федеральный университет. Том III. – Ростов-на-Дону - Таганрог: Южный федеральный университет, 2018. – С. 146-152. – EDN XSNWWI.
2. Федоров, П. А. Особенности разработки алгоритмов 3D-рендеринга для графических процессоров, использующих технологию CUDA / П. А. Федоров // Евразийский союз ученых. – 2015. – № 10-2(19). – С. 149-152. – EDN VBGYED.
3. 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”. – 2022. – No. 32. – P. 15-25. – EDN SANPAG.
4. 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 2016 года / С. В. Абламейко (гл. редактор). – Минск: Белорусский государственный университет, 2016. – С. 968-972. – EDN XWMVTR.
5. 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.

Приложение В

Задание и содержание 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»)

Кафедра бизнес-информатики и
информационных технологий

УТВЕРЖДАЮ:
Заведующий кафедрой
_____ Г.Н. Чусавитина

00.01.2025 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 № 00-00/000 от 00.01.2025 г.

Срок выполнения 01.06.2025 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
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. Спроектировать игровой движок
 - 4.1. Осуществить концептуальное проектирование
 - 4.2. Осуществить проектирование 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


СОДЕРЖАНИЕ

ВВЕДЕНИЕ	Ошибка! Закладка не определена.
1 ТЕОРЕТИЧЕСКИЕ ОСНОВЫ ПРОЕКТИРОВАНИЯ И РАЗРАБОТКИ ИГРОВЫХ ДВИЖКОВ.....	Ошибка! Закладка не определена.
1.1. Анализ современного состояния проблемы разработки игровых движков ...	Ошибка! Закладка не определена.
1.2. Постановка задачи на разработку игрового движка «LampyEngine» с использованием API Vulkan.....	Ошибка! Закладка не определена.
1.3. Выбор и обоснование методов и средств разработки игрового движка «LampyEngine»	Ошибка! Закладка не определена.
ВЫВОДЫ ПО РАЗДЕЛУ 1	Ошибка! Закладка не определена.
2 РАЗРАБОТКА ИГРОВОГО ДВИЖКА «LampyEngine»	Ошибка! Закладка не определена.
2.1. Разработка проектных решений для игрового движка «LampyEngine»	Ошибка! Закладка не определена.
2.2. Реализация и тестирование игрового движка «LampyEngine» .	Ошибка! Закладка не определена.
2.3. Экспериментальное тестирование игрового движка и оценка его эффективности.	Ошибка! Закладка не определена.
ВЫВОДЫ ПО РАЗДЕЛУ 2	Ошибка! Закладка не определена.
ЗАКЛЮЧЕНИЕ	Ошибка! Закладка не определена.
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ.....	Ошибка! Закладка не определена.
ПРИЛОЖЕНИЯ.....	Ошибка! Закладка не определена.
Приложение А. Календарный график ВКР.....	Ошибка! Закладка не определена.
Приложение Б. Скриншоты игрового движка	Ошибка! Закладка не определена.
Приложение В. Портфолио Егорова Михаила Игоревича, АПИб-21-22 за время обучения	Ошибка! Закладка не определена.
Приложение Г. Доклад на защиту	Ошибка! Закладка не определена.
Приложение Д. Справка о проверке на антиплагиат .	Ошибка! Закладка не определена.
Приложение Е. Согласие на размещение ВКР в электронной библиотеке	Ошибка! Закладка не определена.

Приложение Г

Справка о проверке на антиплагиат

**АНТИПЛАГИАТ**
ОБНАРУЖЕНИЕ ЗАИМСТВОВАНИЙ

ИИТЦ МГУ
КОРПОРАТИВНОСТЬ

ТАРИФЫ
Demo ⚠
ИЗМЕНИТЬ

ПРОВЕРКИ
1 в 6 минут ⚙
ПРОВЕРИТЬ ДОКУМЕНТ

ПОЛЬЗОВАТЕЛЬ
mishail6667@gmail.com
ВОЙТИ В КАБИНЕТ

МЕНЮ ru ▼

ГЛАВНАЯ / КАБИНЕТ / РЕЗУЛЬТАТЫ ПРОВЕРКИ

Оригинальность91,37%

Совпадения8,63%

Цитирования0%

Самоцитирования0%

Проверено: 79,51% текста документа, исключено из проверки: 20,49% текста документа. Некоторые разделы были исключены пользователем при загрузке документа.

ПОЛНЫЙ ОТЧЕТ

КРАТКИЙ ОТЧЕТ

ИСТОРИЯ ОТЧЕТОВ

РАСПЕЧАТАТЬ ▼

ВЫГРУЗИТЬ ▼

СОЗДАТЬ ССЫЛКУ ▼

Свойства документа

Структура документа

Поиск по изображениям **NEW**

Текстовые метрики

Параметры проверки

Статистика по документу

Авторы документа ⓘ

Имя исходного файла

Название документа

Тип документа

РЕДАКТИРОВАТЬ СВОЙСТВА

Егоров

Михаил Игоревич

ОТЧЕТ_НИР_ЕгоровМИ(АПИБ-21-22).pdf

ОТЧЕТ_НИР_ЕгоровМИ(АПИБ-21-22).

Дипломная работа