



Министерство науки и высшего образования Российской Федерации  
федеральное государственное бюджетное образовательное учреждение  
высшего образования  
«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ  
МОРДОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ  
ИМ. Н. П. ОГАРЁВА»  
(ФГБОУ ВО «МГУ им. Н.П. Огарёва»)

Факультет математики и информационных технологий

Кафедра фундаментальной информатики

УТВЕРЖДАЮ  
Зав. кафедрой  
канд. физ.-мат. наук, доц.  
 А. Г. Смольянов  
«14» 06 2024 г.

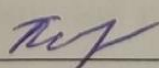
ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА БАКАЛАВРА  
по теме:  
ПРОГРАММНАЯ РЕАЛИЗАЦИЯ ДВИЖКА ДЛЯ ПОСТРОЕНИЯ  
3D-МОДЕЛЕЙ

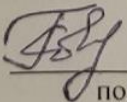
Автор бакалаврской работы  03.06.2024 К. В. Назаров  
подпись, дата

Обозначение бакалаврской работы БР-02069964-02.03.02-15-24

Направление подготовки 02.03.02 Фундаментальная информатика и  
информационные технологии

Профиль Информатика и компьютерные науки

Руководитель работы  
канд. физ.-мат. наук  03.06.2024 А. В. Попов  
подпись, дата

Нормоконтролёр  
канд. техн. наук, доц.  05.06.2024 С.В. Гарина  
подпись, дата

Саранск 2024

Министерство науки и высшего образования Российской Федерации  
федеральное государственное бюджетное образовательное учреждение  
высшего образования  
«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ  
МОРДОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ  
ИМ. Н. П. ОГАРЁВА»  
(ФГБОУ ВО «МГУ им. Н.П. Огарёва»)


Факультет математики и информационных технологий

Кафедра фундаментальной информатики

УТВЕРЖДАЮ

Зав. кафедрой

канд. физ.-мат. наук, доц.

 А. Г. Смольянов

«25» 12 2023 г.

ЗАДАНИЕ НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ  
БАКАЛАВРА

Студенту Назарову Кириллу Викторовичу

1 Тема Программная реализация движка для построения 3D-моделей

Утверждена приказом № 11140-с от 25.12.2023г.

2 Срок представления работы к защите 14.06.2024г.

3 Исходные данные для научного исследования: электронные ресурсы по теме исследования, существующие приложения

4 Содержание выпускной квалификационной работы

4.1 Обзор существующих движков

4.2 Методы представления трехмерной графики

4.3 Реализация движка

Продолжение на следующем листе

5 Приложения: коды программ.

Руководитель работы  
канд. физ.-мат. наук

Попов 25.12.2023 А. В. Попов  
подпись, дата

Задание принял к исполнению

Назаров 25.12.2023 К. В. Назаров  
подпись, дата

## РЕФЕРАТ

Выпускная квалификационная работа бакалавра 65 с., 21 источн., 35 рис., 5 прил.

ГРАФИКА, МОДЕЛИРОВАНИЕ, ДВИЖОК, 3D, PYTHON, ИНСТРУМЕНТ, ПРОГРАММА, РЕАЛИЗАЦИЯ, ПОСТРОЕНИЕ, РАЗРАБОТКА

Объект исследования – методы представления трехмерной графики.

Цель данной работы – разработка графического 3D-движка для построения трехмерных моделей.

В ходе данной работы были рассмотрены существующие методы и реализации представления трехмерной графики и особенности работы с ними.

Результатом работы стал программный движок способный выводить на экран построенные 3D-модели.

Практическая значимость работы заключается в освоении области работы с 3D, трехмерными моделями и приложениями для взаимодействия с ними.

## СОДЕРЖАНИЕ

ТЕРМИНЫ И ОПРЕДЕЛЕНИЯ .....	6
ВВЕДЕНИЕ .....	7
1 Обзор существующих движков .....	9
1.1 Редакторы трехмерной графики .....	9
1.2 Игровые движки .....	13
1.3 Научные и узконаправленные инструменты .....	16
1.4 Итог обзора .....	21
2 Методы представления трехмерной графики .....	22
2.1 Ray casting .....	22
2.2 Ray marching .....	25
2.3 Метод построения проекций .....	27
3 Реализация движка .....	28
3.1 Разбор метода построения проекций .....	28
3.2 Проектирование движка .....	36
3.3 Реализация движка .....	37
ЗАКЛЮЧЕНИЕ .....	52
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ .....	53
ПРИЛОЖЕНИЕ А (обязательное) Исходный код файла «main.py» .....	55
ПРИЛОЖЕНИЕ Б (обязательное) Исходный код файла «matrix_functions.py» .....	58
ПРИЛОЖЕНИЕ В (обязательное) Исходный код файла «object_3d.py» .....	59
ПРИЛОЖЕНИЕ Г (обязательное) Исходный код файла «camera.py» .....	62
ПРИЛОЖЕНИЕ Д (обязательное) Исходный код файла «projection.py» .....	65

## ТЕРМИНЫ И ОПРЕДЕЛЕНИЯ

В настоящей выпускной квалификационной работе применяют следующие термины с соответствующими определениями:

**Баг** – ошибка в программе или в системе, приводящая к неожиданному поведению программы и, как следствие, выдаче некорректного результата.

**Композитинг** – это процесс, во время которого два или более слоев объединяются, чтобы создать единое изображение. Объединяемые слои могут быть фотографиями, видеокадрами, 3D-анимацией и даже видеоэффектами.

**Плагин** – это дополнение (добавление новых функций и возможностей) для программного обеспечения и движков сайтов.

**Рендеринг** – термин в компьютерной графике, обозначающий процесс получения изображения по модели с помощью компьютерной программы.

**Риггинг** – это подготовка 3D-модели персонажа к анимации, при которой внутри заранее отрисованной заготовки размещается риг — набор виртуальных суставов и костей, устанавливаются закономерности его функционирования и возможные трансформации.

**Скульптинг** – разновидность компьютерного 3D-моделирования объектов из виртуального материала, напоминающего глину, путем его растягивания, сжатия, разглаживания и других манипуляций. Он позволяет создавать высокополигональные (до сотен миллионов полигонов) трехмерные модели с высоким уровнем детализации.

**Текстурирование** – это наложение растровых изображений (текстур) на 3D-модель для придания рельефности, фактуры и цвета.

**Трекинг** – определение местоположения движущегося объекта (нескольких объектов) во времени с помощью камеры. Алгоритм анализирует кадры видео и выдает положение движущихся целевых объектов относительно кадра.

## ВВЕДЕНИЕ

В современном мире 3D-моделирование играет важную роль. Визуализация трехмерных объектов используется в огромном количестве отраслей, начиная от разработки игр и анимации до проектирования инженерных систем, виртуального моделирования и даже медицины. Поэтому данная область информационных технологий важна не только в удовлетворении человеческих потребностей, но и в спасении человеческих жизней.

Также 3D-моделирование широко используется в науке. Например, построение трехмерных графиков дает визуальное представление сложных объектов и процессов, улучшая понимание и предсказание их характеристик, взаимодействий и поведения.

Неудивительно, что эта область сейчас стремительно развивается. Появляется много решений – 3D-движков, которые используются для различных задач. Большинство из них – профессиональные инструменты, которые предназначены для конкретной отрасли. Человек без определенных знаний вряд ли сможет понять работу этой программы, что может оттолкнуть как от использования данного решения, так и от отрасли в целом.

Для лучшего понимания области 3D-моделирования стоит разобраться в том, из чего состоит движок, потому что это может избавить от многих трудностей в освоении той или иной программы. Ведь многие даже не знают, что в основе программной реализации любого движка находятся основы линейной алгебры. В настоящее время не хватает решений, которые могли бы снизить порог вхождения и улучшить понимание и представление о том, из чего состоит 3D-пространство, какие процессы и какие вычисления в нем происходят.

Все вышеперечисленное обуславливает актуальность данной работы.

Целью данной работы является исследование методов представления 3D-графики и их реализация в собственном программном движке. В соответствии с целью были сформулированы следующие задачи:

- изучить существующие движки, описать их особенности, преимущества и недостатки,
- классифицировать их по области применения и сравнить внутри групп,
- сравнить их по общим характеристикам,
- выделить и описать наиболее распространенные методы представления трехмерной графики,
- выбрать метод представления для реализации в движке и описать его алгоритм,
- разработать на основе метода приложение и инструменты для его работы.

Данная работа даст материал для комплексного представления о 3D-графике, 3D-моделировании и создании программных инструментов для работы с трехмерными объектами.

## **1 Обзор существующих движков**

В этой главе представлен обзор различных движков для 3D-моделирования в различных областях. Обзор призван обеспечить понимание состояния рынка продуктов, предназначенных для 3D-моделирования, обозначить их особенности, преимущества и недостатки для составления требований для собственного движка, выбора методов отображения и проектирования 3D-пространства. Движки будут разделены по области их применения для более удобного сравнения. Также движки будут сравниваться по общим характеристикам, таким как архитектура, производительность и т. д.

### **1.1 Редакторы трехмерной графики**

Редакторы трехмерной графики предназначены для большинства задач, связанных с 3D-моделированием, таких как работа с графикой, визуализация, скульптинг, анимация. Модели, созданные в таких программах, применяются в основном в кинематографе и игровой индустрии, но мы можем видеть их повсюду. В настоящее время большое количество сайтов, сервисов использует трехмерные модели, которые помогают пользователям в использовании продукта. Качественная трехмерная графика выглядит приятнее и оставляет более яркие впечатления, чем простые рисунки.

Большинство таких редакторов просты в освоении, и каждый желающий пользователь сможет сделать в них первую простейшую модель. Тем не менее, для создания моделей более продвинутого уровня потребуется немало времени и сил.

Blender 3D – это бесплатный пакет программ для создания трехмерной графики с открытым исходным кодом (рисунок 1.1). Он охватывает всю цепочку процесса работы с 3D – моделирование, риггинг, анимация, симуляция, рендеринг, композитинг и отслеживание движения, а также редактирование видео и создание игр. Blender идеально подходит для

индивидуальных пользователей и малых студий, которые ценят единую рабочую среду и отзывчивый процесс разработки.

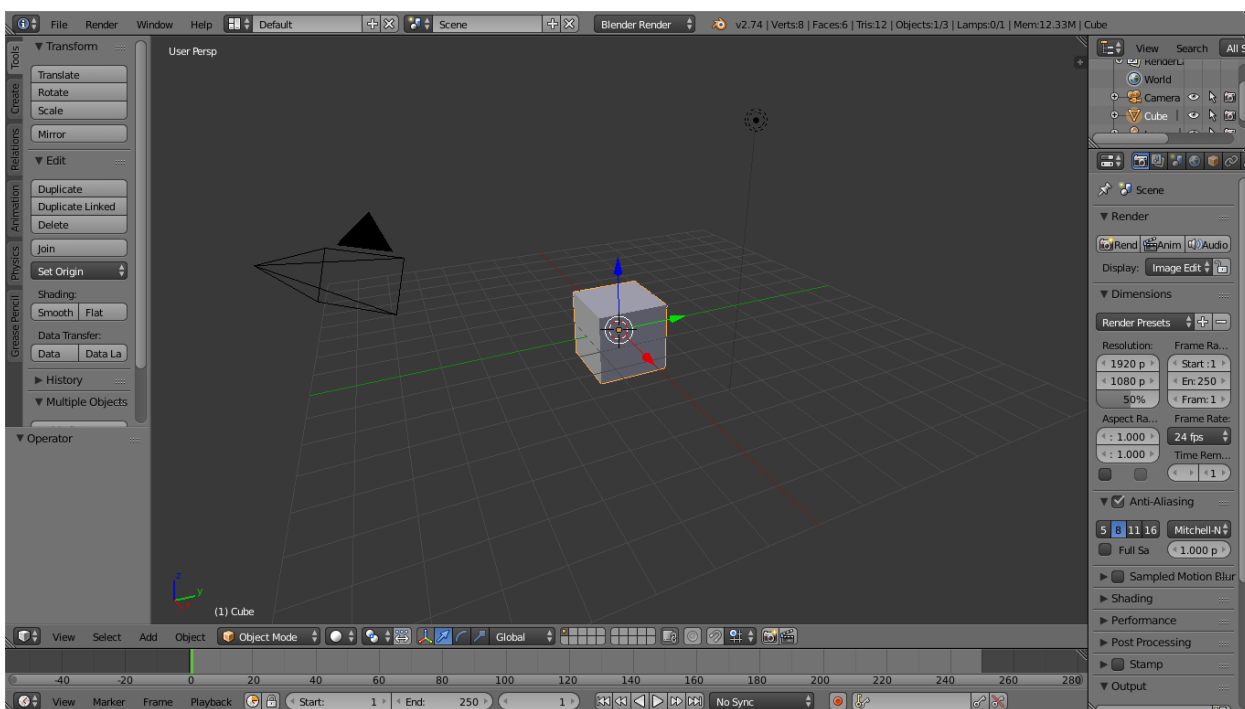


Рисунок 1.1 – Интерфейс программы Blender 3D

Blender написан на языке программирования C и C++, с использованием библиотеки пользовательского интерфейса Qt для разработки графического интерфейса пользователя (GUI). Кроме того, для расширения функциональности и создания дополнительных инструментов и плагинов Blender поддерживает язык программирования Python. Движок имеет модульную структуру, где каждый модуль отвечает за определенный аспект программы [1].

Главным преимуществом Blender является то, что он бесплатный для скачивания и использования. Blender поддерживает работу на самых распространенных операционных системах, включая Windows, Linux и macOS, что делает его доступным для широкого круга лиц. Сообщество Blender одно из самых больших, благодаря чему программа имеет большое количество обучающих материалов для пользователей любого уровня. Программа постоянно развивается и обновляется разработчиками, а благодаря

открытому исходному коду пользователи могут свободно изучать, изменять и распространять программу.

Также Blender обладает существенными недостатками. Несмотря на кажущуюся простоту, новых пользователей пугает сложный, перегруженный интерфейс. В программе очень много инструментов, которые сбивают с толку только что зашедших пользователей. Продвинутые пользователи жалуются на то, что многие инструменты, хотя почти все они настраиваемые, обрабатывают не так, как им хотелось бы. Частое обновление Blender приносит как дополнительный интересный функционал, так и баги, которые создают неудобства.

Blender 3D является одним из наиболее мощных и популярных инструментов для создания трехмерной графики. Поэтому ему находят применение в различных областях, включая анимацию, игровую разработку, визуализацию архитектуры, научные исследования и многое другое.

Ближайшим аналогом является программный пакет Cinema 4D или сокращенно C4D (рисунок 1.2). В этой программе поддерживается моделирование, рисование, скульптинг, композитинг, трекинг, анимация и высококачественный рендеринг. C4D дает потрясающие результаты, независимо от того, работаете ли вы самостоятельно или в команде [2].

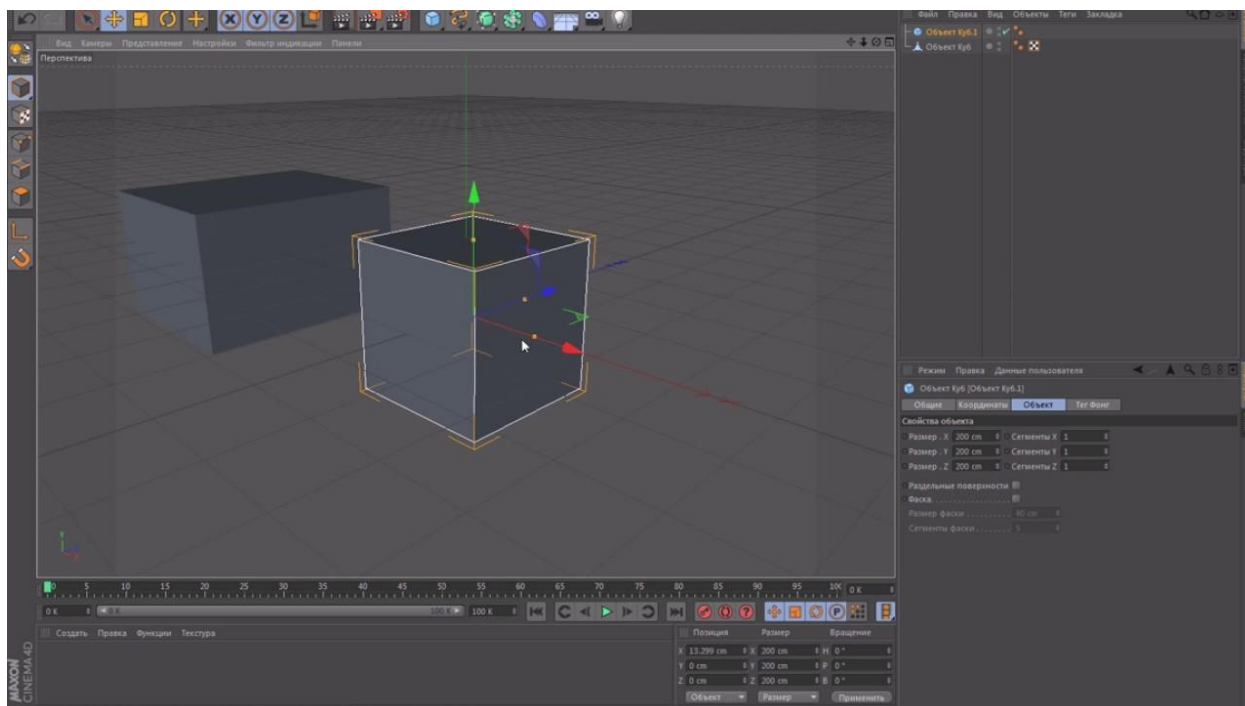


Рисунок 1.2 – Интерфейс программы Cinema 4D

Написан он на Coffee++ – небольшой язык программирования, компилируемый на C++, и также имеет модульную структуру [3]. За счет такого сочетания имеет высокую производительность и качество.

Из плюсов сами разработчики выделяют интуитивно понятный интерфейс, который делает программу доступной для пользователей на всех уровнях опыта. Они также предлагают широкий набор мощных инструментов для моделирования и анимации, который позволяет реализовывать самые сложные проекты. C4D поддерживает множество форматов файлов, что облегчает совместную работу с другими программами и обмен файлами. Программа также имеет большое и активное сообщество.

К недостаткам можно отнести то, что C4D является коммерческим программным обеспечением, поэтому пользоваться программой на бесплатной основе нельзя. Также, в отличие от Blender, инструменты мощные, но практически не настраиваемые и маловариативные.

Из особенностей можно выделить то, что программа предоставляет широкий выбор рендеринга – это позволяет пользователям выбрать наиболее подходящий для своих потребностей стиль и качество рендеринга. Программа

относится к редакторам трехмерной графики, но у нее есть выделяющаяся область применения – анимация. В C4D предоставлены широкие возможности для создания анимации персонажей, объектов и камер, поддерживается динамическая и физическая анимации, поэтому чаще всего программу используют в киноиндустрии.

Представленные выше графические редакторы дают ясное представление о рынке программного обеспечения.

## **1.2 Игровые движки**

Другим типом программного обеспечения являются игровые движки. Игровой движок – это программное обеспечение, которое предоставляет разработчикам инструменты и функциональность для создания видеоигр. Он включает в себя наборы инструментов для работы с графикой, физикой, звуком, искусственным интеллектом, анимацией и другими аспектами игрового процесса. Эти движки позволяют разработчикам концентрироваться на креативной части процесса, вместо написания кода с нуля для каждой игры.

Самым популярным игровым движком является Unity (рисунок 1.3). Unity – один из самых популярных и востребованных игровых движков в индустрии разработки видеоигр. Unity широко используется для разработки игр различных жанров, включая аркады, стратегии, симуляторы, платформеры и многие другие. Его простота в использовании, мощные возможности и кроссплатформенность делают его идеальным выбором для как начинающих, так и опытных разработчиков. Unity написан на языке программирования C++. Он обеспечивает высокую производительность и эффективное использование ресурсов компьютера, что особенно важно для игровых движков. Кроме того, Unity использует C# для написания скриптов и логики игры, что делает его доступным и привлекательным для широкого круга разработчиков. Использование комбинации этих языков программирования обеспечивает

гибкость, производительность и расширяемость Unity как программного обеспечения [4].

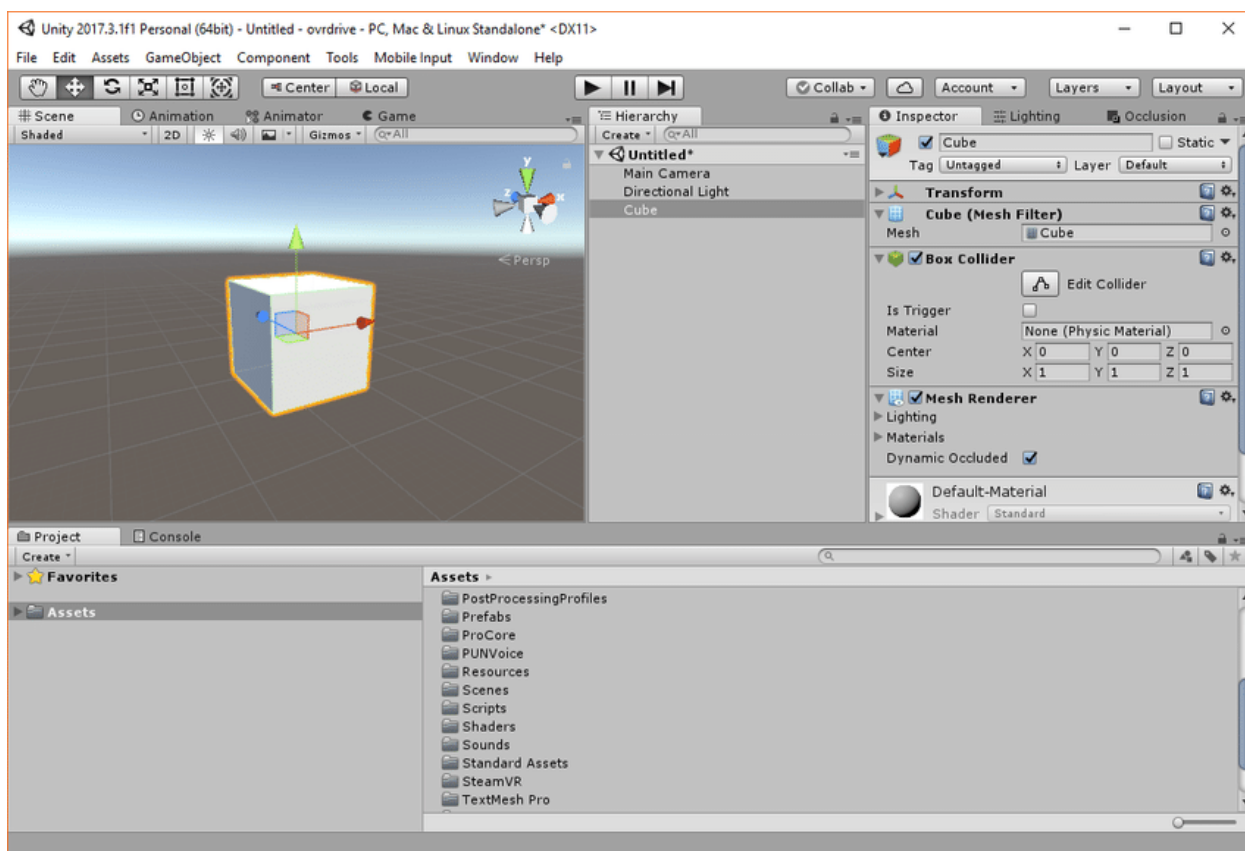


Рисунок 1.3 – Интерфейс программы Unity

Unity поддерживает разработку игр для широкого спектра платформ, включая ПК, мобильные устройства (iOS и Android), консоли (PlayStation, Xbox, Nintendo Switch), виртуальную реальность (VR), а также веб-браузеры. Движок предоставляет мощные инструменты для создания качественной графики и анимации. Включает в себя поддержку шейдеров, освещения, систем частиц, риггинга и анимации персонажей. Unity имеет обширную экосистему, включая магазин активов, где разработчики могут найти готовые ресурсы, плагины и инструменты для ускорения разработки игр, а так как сообщество у движка достаточно большое, то магазин постоянно активно пополняется.

Хотя Unity является мощным и широко используемым игровым движком, у него также есть некоторые недостатки, которые стоит учитывать. Бесплатная версия Unity имеет некоторые ограничения, такие как ограниченные возможности мобильной разработки, отсутствие некоторых продвинутых функций и водяные знаки на экспортированных изображениях и анимациях. Unity-проекты могут иметь большой размер, особенно если в них используются множество ресурсов или активов. Это может привести к увеличению времени загрузки игр и местоположению на диске.

Несмотря на то, что Unity дружелюбен для новичков, для профессиональной разработки могут потребоваться дополнительные знания и навыки в области оптимизации, работе с большими проектами и использовании продвинутых функций. В сравнении с некоторыми другими игровыми движками, Unity может иметь ограничения в области графической гибкости и возможностей, особенно для создания фотореалистичных изображений и визуализаций. Движком, который не имеет этих недостатков, является Unreal Engine.

Unreal Engine – это мощный игровой движок (рисунок 1.4). Unreal Engine известен своими впечатляющими графическими возможностями и реалистичной визуализацией. Он включает в себя передовые технологии рендеринга, такие как реалистичные материалы, динамические тени и освещение, а также поддержку технологий трассировки лучей. Написан на языке C++, что позволяет создавать игры для большинства операционных систем, а также на различных портативных устройствах. Имеет модульную структуру, поддерживает различные системы отрисовки, воспроизведения звука, средства голосового воспроизведения текста, распознавание речи, модули для работы с сетью и поддержки различных способов ввода. Unreal Engine предоставляет интуитивный визуальный интерфейс для создания логики игры с помощью графических скриптов, называемых "блюпринтами". Также поддерживается программирование на C++ для создания более сложной и производительной логики [4].

С 2015 года доступен для использования бесплатно, но разработчики обязаны платить роялти только при выпуске коммерческих продуктов. Это делает его доступным для широкого круга разработчиков и проектов.

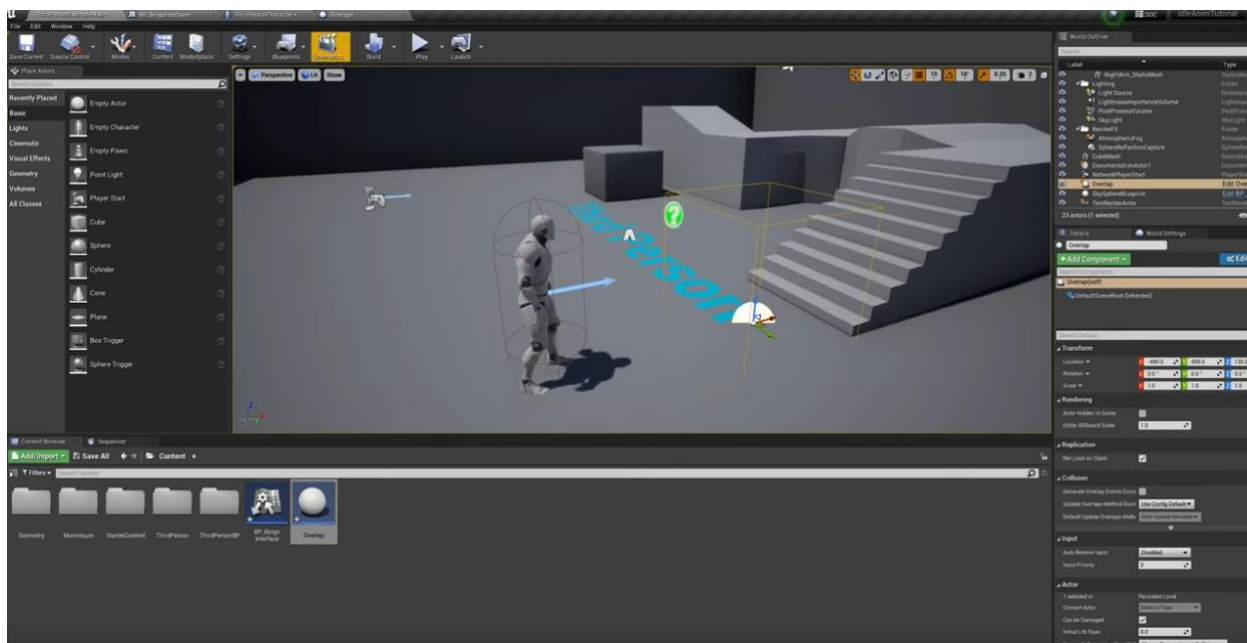


Рисунок 1.4 – Интерфейс программы Unreal Engine

Но движок имеет и значительные недостатки. Потрясающая графика требует высокопроизводительное оборудование, большое количество свободного места на компьютере; длительное время загрузки и рендеринга, сложности с оптимизацией проекта может оттолкнуть новичков в освоении данного продукта.

### 1.3 Научные и узконаправленные инструменты

Современные научные и инженерные исследования требуют высокоточного и узконаправленного анализа трехмерных данных. Это специализированные инструменты, в которых могут разобраться только профессионалы в той или иной области, поэтому сравнивать их стоит по архитектуре, удобству применения со стороны профессионалов и производительности.

Avizo – это программное обеспечение для визуализации и анализа научных и инженерных данных в трех измерениях (рисунок 1.5). Оно широко используется в научных исследованиях, медицинском моделировании, геологических исследованиях, инженерном проектировании и других областях. Avizo позволяет визуализировать и интерактивно исследовать сложные трехмерные датасеты, включая визуализацию облачных точек, воксельных данных, поверхностей, объемов и других типов данных. Он предоставляет широкий набор инструментов для анализа данных, таких как измерение, фильтрация, сегментация, реконструкция поверхности, извлечение признаков и др. Эти инструменты позволяют находить закономерности и выявлять характеристики данных [6].

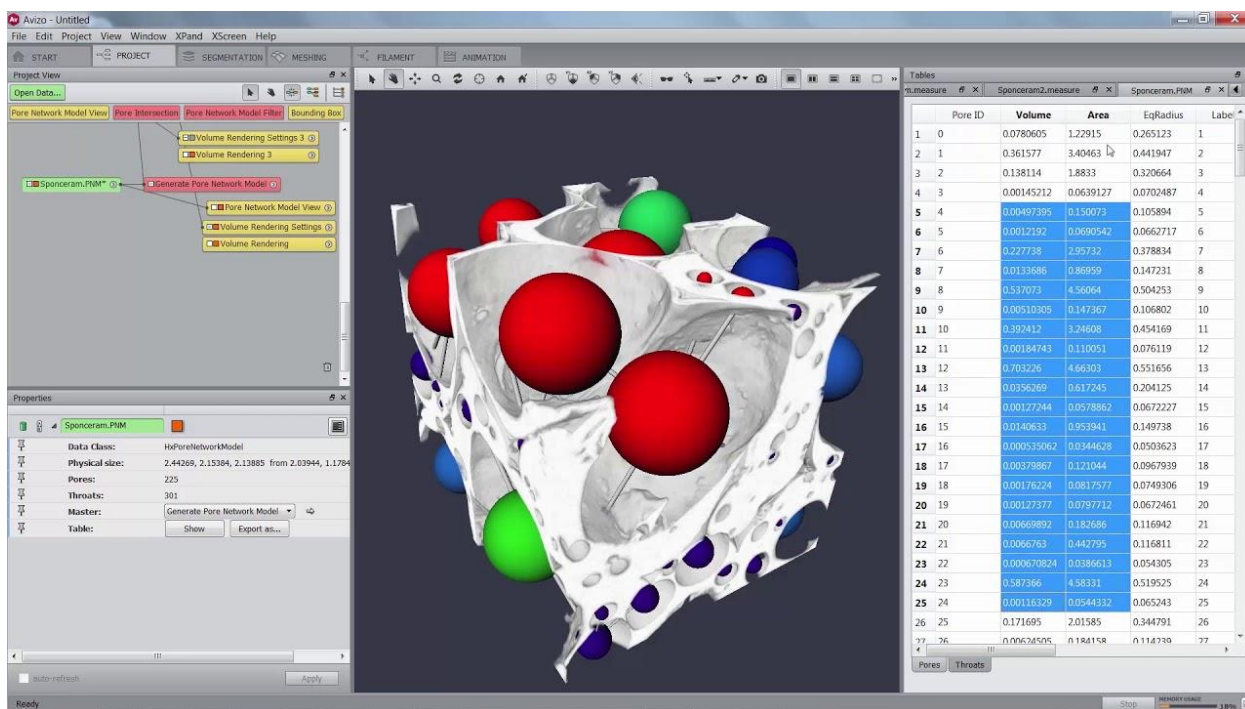


Рисунок 1.5 – Интерфейс программы Avizo

Архитектура Avizo организована вокруг модульного и расширяемого подхода, что делает его гибким и мощным инструментом для визуализации и анализа научных данных в трех измерениях. Относительно других научных движков, написанных на других языках программирования, преимущества

Avizo, разработанного на C++, включают высокую производительность, эффективное управление ресурсами и широкую гибкость в разработке.

К недостаткам можно отнести то, что Avizo требует достаточно мощного оборудования для эффективной работы, особенно при обработке больших объемов данных. Пользователи могут столкнуться с ограниченными возможностями Avizo, особенно в сравнении с другими программными решениями, а редкие обновления для расширения функциональности и исправления ошибок только ярче выявляют этот недостаток.

AutoCAD – это система автоматизированного проектирования, предназначенное для создания двумерных чертежей, проектирования и моделирования (рисунок 1.6). AutoCAD является одним из самых популярных программных продуктов, а его широкий функционал и гибкость делают его неотъемлемым инструментом для проектирования и создания планов зданий, схем, электрических схем и других технических чертежей [7].

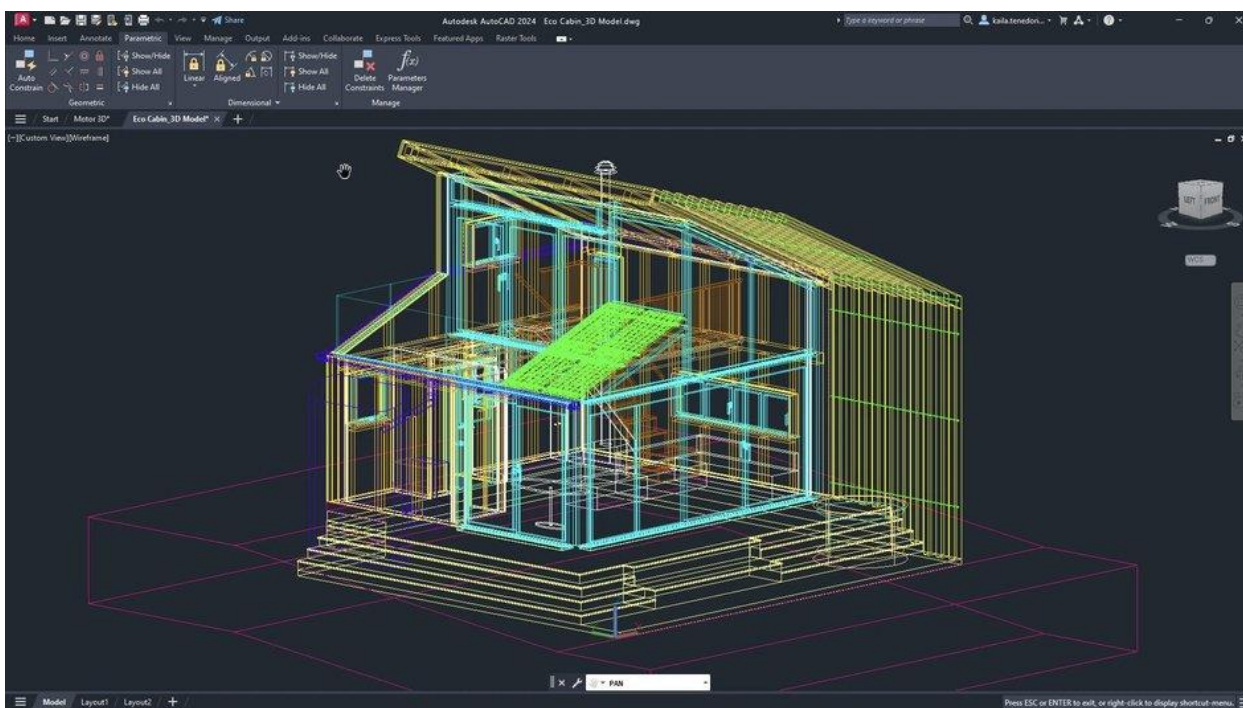


Рисунок 1.6 – Интерфейс программы AutoCAD

AutoCAD написан на AutoLISP – диалект языка Lisp, созданный специально для использования с полной версией AutoCAD и его

производными [8]. Он обеспечивает эффективное использование ресурсов и простоту использования даже для непрофессиональных пользователей. Также в движке используется C++, что дает высокую производительность.

AutoCAD имеет клиент-серверную архитектуру, где пользовательский интерфейс (клиент) взаимодействует с графическим ядром (сервером), обеспечивая визуализацию и манипулирование графическими объектами. Графическое ядро обрабатывает графические команды и отрисовывает объекты на экране.

Несмотря на то, что AutoCAD достаточно простой и мощный движок, многих пользователей не устраивает устаревший, по сравнению с другими программными решениями, интерфейс, а также собственный формат файла (.dwg), который не всегда легко читается и редактируется другими программами. Также на данный момент не доступен пользователям в РФ.

VisIt – это мощный и гибкий программный инструмент для визуализации и анализа научных данных (рисунок 1.7). VisIt поддерживает широкий спектр форматов данных, включая данные о сетке, томограммы, многомерные данные и многие другие. Это делает его универсальным инструментом для визуализации данных из различных научных областей. Движок написан в основном на C++ и имеет модульную архитектуру. В отличие от многих других научных инструментов VisIt бесплатный и с открытым исходным кодом. В совокупности это делает его популярным выбором даже среди непрофессиональных пользователей. Но для обработки больших объемов данных требуется мощное аппаратное обеспечение [9].

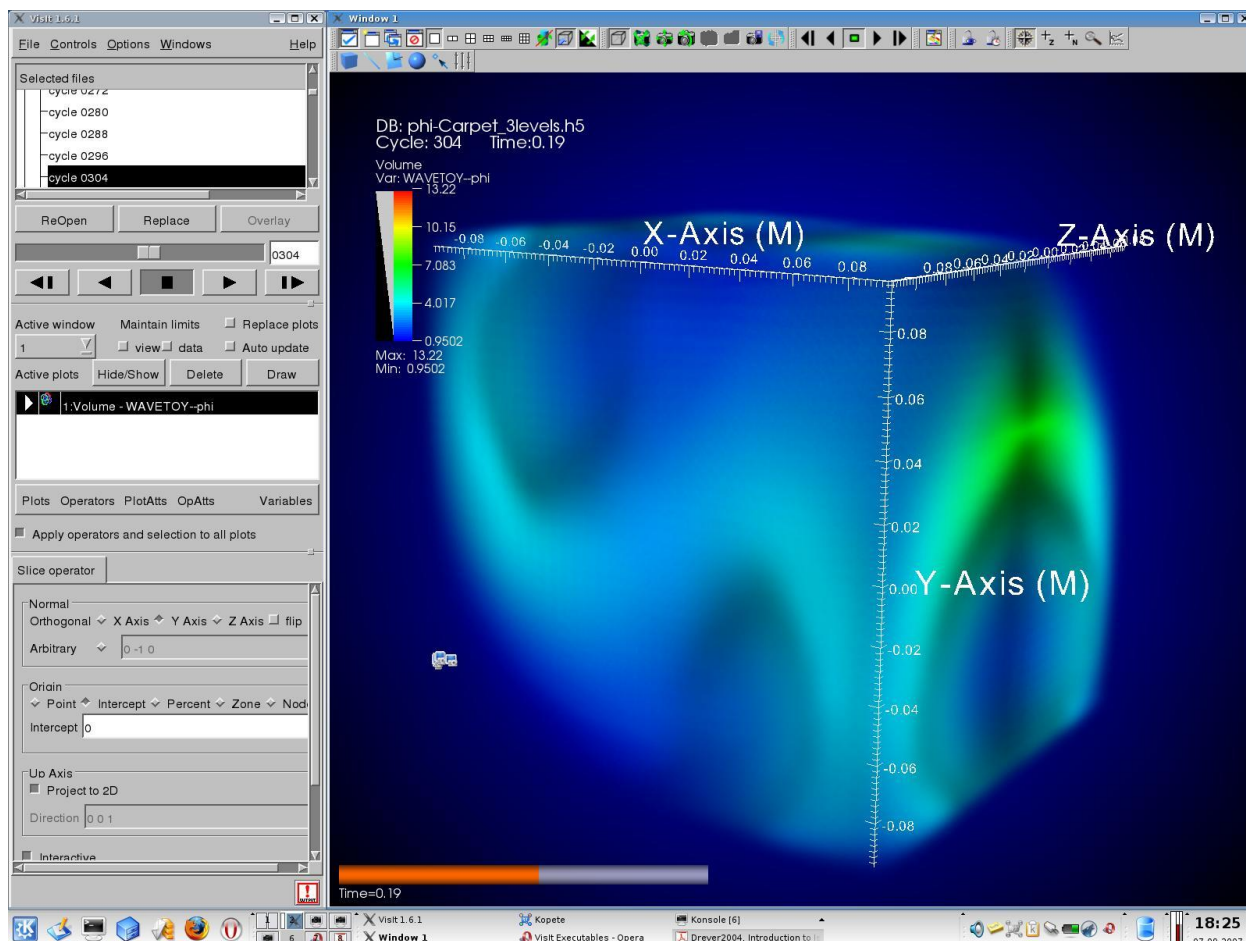


Рисунок 1.7 – Интерфейс программы VisIt Software

SketchUp – программа для 3D-дизайна и архитектурного проектирования (рисунок 1.8). В основном используется для моделирования жилых домов, мебели и интерьера. Относится к специализированным редакторам трехмерной графики. Также, как и другие движки, написан на C++ и имеет модульную архитектуру. Программа имеет простой и понятный пользовательский интерфейс, а разработчики предлагают бесплатные и платные версии программы для пользователей различных уровней. Несмотря на узкую специализацию программа имеет широкий функционал, что позволяет ее использовать в различных целях. И все же ограниченные возможности визуализации и моделирования не позволяют сравнивать программу с другими редакторами трехмерной графики. Также некоторые пользователи могут столкнуться с ограниченной поддержкой для конкретных функций или потребностей [10].

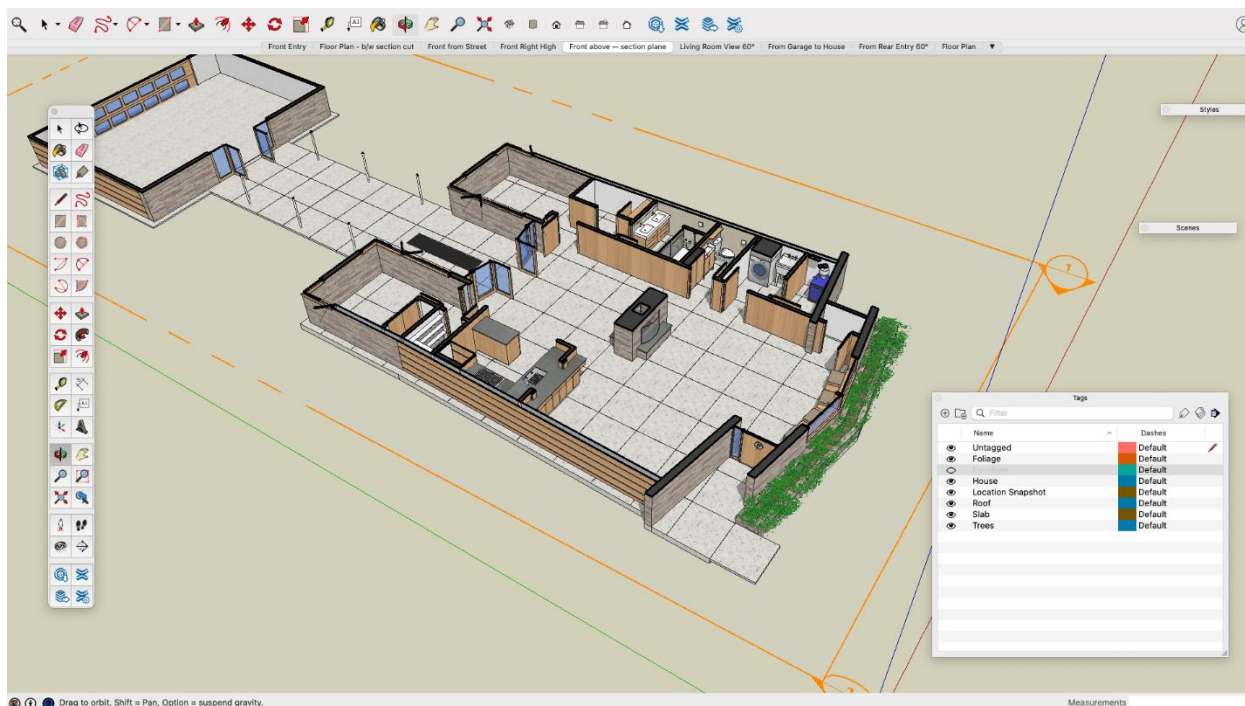


Рисунок 1.8 – Интерфейс программы SketchUp

## 1.4 Итог обзора

Почти все движки из обзора обладают высокой производительностью. В основном это достигается благодаря выбору языка программирования и архитектуры.

C++ является компилируемым языком, что позволяет создавать эффективный и оптимизированный код. Большинство движков, написанных на C++, используют графические библиотеки и фреймворки для интегрирования в проект OpenGL. Это дает возможность создать производительный графический движок благодаря использованию вычислений на графическом ядре компьютера, которое предназначено для быстрого перемножения матриц и векторов [11].

Модульная архитектура способствует упрощению разработки, обслуживания и масштабирования графических приложений.

## **2 Методы представления трехмерной графики**

В этой главе описаны различные существующие методы представления трехмерной графики. Все они использовались в движках в разное время развития информационных технологий, исходя из этого некоторые методы более просты в реализации и скорости вычислений, но имеют меньше возможностей для представления моделей.

### **2.1 Ray casting**

Рейкастинг (англ. ray casting – бросание лучей) – один из методов рендеринга в компьютерной графике, при котором сцена строится на основе замеров пересечения лучей с визуализируемой поверхностью. Был одним из первых методов создания 3D-изображения, не требующий больших вычислительных затрат [12].

Термин появился в 1982 году, а сам метод использовался для рендеринга моделей конструктивной блочной геометрии. Метод был достаточно простым в понимании, так как модели состояли из примитивных трехмерных фигур – сфера, куб, цилиндр, конус и т.д. Для их отображения требовалось задать функции, принимающие параметры фигур, например, положение центра в пространстве и радиус для сферы, а для создания более сложных моделей требовалось применить к нескольким примитивам одну из булевых операций на множествах – объединение, пересечение и разность. Такое представление позволяло заметно сократить занимаемое место сцены и упростить вычисления для рендеринга, так как двоичные логические операции выполняются на процессоре быстрее всего. Правда для более реалистичного отображения, требовались методы, которые значительно усложняли структуру программ и замедляли вычисления.

В основном рейкастинг использовался в игровых движках, так как стационарные компьютеры еще не имели больших вычислительных мощностей. Даже графика, которая использовалась в играх того времени,

называется псевдотрехмерной, потому что она только имитирует трехмерное пространство. Самыми яркими реализациями такой графики являются игры «Wolfenstein 3D» и «Doom», вышедшие в 1992 и 1993 годах соответственно. В них игрок помещен в напоминающее лабиринт трехмерное помещение, из которого он должен выбраться, сражаясь с врагами, управляемыми компьютером. Разработчики, взяв существующих движок, использующий рейкастинг, усовершенствовали его, сосредоточив силы на том, чтобы сделать игру плавной и быстрой при использовании графики более высокого разрешения, а также добавив возможность взаимодействия с миром (например, открытие дверей, столкновение с объектами, расположение объектов не только на стенах и т. д.). Но в действительности персонаж перемещался по двумерной плоскости, а благодаря методу рейкастинга и законам перспективы создавалась та самая иллюзия 3D-пространства [13].

Алгоритм рейкастинга в этих играх достаточно прост. Наблюдатель представляется точкой на двумерной плоскости. Он может перемещаться вдоль осей X и Y и менять направление взгляда, что дает возможность хранить всю информацию об игроке в виде трех переменных. Направление взгляда – это область угла, в пределе которого игрок будет видеть предмет. В этой области с шагом равным некоторой величине от наблюдателя бросается пучок лучей. Лучи тоже имеют некоторую длину для того, чтобы ограничить дальность видимости. Каждый луч, встретив препятствие, запоминает расстояние от наблюдателя до встретившегося объекта. В зависимости от этого расстояния движок визуализирует препятствие по законам перспективы – чем объект дальше, тем его размер будет меньше, и наоборот, чем объект ближе, тем размер больше. Визуализация этого алгоритма представлена на рисунке 2.1.



Рисунок 2.1 – Визуализация алгоритма Ray casting в псевдотрехмерной графике

В этих играх пространство (карта) представлена в виде двумерного массива, в котором элемент может иметь 2 состояния. Условно, «0» – пустое пространство, а «1» – стена или препятствие. Луч, выпущенный из наблюдателя, «шагает», пока не встретит на своем пути препятствие. На экране определяется линия горизонта, проходящая ровно по центру экрана горизонтально. Это нужно для того, чтобы выделить границы пола и неба, которые заданы заранее и определяют вертикальную область видимости, и по ним вычислить относительную высоту объекта в кадре [12].

Сейчас такая графика выглядит очень простой. Алгоритм настолько легкий, что энтузиасты пытаются запустить данные игры на компьютерах, не предназначенных для тяжелых вычислений. Но технологию рейкастинга доработали, вследствие чего можно отрисовать более реалистичную картинку. Для этого используется несколько иной алгоритм, но картинка, получаемая таким методом, является полноценным 3D-изображением.

Наблюдатель также является точкой, но уже в трехмерном пространстве, и имеет направление взгляда, определяемое трехмерным нормированным вектором. Объекты на сцене заданы параметрически. Для каждого пикселя монитора как бы выпускается луч, и при пересечении луча с объектом, пиксель красится в цвет объекта. Для каждого из примитивов существуют формулы пересечения луча с объектом – это преобразованные уравнения, которые

позволяют разработчикам легче запрограммировать вычисление расстояний. Также этот алгоритм позволяет создать эффект освещения. Для этого создается еще одна точка с направлением взгляда – источник освещения. У граней объекта вычисляется нормаль – перпендикуляр к этой грани. Чем больше совпадают вектор направления источника освещения и нормаль грани, тем эта грань будет светлее, а цвет этой грани – ярче.

Для реализации такого метода нужна гораздо более высокая вычислительная мощность машины, ведь количество пикселей на экране может достигать до нескольких миллионов. Поэтому этот алгоритм чаще всего выполняется с помощью шейдеров – программ, которые выполняются на видеокарте.

При зарождении компьютерной графики синонимом к рейкастингу использовался термин «ray tracing» (трассировка лучей), но в настоящее время это совершенно другая технология. В начале она действительно использовалась для построения моделей. Сейчас же это технология, которая позволяет создать реалистичное освещение, отражение и тени. Его можно было бы использовать для реализации движка, но метод очень требовательный к производительности системы. Производители современных видеокарт даже разработали новую архитектуру, позволяющую проводить трассировку лучей в реальном времени.

## **2.2 Ray marching**

Реймарчинг (англ. ray marching – марширование лучей) – современный метод рендеринга в трехмерной компьютерной графике. Он схож с рейкастингом, но выпускаемые лучи и пересечения с объектами обрабатываются несколько иначе. Этот метод более требовательный к ресурсам компьютера, потому что производится больше вычислений, но отрисовка объектов становится более точной, что позволяет добиться большего реализма, а возможности такого движка гораздо шире. Например,

его можно использовать интересных и красивых взаимодействий между объектами.

Наблюдатель также представлен точкой в пространстве, имеющей направление взгляда. Для каждого пикселя выпускается луч в область видимости камеры. Первоначально луч имеет длину, равную расстоянию от наблюдателя до ближайшего объекта. Далее вычисляется расстояние снова до ближайшего объекта, но от точки, в которой остановился луч, и луч перемещается в том же направлении уже на это расстояние. Это происходит до тех пор, пока луч не столкнется с объектом или не пройдет определенное количество шагов. Таким образом гораздо точнее вычисляется расстояние до сферы и более точно подбирается цвет для окрашивания пикселей [14].

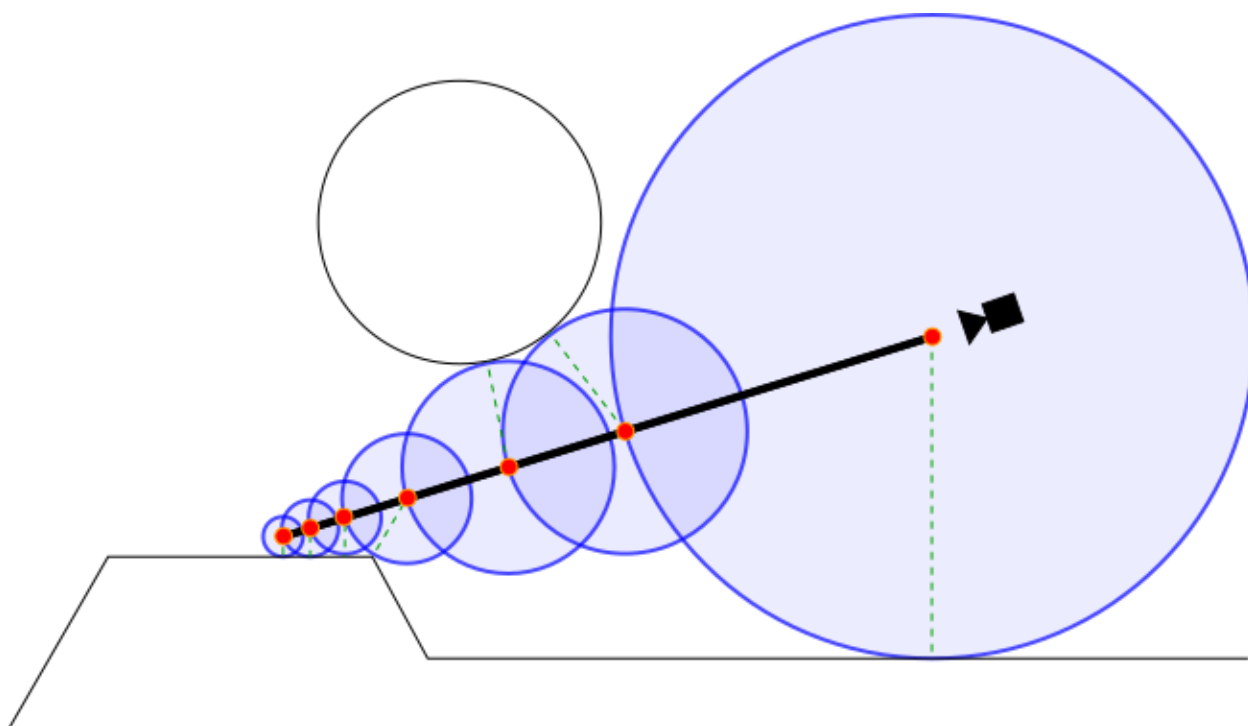


Рисунок 2.2 – Визуализация алгоритма Ray marching

Этот метод за счет простоты исполнения и широких графических возможностей и визуальных эффектов используется в кинематографе.

## 2.3 Метод построения проекций

Большинство современных движков создают изображение с помощью построения двумерных проекций. Этот метод использует начертательную геометрию и декартову систему координат и работает с настоящим трехмерным пространством, так как объекты задаются трехмерными точками (вершинами). Грани объекта – это полигоны, связывающие несколько вершин. Двумерное изображение получается благодаря проецированию трехмерных точек на двумерную плоскость проекции камеры. Видеокарты созданы для эффективного вычисления матричных операций, поэтому ее наличие заметно ускорит отрисовку изображений и даже позволит производить ее в реальном времени. Процессор же справится с такой задачей эффективно при небольшом количестве точек.

Для начала задается ортонормированный базис – декартова система координат с началом в точке  $(0, 0, 0)$  и осями  $OX$ ,  $OY$ ,  $OZ$ . Будем называть ее мировой системой координат. Ориентация системы зависит от разработчика и ее выбор не играет особой роли. Далее описывается объект – для него задаются вершины, грани, а также собственная (локальная) система координат, относительно которой будут изменяться координаты вершин. Создается объект камера, которая имеет положение и направление взгляда, задаваемое тремя ортонормированными векторами. Для определения той части пространства, которая будет проецироваться на экран, используется усеченная пирамида обзора, задаваемая параметрически. Далее для каждой попавшей в пирамиду вершины вычисляется проекция на одну из ее плоскостей, которая является нашим изображением [15].

Этот метод является самым ресурсоемким, но в то же время самым точным и наиболее приближенным к реальности. Алгоритмы, которые используются в методе, задействуют основы векторной алгебры, начертательной геометрии, матричного исчисления и т.д.

### 3 Реализация движка

В этой главе выбран метод представления 3D-графики и инструменты для программной реализации собственного движка.

#### 3.1 Разбор метода построения проекций

В приложении будет использоваться метод построения проекций для отображения 3D-графики. Он наиболее распространен среди графических средств, имеет за собой широкую математическую основу, подробные описания алгоритмов и хорошо отражает работу с трехмерными объектами.

Пусть задан ортонормированный базис – декартова система координат с началом в точке  $O(0, 0, 0)$  и осями  $Ox$ ,  $Oy$  и  $Oz$ . Он выступает в качестве мировой системы координат или системы координат «сцены». Объекты задаются массивом из точек, которые называются вершинами объекта, и граней.

Рассмотрим точку  $A$  в трехмерном пространстве. Координатами этой точки будем считать координаты радиус-вектора  $OA$  в системе координат  $Oxyz$  [15].

Соединив вершины между собой, можно получить визуальное представление объекта в пространстве, но не обязательно соединять все вершины. Соединение вершин производится по заданным граням, где для каждой грани указаны ее вершины [16].

Каждый объект имеет собственную систему координат. Любое действие с объектом означает перемещение локальной системы координат объекта в нужное положение в мировой сцене. Основные действия над объектами это:

- перемещение,
- вращение,
- масштабирование,
- сдвиг.

Сдвиг не будем рассматривать и реализовывать, так как вполне достаточно наличия матрицы перемещения и заданного заранее шага.

Любое изменение вершины в пространстве осуществляется благодаря умножению вектора координат вершины на матрицу, предназначенную для определенного действия, что соответствует формуле (1).

$$V' = V \times M, \quad (1)$$

где  $V'$  – вектор новых координат вершины.

Важно отметить, что для программной реализации нужно ввести использование однородных координат. Это делается по двум причинам:

- перемещение в трехмерном пространстве не может быть представлено как квадратная матрица размерностью 3,
- перспективная проекция не может быть представлена как квадратная матрица размерностью 3.

Поэтому в программной реализации вершины будут представлены как четырехмерный вектор – первые 3 координаты будут хранить координаты вершины по осям OX, OY и OZ, а четвертая вершина (обозначим ее буквой W) будет равняться единице. Следовательно, все матрицы действий будут иметь размерность 4 [17].

Матрица перемещения позволяет двигать объект вдоль любой из осей и тем самым устанавливает положение объекта в мировой сцене. Она включает в себя значения a, b, c – смещения по осям OX, OY и OZ соответственно. Изображены на рисунках 3.1, 3.2.

$$\begin{pmatrix} 1 & 0 & 0 & a \\ 0 & 1 & 0 & b \\ 0 & 0 & 1 & c \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Рисунок 3.1 – Матрица перемещения для правосторонней системы координат

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ a & b & c & 1 \end{pmatrix}$$

Рисунок 3.2 – Матрица перемещения для левосторонней системы координат

Матрица масштабирования позволяет устанавливать размер объекта. При это объект можно растягивать вдоль любой из осей. Включает в себя значения  $a$ ,  $b$ ,  $c$  – изменения угла по осям  $OX$ ,  $OY$  и  $OZ$  соответственно.

$$\begin{pmatrix} a & 0 & 0 & 0 \\ 0 & b & 0 & 0 \\ 0 & 0 & c & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Рисунок 3.3 – Матрица масштабирования

Матрица вращения позволяет вращать объект по одной из осей. Соответственно для каждой из осей представлена собственная матрица вращения. Каждая из них принимает угол поворота по какой-либо из осей. Все матрицы представлены на рисунках 3.4 – 3.9.

$$\begin{array}{cccc}
 1 & 0 & 0 & 0 \\
 0 & \cos(x) & -\sin(x) & 0 \\
 0 & \sin(x) & \cos(x) & 0 \\
 0 & 0 & 0 & 1
 \end{array}$$

Рисунок 3.4 – Матрица поворота относительно оси OX для правосторонней системы координат

$$\begin{array}{cccc}
 1 & 0 & 0 & 0 \\
 0 & \cos(x) & \sin(x) & 0 \\
 0 & -\sin(x) & \cos(x) & 0 \\
 0 & 0 & 0 & 1
 \end{array}$$

Рисунок 3.5 – Матрица поворота относительно оси OX для левосторонней системы координат

$$\begin{array}{cccc}
 \cos(y) & 0 & \sin(y) & 0 \\
 0 & 1 & 0 & 0 \\
 -\sin(y) & 0 & \cos(y) & 0 \\
 0 & 0 & 0 & 1
 \end{array}$$

Рисунок 3.6 – Матрица поворота относительно оси OY для правосторонней системы координат

$$\begin{array}{cccc}
 \cos(y) & 0 & -\sin(y) & 0 \\
 0 & 1 & 0 & 0 \\
 \sin(y) & 0 & \cos(y) & 0 \\
 0 & 0 & 0 & 1
 \end{array}$$

Рисунок 3.7 – Матрица поворота относительно оси OY для левосторонней системы координат

$$\begin{array}{cccc}
 \cos(z) & -\sin(z) & 0 & 0 \\
 \sin(z) & \cos(z) & 0 & 0 \\
 0 & 0 & 1 & 0 \\
 0 & 0 & 0 & 1
 \end{array}$$

Рисунок 3.8 – Матрица поворота относительно оси OZ для правосторонней системы координат

$$\begin{array}{cccc}
 \cos(z) & \sin(z) & 0 & 0 \\
 -\sin(z) & \cos(z) & 0 & 0 \\
 0 & 0 & 1 & 0 \\
 0 & 0 & 0 & 1
 \end{array}$$

Рисунок 3.9 – Матрица поворота относительно оси OZ для левосторонней системы координат

Для правильного построения граней объекта нужно знать номера вершин. В программной реализации для построения граней объекта будем обращаться к номерам вершин в массиве.

Чтобы спроецировать трехмерный объект на двумерную плоскость, надо понимать, какую часть пространства нужно поместить в проекцию. Для этого используется усеченная пирамида обзора с ближней и дальней плоскостью отсечения, которая изображена на рисунке 3.10. Эта пирамида базируется на параметрах камеры, которая определяет, какая часть мировой сцены попадет на итоговое изображение.

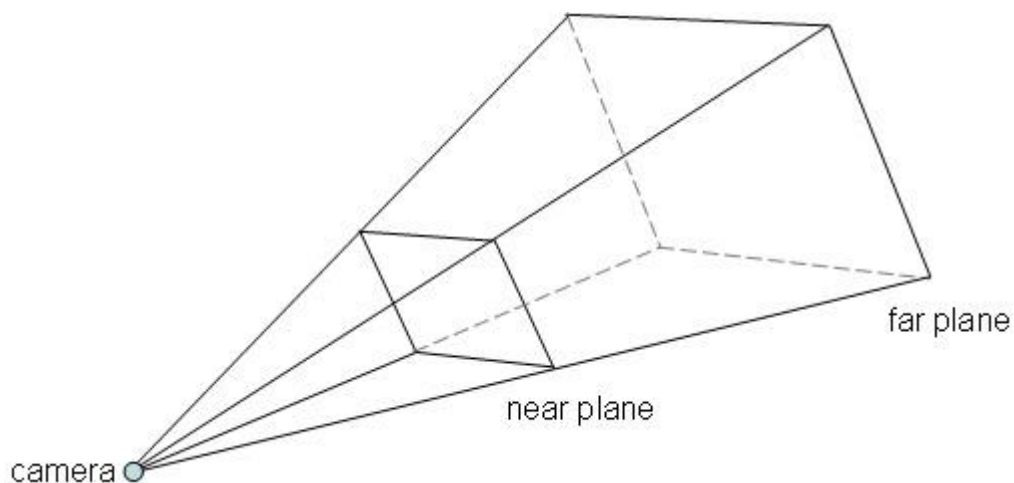


Рисунок 3.10 – Усеченная пирамида обзора

Камера имеет собственную систему координат, которая характеризуется положением в мировой сцене, а также векторами ориентации – угол тангажа (ось  $Ox$  для правосторонней и ось  $Oz$  для левосторонней), рысканья (ось  $Oy$ ) и крена (ось  $Oy$  для правосторонней и ось  $Ox$  для левосторонней), которые представлены на рисунке 3.11. Также в параметры камеры входит горизонтальная область видимости, которая задается заранее, и вертикальная область видимости, которая вычисляется на основании горизонтальной области видимости и соотношения величин заданного разрешения окна программы.

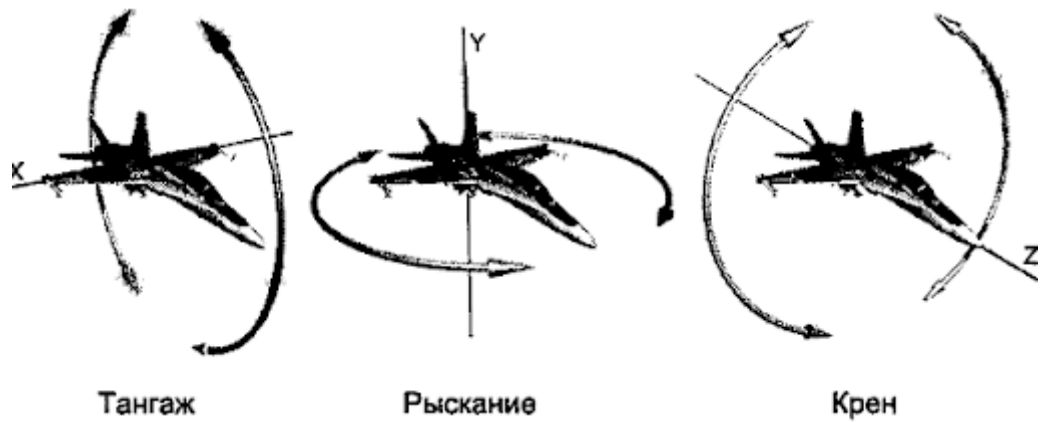


Рисунок 3.11 – Наглядное изображение поворотов относительно каждой из осей

Далее следует попасть в систему координат камеры. Для этого сначала нужно переместить мир таким образом, чтобы положение камеры совпало с началом системой координат мировой сцены. После нужно повернуть систему координат камеры таким образом, чтобы она совпала с системой координат мировой сцены. Для этого используются матрица перемещения камеры (рисунки 3.12, 3.13) и матрица вращения камеры (рисунки 3.14, 3.15).

$$\begin{array}{cccc}
 1 & 0 & 0 & -a \\
 0 & 1 & 0 & -b \\
 0 & 0 & 1 & -c \\
 0 & 0 & 0 & 1
 \end{array}$$

Рисунок 3.12 – Матрица перемещения камеры для правосторонней системы координат

$$\begin{array}{cccc}
 1 & 0 & 0 & 0 \\
 0 & 1 & 0 & 0 \\
 0 & 0 & 1 & 0 \\
 -a & -b & -c & 1
 \end{array}$$

Рисунок 3.13 – Матрица перемещения камеры для левосторонней системы координат

$$\begin{array}{cccc}
 rx & ry & rz & 0 \\
 ux & uy & uz & 0 \\
 fx & fy & fz & 0 \\
 0 & 0 & 0 & 1
 \end{array}$$

Рисунок 3.14 – Матрица поворота камеры для правосторонней системы координат

$$\begin{array}{cccc}
 rx & ux & fx & 0 \\
 ry & uy & fy & 0 \\
 rz & uz & fz & 0 \\
 0 & 0 & 0 & 1
 \end{array}$$

Рисунок 3.13 – Матрица поворота камеры для левосторонней системы координат

В матрице поворота  $r$ ,  $u$ ,  $f$  – векторы тангажа, рысканья и крена соответственно, а индексы  $x$ ,  $y$ ,  $z$  – координаты этих векторов.

Таким образом произведение этих матриц дает матрицу для перехода в пространство камеры по формуле (2):

$$M_{cam} = T \times R . \quad (2)$$

Следующим шагом является переход из пространства камеры в пространство отсечения. Для этого необходимо сформировать матрицу отсечения или матрицу проекции. Она вытекает из рассмотрения плоскости, на которую нужно спроецировать изображение. Вычислим ее при программной реализации, так как для этого следует знать, какую двумерную систему координат имеет инструмент для создания окна и отрисовки изображения.

После умножения на матрицу проекции нужно нормализовать координаты вершин. Это реализуется благодаря делению координат вершин на четвертую величину  $W$ . Нормализованные вершины, имеющие в компонентах значения больше единицы, не будут видны, поэтому их следует отсечь.

Нормализованные вершины находятся в нормализованном пространстве отсечения. Чтобы вывести их на экран, нужно умножить их координаты на матрицу преобразования. Ее тоже вычислим исходя из особенностей инструмента и разрешения экрана.

После всех действий будет получен набор двумерных вершин, готовых к выводу на экран [17].

### **3.2 Проектирование движка**

Выберем архитектуру движка. Приложение будет иметь модульную архитектуру. Это необходимо для того, чтобы упростить разработку, а также дальнейшее расширение или изменение программы. Модульная архитектура позволит разделить компоненты приложения на блоки и реализовывать их по отдельности. Для разработки выберем объектно-ориентированный подход, так как он хорошо подходит для реализации модульной архитектуры. Компоненты

программы будут представлены как отдельные объекты с собственными характеристиками и методами.

Так как текстурирование в приложении не нужно, было решено не писать шейдеры и возложить вычисления на процессор. В таком случае выбор языка направлен на упрощение структуры. Для реализации движка выбран язык Python. Он структурно прост и понятен, что даст преимущество в реализации движка и дальнейшем разборе его структуры. Python достаточно медленный, но так как вектор разработки направлен на упрощение структуры приложения, это не играет большой роли. Он также имеет большое количество библиотек нужных для разработки – создание окна, рисование графики, ускорение вычислений и т.д [18].

Для отображения примитивов на экране была выбрана библиотека Pygame. Она имеет достаточный набор функций для создания окна приложения, отрисовки точек, полигонов и вывода необходимой информации. Так как все вычисления будут производиться на языке Python, то в дальнейшем при желании можно спокойно сменить эту библиотеку на любую другую, изменив блок создания окна и заменив функции отображения на функции другой библиотеки [19].

Также использованы несколько библиотек для оптимизации работы кода. Numpy предоставит возможности использовать списки и математические функции, которые инициализируются гораздо быстрее встроенных в язык Python [20]. Библиотека Numba предоставит декораторы для ускорения некоторых функций. [21]

### **3.3 Реализация движка**

Для начала создадим виртуальную среду Python и загрузим все необходимые библиотеки, а именно – Pygame, Numpy и Numba.

Код приложения состоит из 5 файлов:

- main.py – запускной файл, в котором помимо запуска описан класс Engine для окна pygame (приложение А),
- matrix\_function.py – содержит функции, возвращающие матрицы преобразований (приложение Б),
- object\_3d.py – описывает класс Object3D трехмерного объекта, который будет располагаться на сцене. Также опишем дочерний класс Axis для отрисовки систем координат (приложение В),
- camera.py – описывает класс камеры Camera (приложение Г),
- projection.py – описывает класс проекции Projection (приложение Д).

Сначала напишем код для файла main.py, чтобы запустить приложение. В классе Engine установим параметры окна, например, 1600 пикселей в ширину и 900 пикселей в высоту, а также ограничим количество кадров в секунду до 60 кадров:

```
class Engine:
    def __init__(self):
        pg.init()
        self.RES = self.WIDTH, self.HEIGHT = 1600, 900
        self.H_WIDTH, self.H_HEIGHT = self.WIDTH // 2,
self.HEIGHT // 2
        self.FPS = 60
        self.screen = pg.display.set_mode(self.RES)
        self.clock = pg.time.Clock()
```

Функция run содержит цикл, который не позволяет приложению закрыться при отправке на него действия. По сути эта функция запускает приложение. Вместе с ней напишем вне класса условие для запуска приложения:

```
def run(self):
    while True:
```

```

        self.draw()
        self.draw_debug()
        self.camera.control()
        [exit() for i in pg.event.get() if i.type ==
pg.QUIT]

        pg.display.set_caption('Engine')
        pg.display.flip()
        self.clock.tick(self.FPS)

if __name__ == '__main__':
    app = Engine()
    app.run()

```

Чтобы проверить приложение на отзывчивость напишем функцию `draw` и в ней установим цвет фона приложения. Далее эта функция понадобится для отрисовки объектов. Вызываем ее из функции `run` перед проверкой действий:

```

def draw(self):
    self.screen.fill(pg.Color((2, 49, 94)))

```

Для реализации метода будем считать мировую систему координат левосторонней. В файле `function.py` опишем функции, возвращающие матрицы преобразований, которые хранятся в массиве `numpy`. Для будущей отладки будем считать положительное направление оси `OZ` южным, а оси `OX` – западным. Код представлен в приложении.

Далее напишем код для файла `object_3d.py`. В ней опишем класс хранящий необходимые функции для манипуляций над объектом:

```

class Object3D:
    def __init__(self, render, vertices='', faces=''):

```

```

self.render = render
self.vertices = np.array(vertices)
self.faces = faces

def translate(self, pos):
    self.vertices = self.vertices @ translate(pos)

def scale(self, scale_to):
    self.vertices = self.vertices @ scale(scale_to)

def rotate_x(self, angle):
    self.vertices = self.vertices @ rotate_x(angle)

def rotate_y(self, angle):
    self.vertices = self.vertices @ rotate_y(angle)

def rotate_z(self, angle):
    self.vertices = self.vertices @ rotate_z(angle)

```

Так же в нем хранятся данные об объекте – координаты вершин и грани. Для примера создадим куб с длиной ребра 1, началом координат в точке (0, 0, 0) и с противоположной точкой в координатах (1, 1, 1).

Далее напишем код для файла camera.py. В ней опишем класс для камеры. В ней установим характеристики камеры: координаты положения в пространстве, координаты векторов вращения, параметры области видимости:

```

class Camera:
    def __init__(self, render, position):
        self.render = render
        self.position = np.array([*position, 1.0])
        self.forward = np.array([0, 0, 1, 1])
        self.up = np.array([0, 1, 0, 1])
        self.right = np.array([1, 0, 0, 1])

```

```

self.h_fov = math.pi / 3
self.v_fov = self.h_fov * (render.HEIGHT / render.WIDTH)
self.near_plane = 0.1
self.far_plane = 100

```

Также опишем функции для перемещения камеры и функции о положении камеры, которые будут использовать для отладки.

Далее опишем класс для проекции. В него передадим расстояния от камеры до ближней и дальней плоскостей отсечения (переменные NEAR и FAR), а также рассчитаем параметры проецируемого изображения с помощью горизонтальной и вертикальной областей видимости:

```

class Projection:
    def __init__(self, render):
        NEAR = render.camera.near_plane
        FAR = render.camera.far_plane
        RIGHT = math.tan(render.camera.h_fov / 2)
        LEFT = -RIGHT
        TOP = math.tan(render.camera.v_fov / 2)
        BOTTOM = -TOP
        HW, HH = render.H_WIDTH, render.H_HEIGHT

```

Начало экранных координат в библиотеке `pygame` находится в левом верхнем углу окна. Учитывая эту информацию, опишем матрицы проекции и матрицу преобразования в экранные координаты:

```

m00 = 2 / (RIGHT - LEFT)
m11 = 2 / (TOP - BOTTOM)
m22 = (FAR + NEAR) / (FAR - NEAR)
m32 = -2 * NEAR * FAR / (FAR - NEAR)

self.projection_matrix = np.array([

```

```

        [m00, 0, 0, 0],
        [0, m11, 0, 0],
        [0, 0, m22, 1],
        [0, 0, m32, 0]
    ])
    self.to_screen_matrix = np.array([
        [HW, 0, 0, 0],
        [0, -HH, 0, 0],
        [0, 0, 1, 0],
        [HW, HH, 0, 1]
    ])

```

Далее в классе `Object3D` опишем функцию для преобразования трехмерных координат в координаты экрана и отрисовки их на экран при помощи инструментов `pygame`:

```

def screen_projection(self):
    vertices = self.vertices @
self.render.camera.camera_matrix()
    vertices = vertices @
self.render.projection.projection_matrix
    vertices /= vertices[:, -1].reshape(-1, 1)
    vertices[(vertices > 1) | (vertices < -1)] = 0
    vertices = vertices @
self.render.projection.to_screen_matrix
    vertices = vertices[:, :2]

```

В классе `Engine` опишем функцию для создания экземпляров камеры, проекции и объекта и будем вызывать ее при инициализации этого класса. В функции `draw` будем вызывать отрисовку объекта, а в функции `draw_debug` будем выводить данные отладки движка – количество кадров в секунду, положение и ориентацию камеры по сторонам света.

После этого можно запустить программу и увидеть результат – наш куб, координаты которого мы задавали ранее (рисунок 3.14).

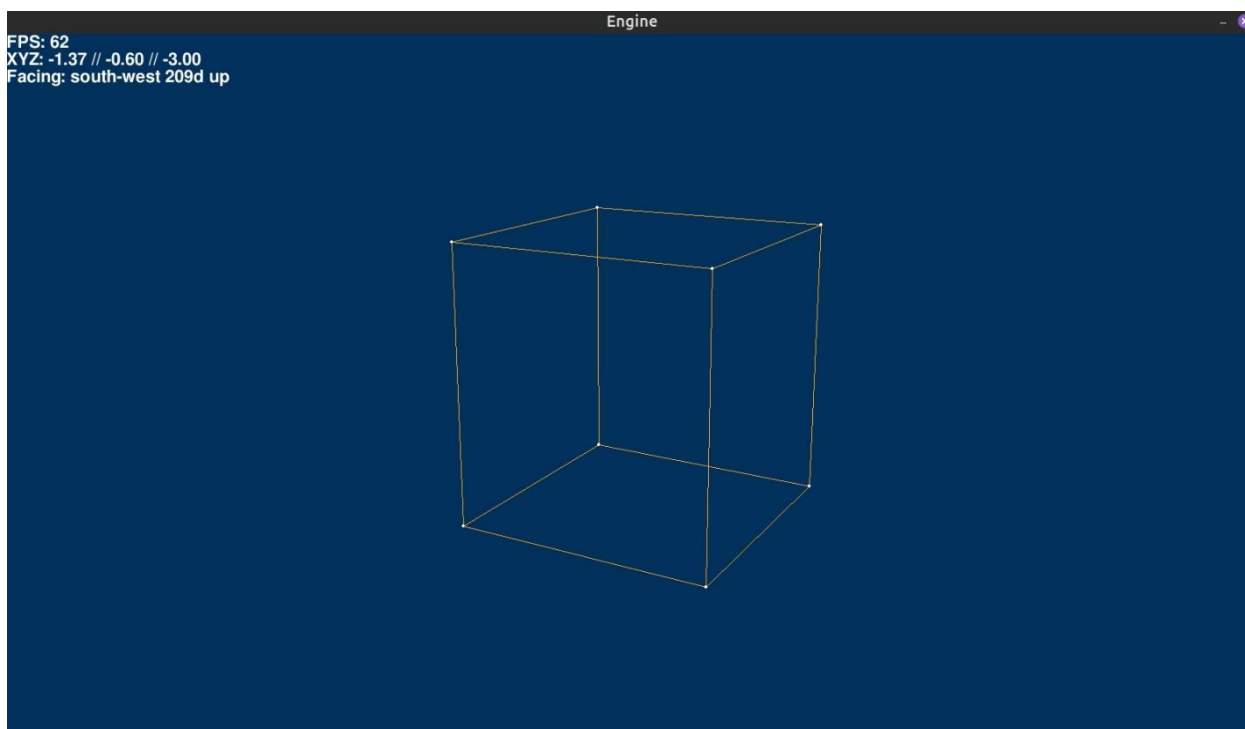


Рисунок 3.14 – Вывод изображения куба

Для перемещения камеры в классе `Camera` опишем функцию `control` и с помощью инструментов библиотеки `rugame` привяжем клавиши:

- W – перемещение вперед,
- S – перемещение назад,
- A – перемещение влево,
- D – перемещение вправо,
- пробел – перемещение прямо вверх,
- левый shift – перемещение прямо вниз,
- стрелки – поворот камеры.

В файле `object_3d.py` дополним класс `Object3D`, присвоив ему флаги для движения, если мы хотим задать объекту движение, и для отрисовки вершин, если мы хотим выбрать отрисовывать их или нет. Отрисовка одних граней

значительно ускорит движок. С той же целью воспользуемся библиотекой `numba` и ускорим медленные функции проверки, например, функцию `any`:

```
@jit(fastmath=True)
def any_func(arr, a, b):
    return np.any((arr == a) | (arr == b))
```

В этом же файле опишем дочерний от `Object3D` класс `Axis`, который будет визуализировать системы координат:

```
class Axis(Object3D):
    def __init__(self, render):
        super().__init__(render)
        self.vertices = np.array([(0, 0, 0, 1), (1, 0, 0, 1),
(0, 1, 0, 1), (0, 0, 1, 1)])
        self.faces = np.array([(0, 1), (0, 2), (0, 3)])
        self.colors = [pg.Color('red'), pg.Color('green'),
pg.Color('blue')]
        self.color_faces = [(color, face) for color, face in
zip(self.colors, self.faces)]
        self.draw_vertices = False
        self.label = 'XYZ'
```

Инициализируем систему координат сцены и локальную систему координат куба. Результат на рисунке 3.15.

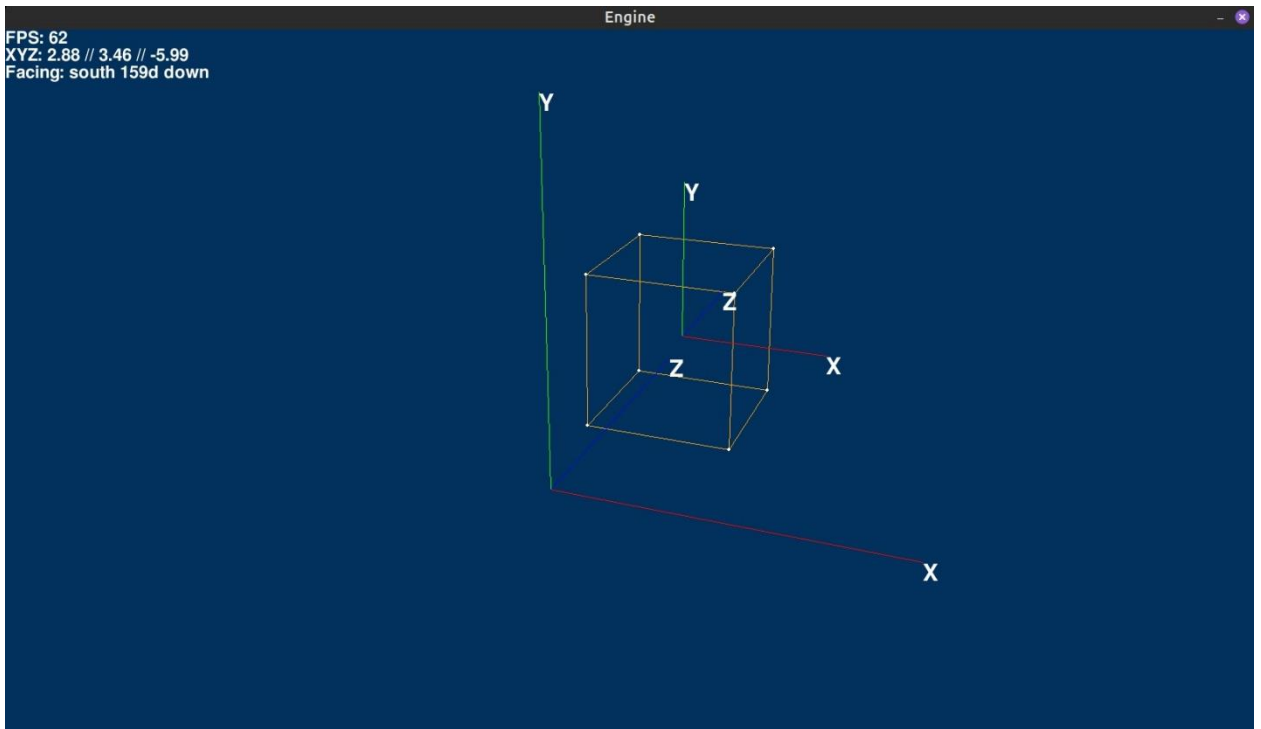


Рисунок 3.15 – Вывод изображения куба и систем координат

Допишем в классе Engine функцию для получения данных об объекте из файла (рисунок 3.16). Для хранения данных о графических трехмерных моделях используется файл с расширением \*.obj.

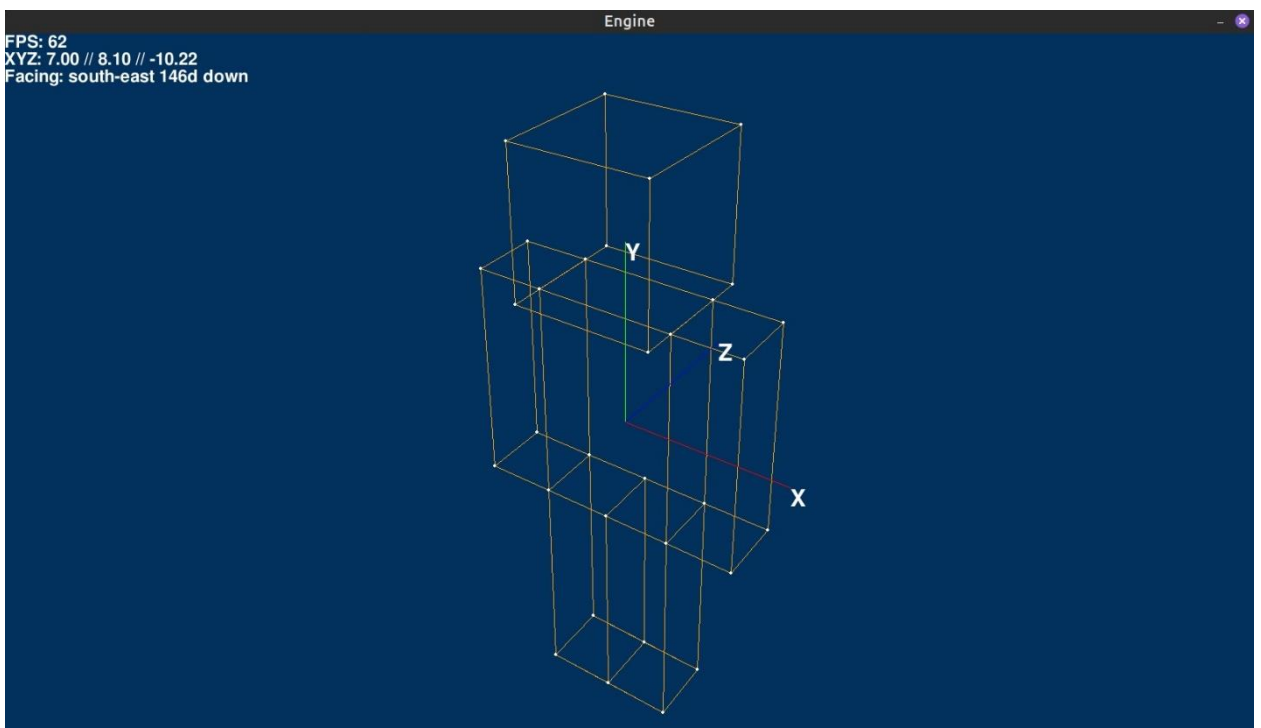


Рисунок 3.16 – Вывод изображения объекта из файла

Также напишем функцию, которая позволит рассчитать координаты точек для отрисовки графика заданной функции. Вычисления будут производиться в некотором диапазоне для каждой оси при помощи метода половинного деления, поэтому необходимо указывать каноническое уравнение функции. Таким образом можно выводить графики функций в трехмерном пространстве. Графики различных функций представлены на рисунках 3.17 – 3.24.

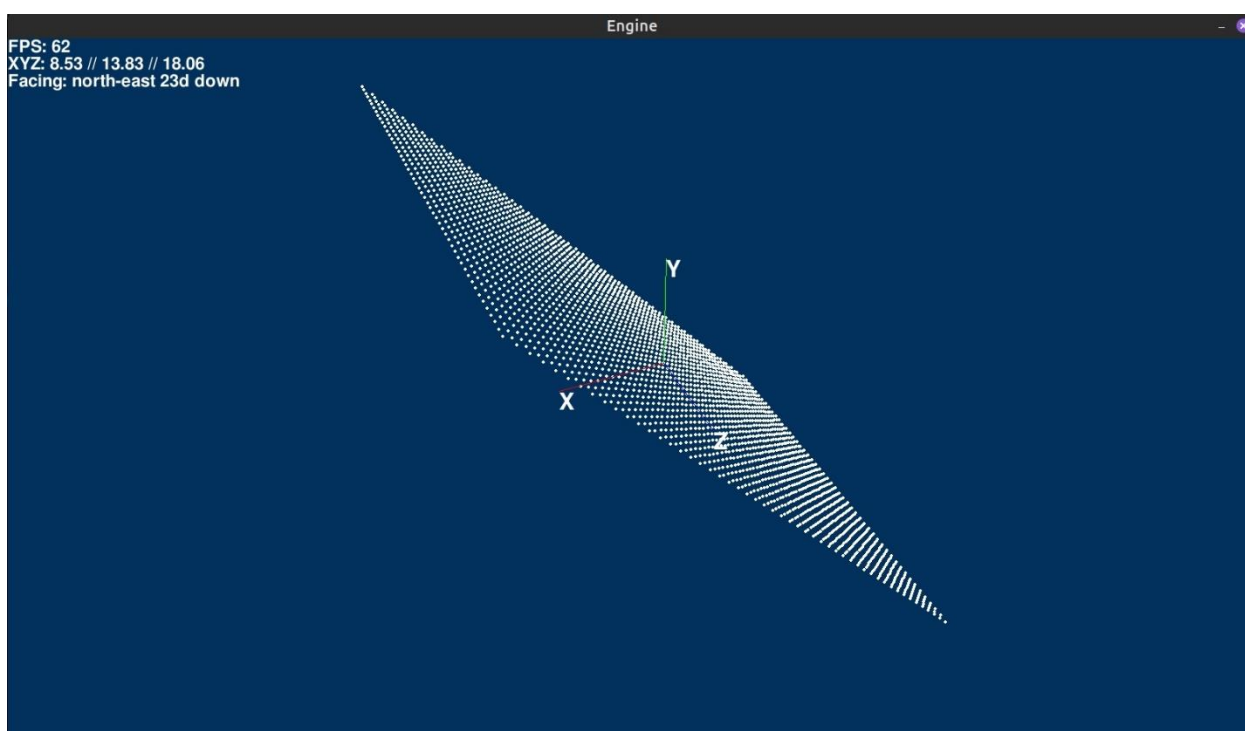


Рисунок 3.17 – Плоскость

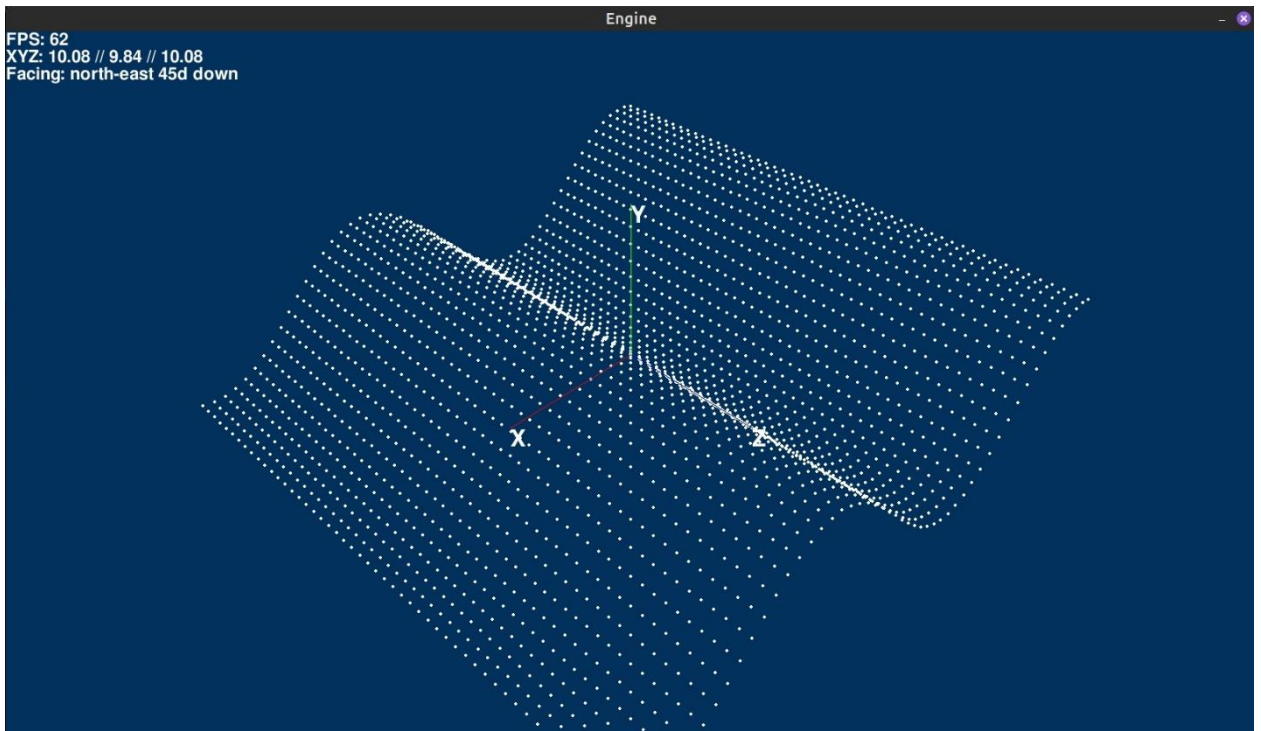


Рисунок 3.18 – График  $\sin(x)$

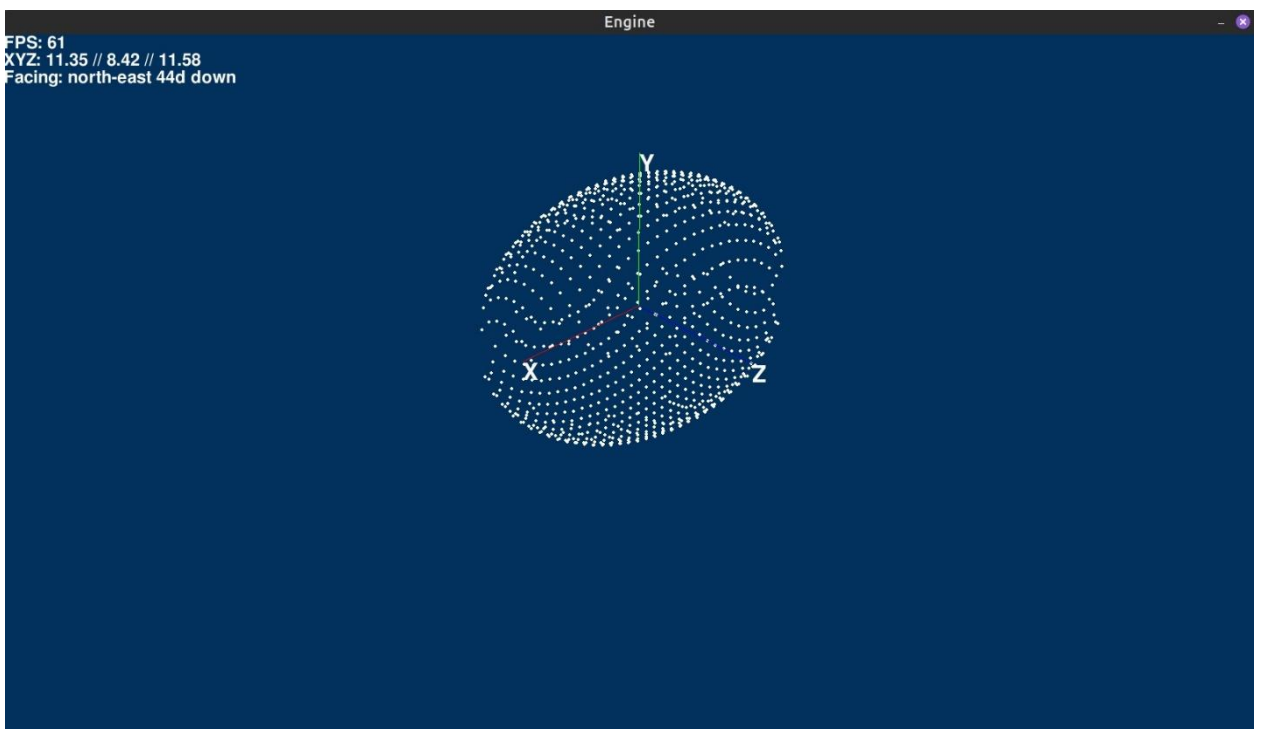


Рисунок 3.19 – Эллипсоид

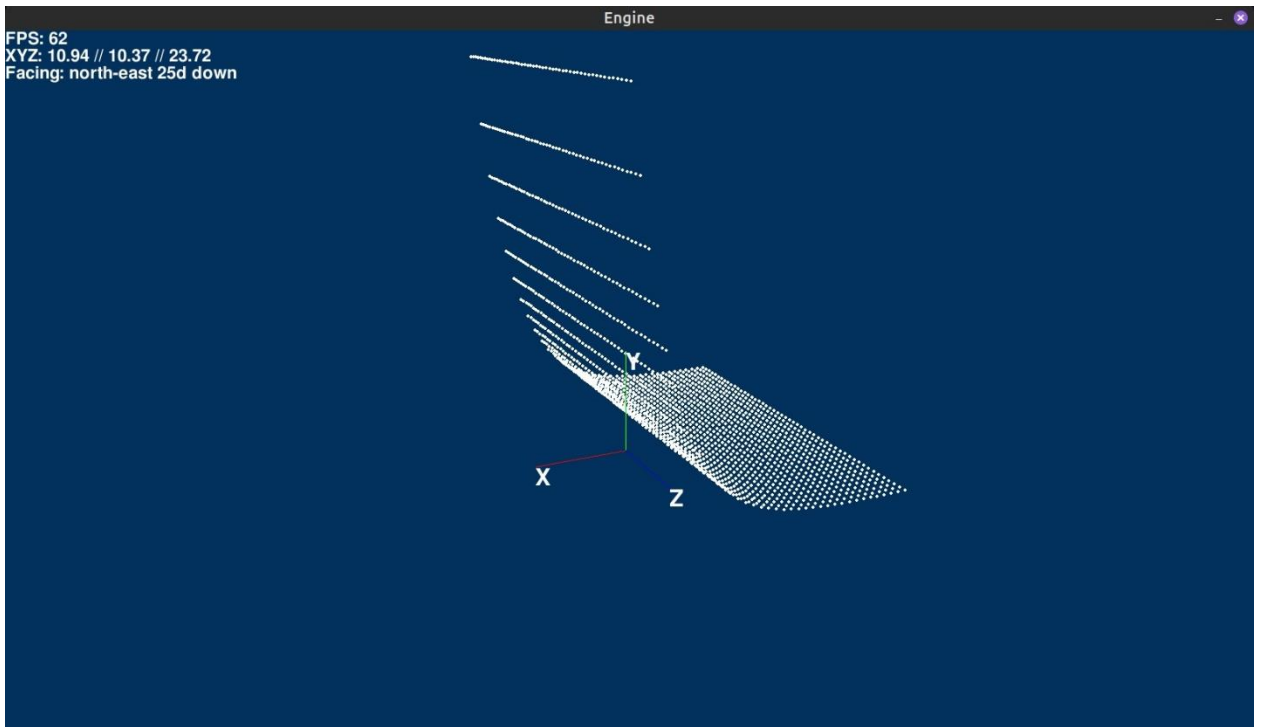


Рисунок 3.20 – График  $exp(x)$

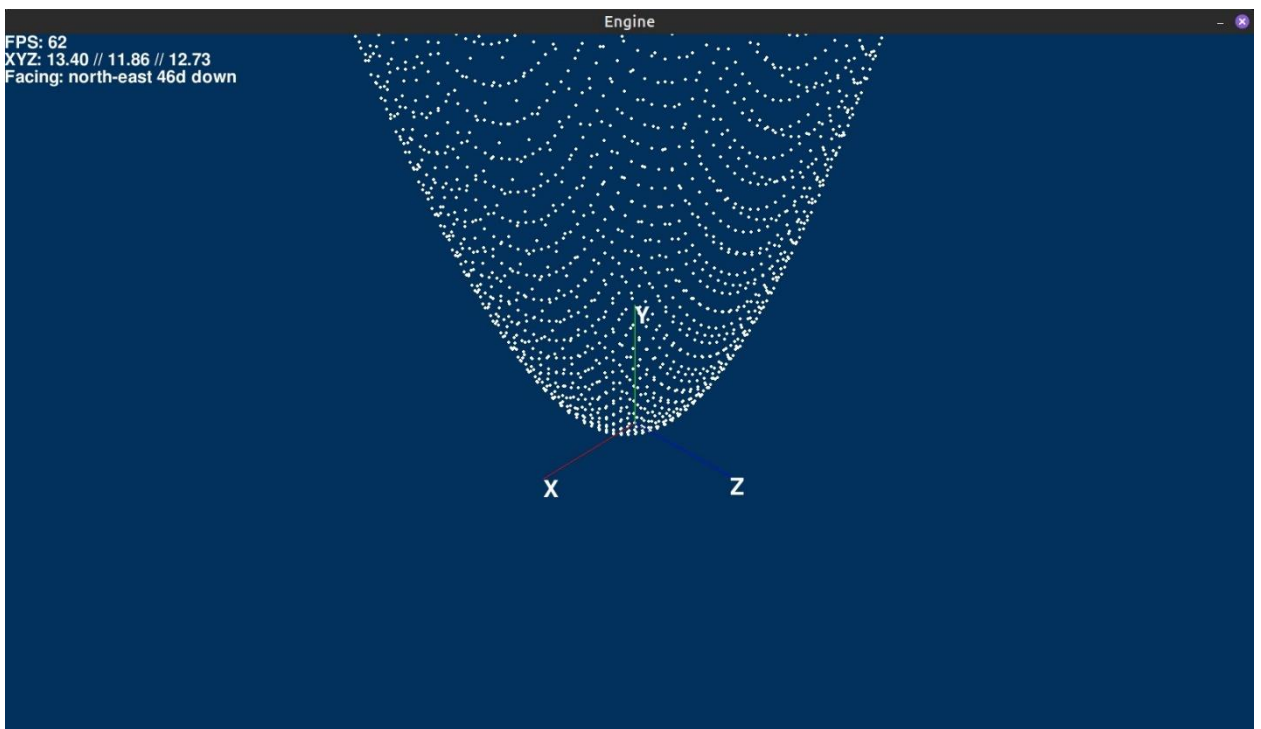


Рисунок 3.21 – Эллиптический параболоид

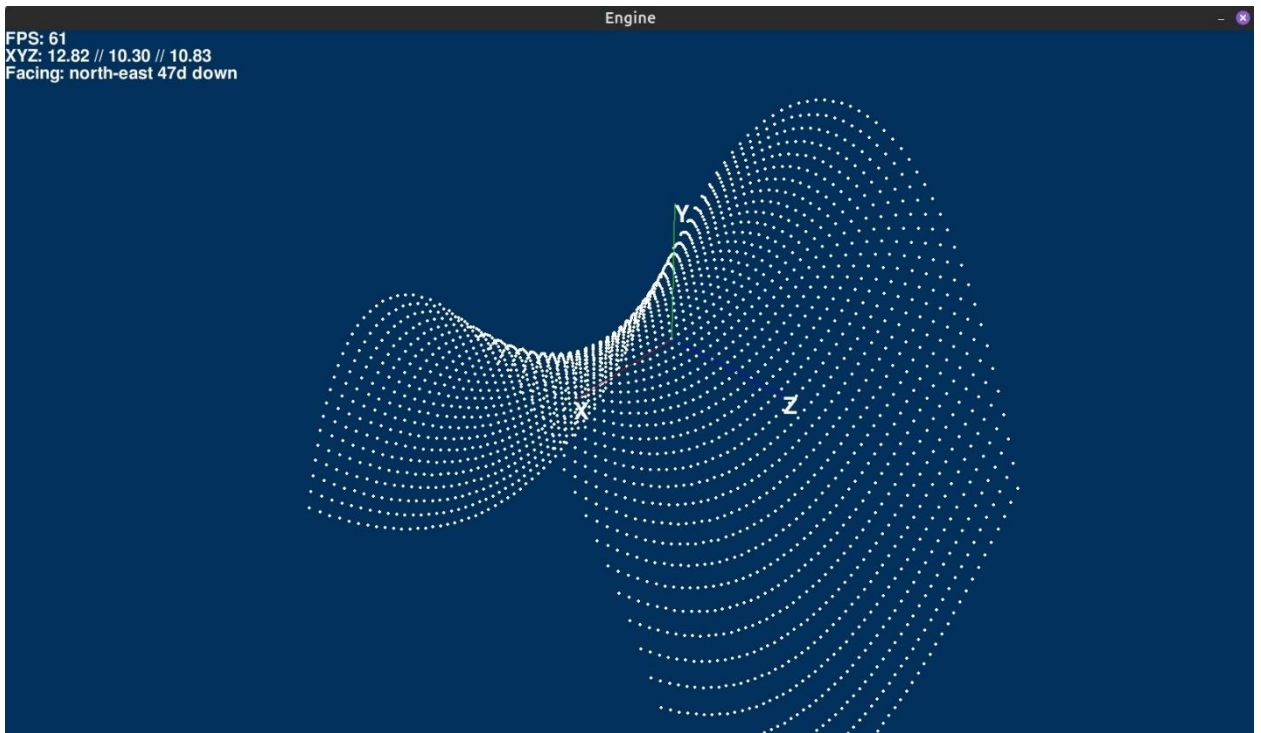


Рисунок 3.22 – Гиперболический параболоид

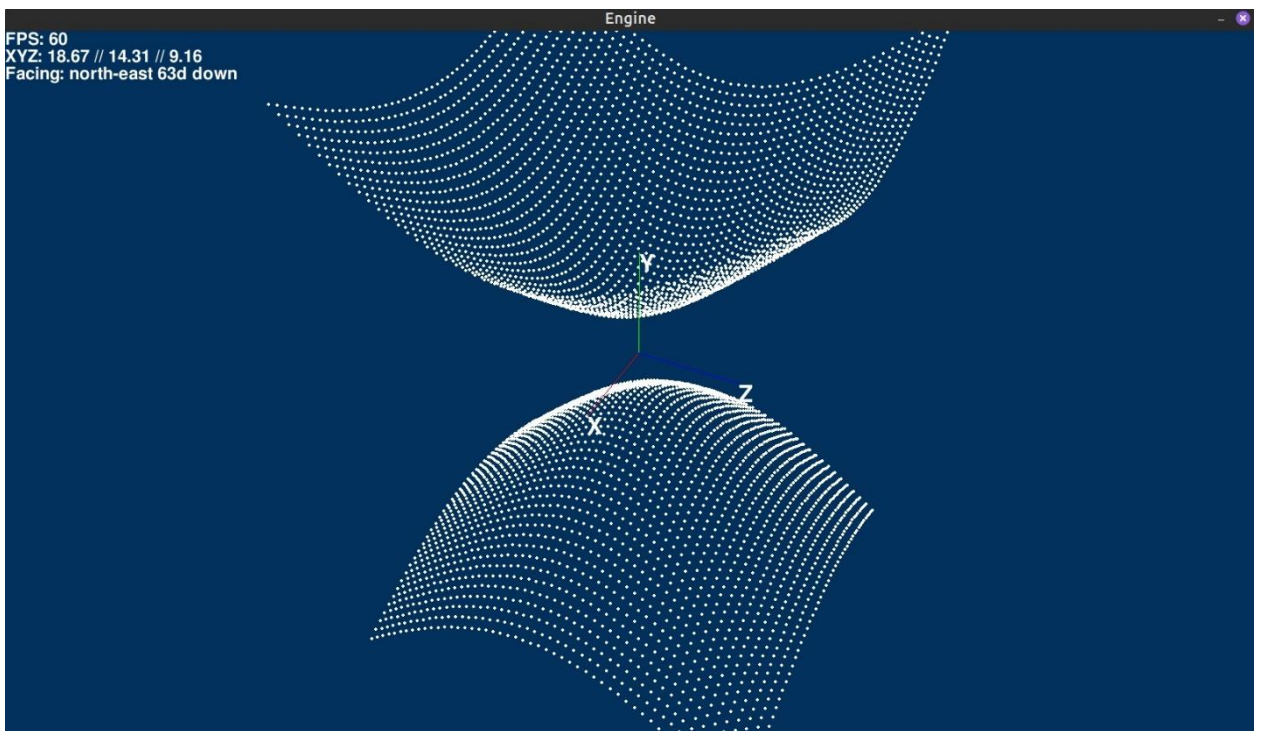


Рисунок 3.23 – Двуполотной гиперboloид

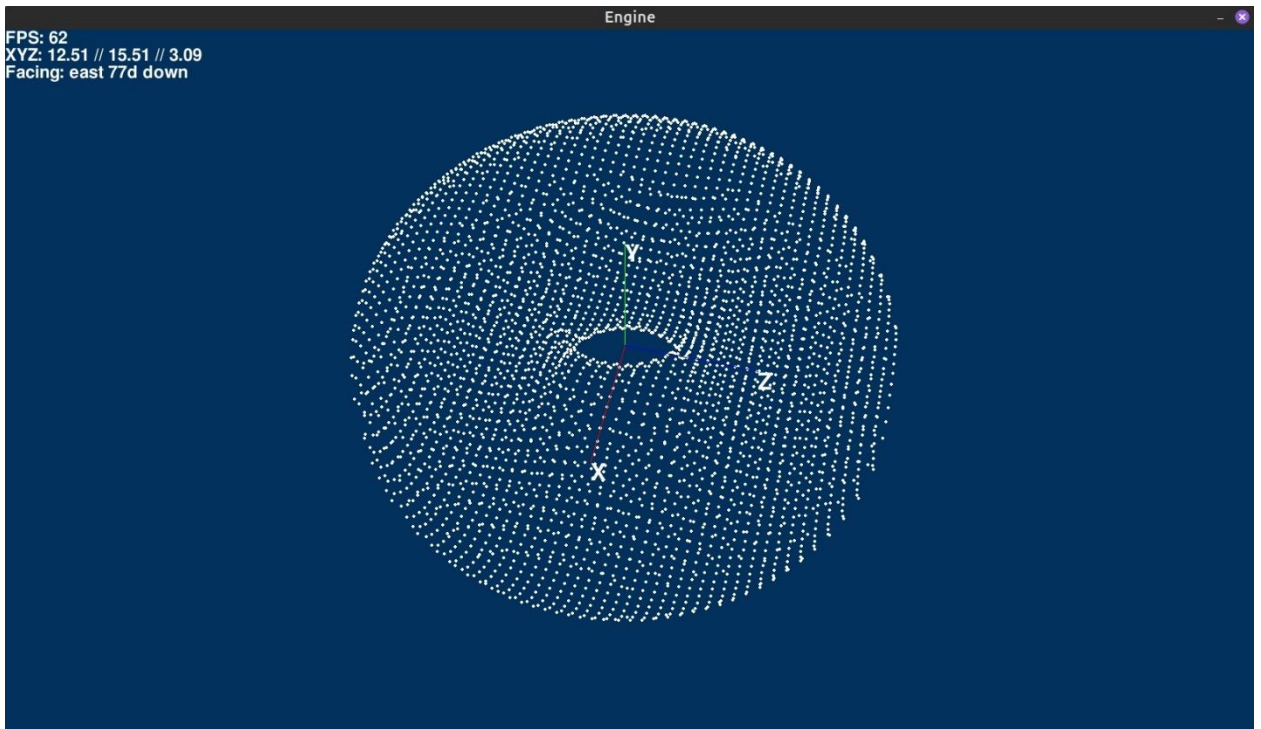


Рисунок 3.24 – Поверхность тора

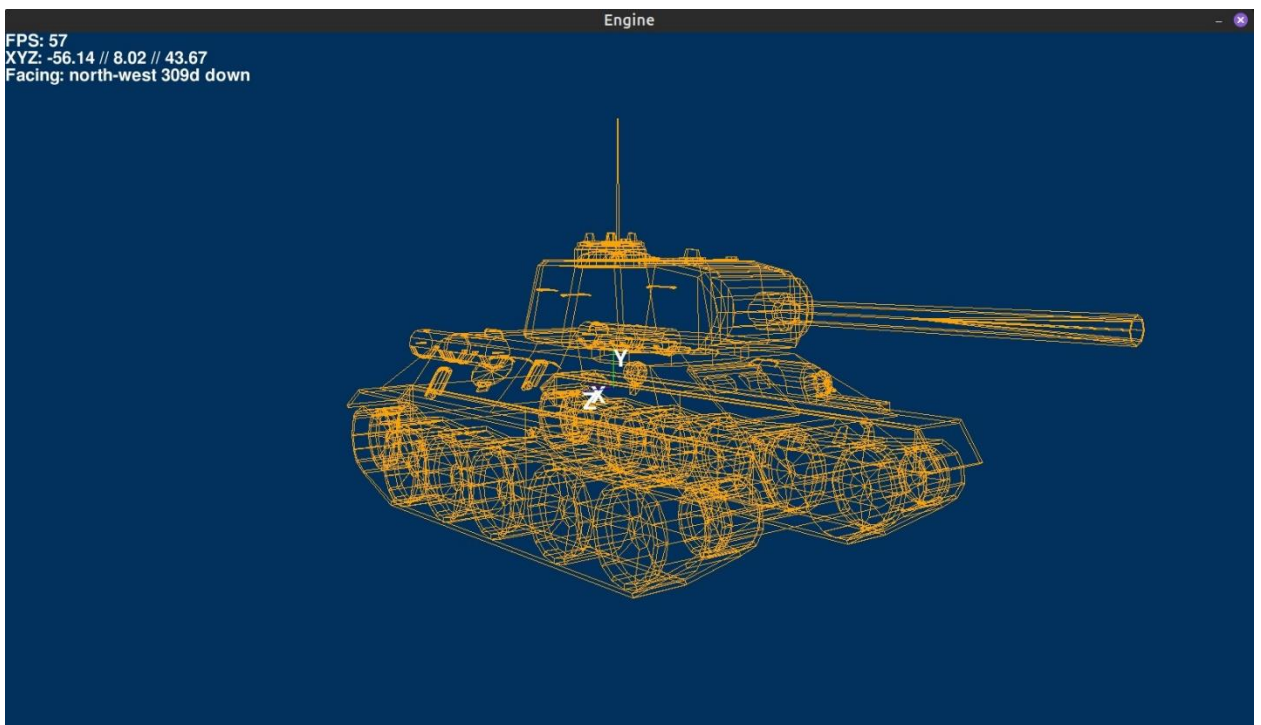


Рисунок 3.25 – Модель танка

Данная программа является самым простым методом для визуализации трехмерных изображений. Эта версия обладает рядом недостатков. Например, необходимость заново запускать приложение при желании изменить объект.

Изображение инвертируется при определенных обстоятельствах, а грани не прорисовываются, если одна из вершин грани выходит за границы экрана. Также хорошим решением является перенос вычислений на видеокарту, так как при достаточно большом количестве вершин программа работает медленнее, как видно на рисунке 3.25. Но даже этого достаточно, чтобы построить график любой функции или же собственного объекта.

## ЗАКЛЮЧЕНИЕ

Данное исследование дает представление многогранности работы с трехмерной графикой. Можно подчеркнуть разнообразие и многообразие программного обеспечения, предназначенного для трехмерного моделирования и визуализации. Рассмотренные в обзоре программы предлагают уникальные возможности и функциональные особенности, и каждая из них может быть оптимальным выбором в зависимости от конкретных потребностей и целей пользователя.

Исследование было подготовительным в программной реализации движка. Список из методов дал представление о многообразии компьютерных алгоритмов и позволил сделать выбор в сторону одного из них. Подробное описание этого метода упростило и ускорило разработку движка. Сформулированные требования создали вектор направления разработки движка, позволили выбрать инструменты и положили начало разработке.

Разработка движка оказалась интересной и увлекательной. Сейчас движок имеет некоторые недостатки, малый набор инструментов, недостаточно оптимизирован и работает на процессоре. Несмотря на это, он может послужить уникальной основой для улучшения и развития собственного программного продукта в области компьютерной графики.

## СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

- 1 Blender : сайт. – URL: <https://www.blender.org> (дата обращения: 20.04.2024). – Режим доступа: свободный. – Текст: электронный.
- 2 Cinema 4D : сайт. – URL: <https://www.maxon.net/en/cinema-4d> (дата обращения: 20.04.2024). – Режим доступа: свободный. – Текст: электронный.
- 3 Coffee++ : сайт. – URL: <http://bixense.com/coffeepp> (дата обращения: 20.04.2024). – Режим доступа: свободный. – Текст: электронный.
- 4 Unity : сайт. – URL: <https://docs.unity3d.com> (дата обращения: 21.04.2024). – Режим доступа: свободный. – Текст: электронный.
- 5 Unreal Engine : сайт. – URL: <https://www.unrealengine.com/en-US> (дата обращения: 21.04.2024). – Режим доступа: свободный. – Текст: электронный.
- 6 Avizo Software: сайт. – URL: <https://www.thermofisher.com/nl/en/home/electron-microscopy/products/software-em-3d-vis/avizo-software.html> (дата обращения: 22.04.2024). – Режим доступа: свободный. – Текст: электронный.
- 7 Autodesk : сайт. – URL: <https://www.autodesk.com/products/autocad/overview> (дата обращения: 22.04.2024). – Режим доступа: свободный. – Текст: электронный.
- 8 AutoLISP : сайт. – URL: <https://www.afralisp.net/autolisp/> (дата обращения: 22.04.2024). – Режим доступа: свободный. – Текст: электронный.
- 9 VisIt Home : сайт. – URL: <https://visit-dav.github.io/visit-website/index.html> (дата обращения: 23.04.2024). – Режим доступа: свободный. – Текст: электронный.
- 10 SketchUp : сайт. – URL: <https://www.sketchup.com/en> (дата обращения: 23.04.2024). – Режим доступа: свободный. – Текст: электронный.
- 11 OpenGL : сайт. – URL: <https://www.opengl.org> (дата обращения: 23.04.2024). – Режим доступа: свободный. – Текст: электронный.
- 12 Евстратов В. В. Создание программы визуализации псевдотрехмерного изображения с помощью рейкастинга / В. В. Евстратов. —

Текст: электронный // Молодой ученый. — 2020. — № 50 (340). — С. 12-15. — URL: <https://moluch.ru/archive/340/76504/> (дата обращения: 30.05.2024).

13 Ray-Casting Tutorial For Game Development And Other Purposes // WebCite : сайт. — URL: <https://www.webcitation.org/66cwwV9q0?url=http://www.permadi.com/tutorial/raycast/index.html> (дата обращения: 17.05.2024). — Режим доступа: свободный. — Текст: электронный.

14 John C. Hart Sphere Tracing: A Geometric Method for the Antialiased Ray Tracing of Implicit Surfaces // The Visual Computer. - 1995. - №12. - С. 70-119. — Текст: непосредственный.

15 Минькова Р.М. Векторная алгебра и аналитическая геометрия: учебно-методическое пособие по курсе "Высшая математика" / Р.М. Минькова. — Екатеринбург : ГОУ ВПО УГТУ–УПИ, 2006. — 42 с. — ISBN 5-321-00547-8. — Текст: непосредственный.

16 Строзотт Т. Нефотореалистичная компьютерная графика: моделирование, рендеринг, анимация / Т. Строзотт, Ш. Шлехтвег. — Иркутск : Кудиц-образ, 2005. — 416 с. — ISBN 5-93378-104-5. — Текст: непосредственный.

17 Порев В. Н. Компьютерная графика / В. Н. Порев. — СПб. : БХВ-Петербург, 2002. — 432 с. — ISBN 5-94157-139-9. — Текст: непосредственный.

18 Python : сайт. — URL: <https://www.python.org> (дата обращения: 26.05.2024). — Режим доступа: свободный. — Текст: электронный.

19 Pygame : сайт. — URL: <https://www.pygame.org> (дата обращения: 26.05.2024). — Режим доступа: свободный. — Текст: электронный.

20 NumPy : сайт. — URL: <https://numpy.org/doc/stable/index.html> (дата обращения: 26.05.2024). — Режим доступа: свободный. — Текст: электронный.

21 Numba : сайт. — URL: <https://numba.pydata.org> (дата обращения: 26.05.2024). — Режим доступа: свободный. — Текст: электронный.

## ПРИЛОЖЕНИЕ А

(обязательное)

### Исходный код файла «main.py»

```
from object_3d import *
from camera import *
from projection import *
import pygame as pg
from test_function import *
from functions import *

class Engine:
    def __init__(self):
        pg.init()
        self.RES = self.WIDTH, self.HEIGHT = 1600, 900
        self.H_WIDTH, self.H_HEIGHT = self.WIDTH // 2,
self.HEIGHT // 2
        self.FPS = 60
        self.MIN_X, self.MAX_X = -5, 5
        self.MIN_Y, self.MAX_Y = -5, 5
        self.MIN_Z, self.MAX_Z = -5, 5
        self.step = 0.2
        self.screen = pg.display.set_mode(self.RES)
        self.clock = pg.time.Clock()
        self.create_objects()
        self.font_size = int(self.HEIGHT * 0.025)
        self.font = pg.font.SysFont('Arial', self.font_size,
bold=True)
    def create_objects(self):
        self.camera = Camera(self, [10, 10, 10])
        self.projection = Projection(self)
        self.object = self.get_object_from_function()
        self.object.movement_flag = False
        self.world_axis = Axis(self)
        self.world_axis.movement_flag = False
```

```

self.world_axis.scale(2.5)
self.camera.camera_pitch(math.pi/5)
self.camera.camera_yaw(-3 * math.pi/4)
def get_object_from_file(self, filename):
    vertex, faces = [], []
    with open(filename) as f:
        for line in f:
            if line.startswith('v '):
                vertex.append([float(i) for i in
line.split()[1:]] + [1])
            elif line.startswith('f'):
                faces_ = line.split()[1:]
                faces.append([int(face_.split('/')[0]) - 1
for face_ in faces_])
    return Object3D(self, vertex, faces)
def get_object_from_function(self, func=function3d()):
    vertices = []
    values_X = points(self.MIN_X, self.MAX_X, self.step)
    values_Y = points(self.MIN_Y, self.MAX_Y, self.step)
    values_Z = points(self.MIN_Z, self.MAX_Z, self.step)
    for x in values_X:
        for z in values_Z:
            y_list = function3d(x=x, y=0, z=z)
            for y in y_list:
                vertices.append((x, y, z, 1))
    return Object3D(self, list(set(vertices)))
def draw(self):
    self.screen.fill(pg.Color((2, 49, 94)))
    self.object.draw()
    self.world_axis.draw()
def draw_debug(self):
    text = self.font.render(f'FPS:
{int(self.clock.get_fps())}', True, pg.Color('white'))

```

## Окончание приложения А

```
        self.screen.blit(text, (0, 0))
        text = self.font.render(self.camera.text_position(),
True, pg.Color('white'))
        self.screen.blit(text, (0, self.font_size))
        text = self.font.render(self.camera.text_facing(), True,
pg.Color('white'))
        self.screen.blit(text, (0, self.font_size * 2))
def run(self):
    while True:
        self.draw()
        self.draw_debug()
        self.camera.control()
        [exit() for i in pg.event.get() if i.type ==
pg.QUIT]

        pg.display.set_caption('Engine')
        pg.display.flip()
        self.clock.tick(self.FPS)

if __name__ == '__main__':
    app = Engine()
    app.run()
```

## ПРИЛОЖЕНИЕ Б

(обязательное)

### Исходный код файла «matrix\_functions.py»

```
import math
import numpy as np
def translate(pos):
    tx, ty, tz = pos
    return np.array([
        [1, 0, 0, 0],
        [0, 1, 0, 0],
        [0, 0, 1, 0],
        [tx, ty, tz, 1]])
def rotate_x(a):
    return np.array([
        [1, 0, 0, 0],
        [0, math.cos(a), math.sin(a), 0],
        [0, -math.sin(a), math.cos(a), 0],
        [0, 0, 0, 1]])
def rotate_y(a):
    return np.array([
        [math.cos(a), 0, -math.sin(a), 0],
        [0, 1, 0, 0],
        [math.sin(a), 0, math.cos(a), 0],
        [0, 0, 0, 1]])
def rotate_z(a):
    return np.array([
        [math.cos(a), math.sin(a), 0, 0],
        [-math.sin(a), math.cos(a), 0, 0],
        [0, 0, 1, 0], [0, 0, 0, 1]])
def scale(n):
    return np.array([
        [n, 0, 0, 0], [0, n, 0, 0],
        [0, 0, n, 0], [0, 0, 0, 1]])
```

## ПРИЛОЖЕНИЕ В

(обязательное)

### Исходный код файла «object\_3d.py»

```
import pygame as pg
from matrix_functions import *
from functions import *
from numba import njit
@njit(fastmath=True)
def any_func(arr, a, b):
    return np.any((arr == a) | (arr == b))
class Object3D:
    def __init__(self, render, vertices='', faces=''):
        self.render = render
        self.vertices = np.array(vertices)
        self.faces = faces
        self.font = pg.font.SysFont('Arial', 30, bold=True)
        self.color_faces = [(pg.Color('orange'), face) for face
in self.faces]
        self.movement_flag, self.draw_vertices = True, True
        self.label = ''
    def draw(self):
        self.screen_projection()
        self.movement()
    def movement(self):
        if self.movement_flag:
            self.rotate_y(-(pg.time.get_ticks() % 0.005))
    def screen_projection(self):
        vertices = self.vertices @
self.render.camera.camera_matrix()
        vertices = vertices @
self.render.projection.projection_matrix
        vertices /= vertices[:, -1].reshape(-1, 1)
        vertices[(vertices > 1) | (vertices < -1)] = 0
```

```
vertices = vertices @
self.render.projection.to_screen_matrix
vertices = vertices[:, :2]
for index, color_face in enumerate(self.color_faces):
    color, face = color_face
    polygon = vertices[face]
    if not any_func(polygon, self.render.H_WIDTH,
self.render.H_HEIGHT):
        pg.draw.polygon(self.render.screen, color,
polygon, 1)
        if self.label:
            text = self.font.render(self.label[index],
True, pg.Color('white'))
            self.render.screen.blit(text, polygon[-1])
    if self.draw_vertices:
        for vertex in vertices:
            if not any_func(vertex, self.render.H_WIDTH,
self.render.H_HEIGHT):
                pg.draw.circle(self.render.screen,
pg.Color('white'), vertex, 2)
def translate(self, pos):
    self.vertices = self.vertices @ translate(pos)
def scale(self, scale_to):
    self.vertices = self.vertices @ scale(scale_to)
def rotate_x(self, angle):
    self.vertices = self.vertices @ rotate_x(angle)
def rotate_y(self, angle):
    self.vertices = self.vertices @ rotate_y(angle)
def rotate_z(self, angle):
    self.vertices = self.vertices @ rotate_z(angle)
def center(self):
    center_of_mass = np.mean(self.vertices, axis=0)
    return center_of_mass[:3] * -1
```

```
class Axis(Object3D):
    def __init__(self, render):
        super().__init__(render)
        self.vertices = np.array([(0, 0, 0, 1), (1, 0, 0, 1),
(0, 1, 0, 1), (0, 0, 1, 1)])
        self.faces = np.array([(0, 1), (0, 2), (0, 3)])
        self.colors = [pg.Color('red'), pg.Color('green'),
pg.Color('blue')]
        self.color_faces = [(color, face) for color, face in
zip(self.colors, self.faces)]
        self.draw_vertices = False
        self.label = 'XYZ'
```

## ПРИЛОЖЕНИЕ Г

(обязательное)

### Исходный код файла «camera.py»

```
import pygame as pg
from matrix_functions import *
class Camera:
    def __init__(self, render, position):
        self.render = render
        self.position = np.array([*position, 1.0])
        self.forward = np.array([0, 0, 1, 1])
        self.up = np.array([0, 1, 0, 1])
        self.right = np.array([1, 0, 0, 1])
        self.h_fov = math.pi / 3
        self.v_fov = self.h_fov * (render.HEIGHT / render.WIDTH)
        self.near_plane = 0.1
        self.far_plane = 100
        self.moving_speed = 0.1
        self.rotation_speed = 0.01
        self.anglePitch = 0
        self.angleYaw = 0
        self.angleRoll = 0
    def control(self):
        key = pg.key.get_pressed()
        if key[pg.K_a]: self.position -= self.right *
self.moving_speed
        if key[pg.K_d]: self.position += self.right *
self.moving_speed
        if key[pg.K_w]: self.position += self.forward *
self.moving_speed
        if key[pg.K_s]: self.position -= self.forward *
self.moving_speed
        if key[pg.K_SPACE]: self.position += self.up *
self.moving_speed
```

```

        if key[pg.K_LSHIFT]: self.position -= self.up *
self.moving_speed
        if key[pg.K_LEFT]: self.camera_yaw(-self.rotation_speed)
        if key[pg.K_RIGHT]: self.camera_yaw(self.rotation_speed)
        if key[pg.K_UP]: self.camera_pitch(-self.rotation_speed)
        if key[pg.K_DOWN]:
self.camera_pitch(self.rotation_speed)
    def camera_yaw(self, angle):
        self.angleYaw += angle
    def camera_pitch(self, angle):
        self.anglePitch += angle
    def axiiIdentity(self):
        self.forward = np.array([0, 0, 1, 1])
        self.up = np.array([0, 1, 0, 1])
        self.right = np.array([1, 0, 0, 1])
    def camera_update_axii(self):
        rotate = rotate_x(self.anglePitch) @
rotate_y(self.angleYaw)
        self.axiiIdentity()
        self.forward = self.forward @ rotate
        self.right = self.right @ rotate
        self.up = self.up @ rotate
    def camera_matrix(self):
        self.camera_update_axii()
        return self.translate_matrix() @ self.rotate_matrix()
    def translate_matrix(self):
        x, y, z, w = self.position
        return np.array([
            [1, 0, 0, 0], [0, 1, 0, 0],
            [0, 0, 1, 0], [-x, -y, -z, 1]])
    def rotate_matrix(self):
        rx, ry, rz, w = self.right
        fx, fy, fz, w = self.forward

```

```

    ux, uy, uz, w = self.up
    return np.array([
        [rx, ux, fx, 0],
        [ry, uy, fy, 0],
        [rz, uz, fz, 0],
        [0, 0, 0, 1]])
def text_position(self):
    position = " // ".join([f"{float(num):.2f}" for num in
self.position[:3]])
    return f"XYZ: {position}"
def text_facing(self):
    facing = self.azimuth()
    return f"Facing: {facing} {self.height()}"
def azimuth(self):
    facing = np.array(self.forward[:3])
    azimuth_rad = math.atan2(facing[0], facing[2])
    azimuth_deg = math.degrees(azimuth_rad)
    azimuth_deg = (180 + azimuth_deg) % 360
    return
f'{self.calculate_cardinal_direction(azimuth_deg)}
{azimuth_deg:.0f}d'
    def calculate_cardinal_direction(self, azimuth_deg):
        cardinal_directions = ['north', 'north-east', 'east',
'south-east', 'south', 'south-west', 'west', 'north-west']
        sector_index = int((azimuth_deg + 22.5) / 45) % 8
        return cardinal_directions[sector_index]
def height(self):
    facing = np.array(self.forward[:3])
    if abs(facing[1]) <= 0.05: return "right"
    elif facing[1] > 0: return "up"
    else: return "down"

```

## ПРИЛОЖЕНИЕ Д

(обязательное)

### Исходный код файла «projection.py»


```
import math
import numpy as np
class Projection:
    def __init__(self, render):
        NEAR = render.camera.near_plane
        FAR = render.camera.far_plane
        RIGHT = math.tan(render.camera.h_fov / 2)
        LEFT = -RIGHT
        TOP = math.tan(render.camera.v_fov / 2)
        BOTTOM = -TOP
        HW, HH = render.H_WIDTH, render.H_HEIGHT
        m00 = 2 / (RIGHT - LEFT)
        m11 = 2 / (TOP - BOTTOM)
        m22 = (FAR + NEAR) / (FAR - NEAR)
        m32 = -2 * NEAR * FAR / (FAR - NEAR)
        self.projection_matrix = np.array([
            [m00, 0, 0, 0],
            [0, m11, 0, 0],
            [0, 0, m22, 1],
            [0, 0, m32, 0]])
        self.to_screen_matrix = np.array([
            [HW, 0, 0, 0],
            [0, -HH, 0, 0],
            [0, 0, 1, 0],
            [HW, HH, 0, 1]])
```

## Заявление о самостоятельном характере выполнения работы

Я, Назаров Кирилл Викторович, обучающийся 4 курса, направления подготовки/специальности 02.03.02 Фундаментальная информатика и информационные технологии, заявляю, что в моей работе на тему «Программная реализация движка для построения 3D-моделей», представленной в Государственную экзаменационную комиссию для публичной защиты, не содержится элементов неправомерных заимствований, а также нет обхода алгоритмов проверки.

Все прямые заимствования из печатных и электронных источников, а также ранее защищённых письменных работ, кандидатских и докторских диссертаций имеют соответствующие ссылки.

Я ознакомлен с действующим в Университете Положением о проверке работ студентов, обучающихся ФГБОУ ВО «МГУ им. Н.П. Огарёва» на наличие заимствований, в соответствии с которым обнаружение неправомерных заимствований, а также обнаружение признаков обхода алгоритмов проверки является основанием для отрицательного отзыва руководителя работы.

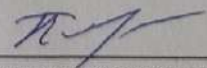
  
подпись обучающегося

« 30 » мая 2024г

Работа представлена для проверки в «Антиплагиат.ВУЗ»

Дата представления работы « 30 » мая 2024г

А.В. Попов

  
подпись руководителя

ОТЧЁТ  
о результатах проверки работы обучающегося  
на наличие заимствований

Ф.И.О. автора работы: Назаров Кирилл Викторович

Тема работы: Программная реализация движка для построения 3D-моделей

Руководитель: Попов Андрей Владимирович, доцент кафедры фундаментальной информатики

Представленная работа прошла проверку на наличие заимствований в системе «Антиплагиат.ВУЗ».

Результаты анализа полного отчета на наличие заимствований:

заимствования: 9,33%

да/нет, количество (%), обоснованность

самоцитирования: нет

да/нет, количество (%), обоснованность

цитирования: 6,68%

да/нет, количество (%), обоснованность

оригинальность: 83,99%

да/нет, количество (%), обоснованность

признаки обхода системы: нет

да/нет, описание

Общее заключение о работе:

Выпускная квалификационная работа К.В. Назарова может быть допущена к защите

Руководитель ВКР,  
Попов Андрей Владимирович  
доцент кафедры фундаментальной информатики



30.05.2024

---

подпись, дата

**Отзыв на бакалаврскую работу**  
**«ПРОГРАММНАЯ РЕАЛИЗАЦИЯ ДВИЖКА ДЛЯ**  
**ПОСТРОЕНИЯ 3D-МОДЕЛЕЙ»**

**студента 4 курса факультета математики и**  
**информационных технологий**

**К.В. Назарова**

Современные технологии моделирования трехмерных объектов используются во многих сферах деятельности человека. Например, без них невозможно представить себе современную медицину, архитектуру, промышленное производство, компьютерные игры и многое другое. Однако создание таких моделей требует специализированного программного обеспечения, которое может быть достаточно дорогим или сложным в использовании. Разработка собственного движка для построения трехмерных моделей может в ряде случаев упростить процесс моделирования и сделать его более доступным для широкого круга пользователей.

В данной бакалаврской работе необходимо было изучить основные методы представления трехмерной графики, провести классификацию популярных графических движков и приложений, разработать собственный движок с модульной архитектурой для построений трехмерных моделей.

В целом бакалаврская работа соответствует заявленной теме и раскрывает исследуемые технологии. В ходе выполнения выпускной работы К.В. Назаров показал себя грамотным специалистом, способным самостоятельно формулировать и решать поставленные задачи.

Считаю, что бакалаврская работа К.В. Назарова заслуживает оценки "отлично".

Научный руководитель бакалаврской работы  
канд. физ. – мат. наук, доцент кафедры  
фундаментальной информатики



А.В. Попов

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ  
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ  
“НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ  
МОРДОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИМ. Н.П. ОГАРЁВА”

ОТЗЫВ РЕЦЕНЗЕНТА  
О ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЕ

Студента Назарова Кирилла Викторовича

Факультет математики и информационных технологий

Кафедра фундаментальной информатики \_\_\_\_\_ Группа 402

Направление подготовки 02.03.02 Фундаментальная информатика и информационные технологии

Квалификация (степень) бакалавр

Наименование темы «Программная реализация движка для построения 3D-моделей»

Рецензент Сухарев Л.А., доцент кафедры математического анализа, алгебры и геометрии ФМиИТ МГУ им. Н.П. Огарёва, кандидат физико-математических наук

**ОЦЕНКА ВЫПУСКНОЙ РАБОТЫ**

№ п/п	Показатели	Оценка				
		5	4	3	2	0*
1.	Актуальность тематики работы		+			
2.	Степень полноты обзора состояния вопроса и корректность постановки задачи	+				
3.	Уровень и корректность использования в работе методов исследования, математического моделирования	+				
4.	Степень комплексности работы, применение в ней знаний естественнонаучных, социально-экономических, общепрофессиональных и специальных дисциплин		+			
5.	Ясность, четкость, последовательность и обоснованность изложения	+				
6.	Применение современного математического и программного обеспечения, компьютерных технологий в работе	+				
7.	Качество оформления (общий уровень грамотности, стиль изложения, качество иллюстраций, соответствие требованиям стандарта)	+				
8.	Оригинальность и новизна полученных результатов (научных, конструкторских и технологических решений)		+			
9.	Тип работы					
	фундаментальная с оригинальными результатами					
	реферативная					
	прикладная	+				
10.	Рекомендации					
	к опубликованию					
	к внедрению	+				
<b>ИТОГОВАЯ ОЦЕНКА</b>		<b>ОТЛИЧНО</b>				

\* – не оценивается (трудно оценить)

**Отмеченные достоинства:**

- 1) В бакалаврской работе Назарова К.В. рассматриваются математические методы компьютерного представления трехмерной графики. В задачу автора входила разработка модульного программного движка для визуализации графических примитивов на основе этих методов.
- 2) Полученные результаты имеют практическое значение, могут использоваться в программных комплексах для визуализации трехмерных моделей.

**Отмеченные недостатки:**

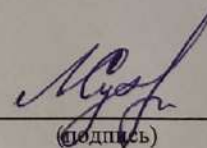
- 1) В разработанном движке реализована только самая базовая функциональность.
- 2) Изложение не всегда строго обосновано и последовательно.

**Заключение:**

Не взирая на отмеченные недостатки, выпускная бакалаврская работа Назарова Кирилла Викторовича удовлетворяет всем требованиям, предъявляемым к выпускным квалификационным работам, и заслуживает оценки «отлично».

13 июня 2024 г.

Рецензент

  
(подпись)

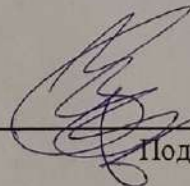
Заведующему кафедрой  
фундаментальной информатики  
А. Г. Смольянову  
студента 4 курса  
очной формы обучения  
на бесплатной основе  
направления подготовки  
02.03.02 – Фундаментальная информатика  
и информационные технологии  
факультета математики и  
информационных технологий  
ФГБОУ ВО «МГУ им. Н.П. Огарёва»  
Назарова Кирилла Викторовича

заявление.

Прошу разместить мою выпускную квалификационную работу на тему  
«Программная реализация движка для построения 3D-моделей» в электронной  
библиотечной системе университета в полном объеме.

17.06.2024

Дата



Подпись