Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н. Э. Баумана (национальный исследовательский университет)»
(МГТУ им. Н. Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К КУРСОВОЙ РАБОТЕ

НА ТЕМУ:

Разработка программного обеспечения для моделирования упругих столкновений объектов в пространстве.

Студент ИУ7-54Б _____ К. А. Рунов
(Группа) (Подпись, дата) (И. О. Фамилия)

Руководитель курсовой работы _____ А. А. Павельев
(Подпись, дата) (И. О. Фамилия)

2023 г.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

На сегодняшний день компьютерная графика является неотъемлемой частью нашей жизни и используется повсеместно. Ей находится применение в самых разных областях человеческой деятельности: она используется в науке, в бизнесе, в кино, играх и везде, где нужно визуальное представление информации на электронном дисплее [1].

Перед разработчиками графического программного обеспечения часто стоит задача синтеза реалистического изображения. Для решения этой задачи существует множество алгоритмов, но, как правило, алгоритмы, дающие наилучшие результаты являются наиболее трудозатратными как по объёму требуемой памяти, так и по количеству требуемых вычислений, потому что учитывают множество световых явлений (дифракция, интерференция, преломление, поглощение, множественное отражение). Поэтому, в зависимости от задачи и от имеющихся вычислительных мощностей, программистам приходится выбирать наиболее целесообразные в их случае алгоритмы и жертвовать либо временем генерации кадра, либо его реалистичностью.

Но генерация реалистического изображения — это лишь одна из многих задач, которые стоят перед программистами графических приложений. При разработке игр или приложений для моделирования физики твёрдого тела возникает задача обнаружения коллизий (столкновений) между объектами виртуального пространства и реагирования на них, например, разрушение объектов в результате столкновения, деформация объектов, или их упругое соударение.

В данной курсовой работе было решено создать программу — песочницу, в которой пользователь сможет размещать объекты в виртуальном пространстве, изменять их свойства (такие как размер, местоположение, цвет, масса), задавать им начальные скорости; после чего наблюдать их перемещение и столкновения.

Цель работы — разработка программы для моделирования упругих столкновений объектов в пространстве.

Для достижения поставленной цели требуется решить следующие задачи:

— описать свойства объекта, которыми он должен обладать для моделирования его движения и столкновения с другими объектами;

— проанализировать существующие способы представления объектов и обосновать выбор наиболее подходящего для решения поставленной задачи;

— проанализировать существующие алгоритмы обнаружения коллизий и обосновать выбор тех из них, которые в наибольшей степени подходят для решения поставленной задачи;

— проанализировать существующие модели освещения и обосновать выбор модели, наиболее подходящей для решения поставленной задачи;

— реализовать выбранные алгоритмы;

— разработать программное обеспечение для решения поставленной задачи;

— провести анализ производительности работы программы в зависимости от количества объектов на сцене и их типов.

1 Аналитическая часть

В данной части под «сценой» понимается виртуальное пространство, в котором расположены объекты, предназначенные для визуализации.

1.1 Описание объектов сцены

В данном разделе будут описаны свойства объектов, которыми он должен обладать для моделирования его движения и столкновения с другими объектами.

Для визуализации объекта, он должен содержать следующую информацию:

— геометрическая информация, какую форму имеет объект;

— информация о местоположении в пространстве (с учётом поворота и масштабирования);

— информация о цвете и/или текстуре объекта.

Для моделирования движения и столкновения объектов, объекты должны содержать следующую информацию:

— информация о «коллайдере» (как правило, упрощённая форма исходного объекта, которая используется при обнаружении столкновений);

— информация о физических свойствах объекта (скорость, ускорение, масса).

1.2 Способы представления объектов сцены

Далее будут рассмотрены возможные способы представления объектов сцены.

1.2.1 Каркасная модель

Объект представляется с помощью его вершин и рёбер, без каких-либо поверхностей [2].

Характеристики каркасной модели:

— простота;

— позволяет получить базовое представление о форме объекта;

— быстрая визуализация;

— не позволяет производить реалистическое освещение, для которого требуется информация о гранях объекта;

— не содержит информации, нужной для обнаружения столкновений.

1.2.2 Поверхностная модель

Объект представляется аппроксимированно, в виде набора поверхностей. Набор поверхностей можно задать как аналитически (уравнением или системой уравнений), так и в виде полигональной сетки [2]. Часто бывает проще задать поверхности в виде полигональной сетки: набора граней и набора вершин, из которых грани состоят.

Характеристики поверхностной модели:

— простота;

— возможность учёта освещения;

— возможность достижения высокого уровня реализма;

— содержит информацию о поверхностях, которая нужна при обнаружении столкновений объектов;

— не математически точное представление, аппроксимация.

1.2.3 Твердотельная модель

Существует несколько методов представления твердотельных моделей: метод констркутивного представления (англ. Constructive representation, сокращённо C – rep) и метод граничного представления (англ. Boundary representation, сокращённо B – rep) [2, 3]. Оба метода предоставляют наиболее полное описание объекта, включая его внешнюю форму и внутреннюю структуру.

Характеристики твердотельной модели:

— математически точное представление;

— наиболее полное описание структуры объекта.

— сложность;

— требовательность к памяти;

— содержит излишнюю информацию, которая не будет использована при обнаружении столкновений объектов.

Выбор представления объектов сцены

На основе проведённого анализа различных способов представления объектов сцены, была выбрана поверхностная модель, так как она обладает всей информацией, нужной для обнаружения столкновений объектов и учёта освещения, а также является менее требовательной к памяти по сравнению с твёрдотельной моделью.

1.3 Алгоритмы обнаружения коллизий

В данном разделе будут проанализированы алгоритмы обнаружения коллизий и выбраны те из них, которые будут использоваться в разработанной программе.

1.3.1 Алгоритм обнаружения коллизий сферы относительно сферы

Алгоритм обнаружения коллизий сферы относительно сферы очень прост: две сферы пересекаются, если длина вектора, проведённого из центра одной сферы к центру другой, будет меньше суммы радиусов этих сфер [4].

Два объекта считаются «столкнувшимися», если пересекаются их сферические оболочки.

Таким образом, алгоритм будет достаточно точно обнаруживать коллизии сферообразных объектов, но менее точно — для объектов, сильно отличающихся от сфер, например, для длинных тонких объектов.

1.3.2 Алгоритм AABB

Согласно алгоритму ограничивающего прямоугольного параллелепипеда, выровненного по осям (англ. Axis Aligned Bounding Box, сокращённо AABB), объекты заключаются в оболочки из прямоугольных параллелепипедов, чьи рёбра параллельны осям координат, после чего применяется несколько последовательных тестов для определения, пересекаются эти оболочки или нет [4].

1.3.3 Алгоритм OBB

Согласно алгоритму ориентированного ограничивающего параллелепипеда (англ. Oriented Bounding Box, сокращённо OBB), объекты заключаются в оболочки из прямоугольных параллелепипедов, чья ориентация совпадает с ориентацией самих объектов, после чего проверяется, пересекаются эти оболочки или нет [4].

Его преимущество относительно алгоритма AABB заключается в том, что при повороте объекта в пространстве, поворачивается также и его OBB — оболочка, в то время как AABB — оболочку надо пересчитывать заново.

Однако проверку пересечения AABB — оболочек выполнять гораздо быстрее и проще, чем проверку пересечения OBB — оболочек.

1.3.4 Алгоритм GJK

Алгоритм Гильберта — Джонсона — Кирти (англ. Gilbert — Johnson — Keerthi, сокращённо GJK) позволяет обнаруживать пересечения любых выпуклых многогранников. В алгоритме используется геометрическая операция под названием «сумма Минковского» (иногда ошибочно называемая разностью Минковского). Для двух множеств точек $A, B \subset \mathbb{R}^3$ сумма Минковского определяется как:

$$A \oplus B = \{\boldsymbol{a} + \boldsymbol{b} : \boldsymbol{a} \in A, \boldsymbol{b} \in B\},$$

где, для векторов $\boldsymbol{a} = (a_x, a_y, a_z)$ и $\boldsymbol{b} = (b_x, b_y, b_z)$, сумма определена, как

$$\boldsymbol{a} + \boldsymbol{b} := (a_x + b_x, a_y + b_y, a_z + b_z).$$

У суммы Минковского есть несколько полезных свойств, которые используются в алгоритме [5].

1) Сумма Минковского двух выпуклых многогранников есть выпуклый многогранник.

2) Если два выпуклых многогранника P и Q пересекаются, то центр координат находится внутри выпуклой оболочки $P \oplus \{-\boldsymbol{q} : \boldsymbol{q} \in Q\}$.

3) Если найдётся хотя бы одно множество точек $S \subseteq P \oplus \{-\boldsymbol{q} : \boldsymbol{q} \in Q\}$, которое заключает в себе центр координат, то и $P \oplus \{-\boldsymbol{q} : \boldsymbol{q} \in Q\}$ заключает в себе центр координат.

Алгоритм заключается в поиске многогранника, составленного из точек суммы Минковского, который содержит в себе центр координат [6]. Также в алгоритме применяются определённые методы, позволяющие считать не всю сумму Минковского целиком, а только её часть.

Выбор алгоритмов обнаружения коллизий

Ниже приведена сравнительная таблица алгоритмов обнаружения коллизий.

Таблица 1 – Сравнение алгоритмов обнаружения коллизий

	Алгоритм обнаружения коллизий сферы относительно сферы	Алгоритм AABB	Алгоритм OBB	Алгоритм GJK
Вычислительная нагрузка	Низкая	Низкая	Средняя	Высокая
Точность обнаружения коллизий у сложных объектов	Низкая	Низкая	Средняя	Высокая
Сложность реализации	Низкая	Низкая	Средняя	Высокая

На основе проведённого анализа алгоритмов обнаружения коллизий, для реализации в программе был выбран алгоритм AABB, по следующим причинам:

— простота реализации;

— достаточность для моделирования упругих столкновений объектов в пространстве;

— алгоритм часто используется для обнаружения коллизий на ранней стадии, и впоследствии его можно будет использовать в паре с GJK для более точного обнаружения коллизий.

1.4 Модели освещения

В данном разделе будут описаны три модели освещения и выбрана одна из них для дальнейшей реализации в программе. Под источником света далее понимается направленный источник света, солнце. Вектор направления света задан и одинаков для всех точек сцены.

1.4.1 Простая модель освещения

В простой модели освещения интенсивность света, отражённого гранью объекта зависит от угла между нормалью к грани и направлением источника света [7]. Таким образом, с максимальной интенсивностью будут освещены грани, нормали к которым противонаправлены вектору направления источника света, а с минимальной интенсивностью будут освещены грани, нормали к которым составляют прямой или острый угол с вектором направления источника света.

1.4.2 Модель освещения Гуро

В модели освещения Гуро интенсивность света, отражённого гранью объекта может быть неоднородной. Интенсивность отражённого света зависит от угла между нормалью к вершине и направлением источника света. Для вычисления значения интенсивности отражённого света в каждой точке грани проводится интерполяция значений интенсивности отражённого света каждой вершиной, принадлежащей этой грани [7].

1.4.3 Модель освещения Фонга

В модели освещения Фонга интенсивность света, отражённого гранью объекта также может быть неоднородной. Интенсивность отражённого света зависит от угла между нормалью к вершине и направлением источника света. Для вычисления значения интенсивность отражённого света в каждой точке грани проводится интерполяция нормалей каждой вершины, принадлежащей этой грани, после чего интенсивность вычисляется в зависимости от угла между интерполированной нормалью и вектором направления источника света [7].

Выбор модели освещения

Ниже приведена сравнительная таблица моделей освещения.

Таблица 2 – Сравнение моделей освещения

	Простая модель освещения	Модель освещения Гуро	Модель Фонга
Реалистичность изображения	Низкая	Средняя	Высокая
Вычислительная нагрузка	Низкая	Средняя	Высокая
Сложность реализации	Низкая	Средняя	Средняя

Несмотря на большие вычислительные затраты и относительную сложность реализации, закраска по Фонгу позволяет достичь более реалистичного изображения, в связи с чем для дальнейшей реализации была выбрана именно она.

1.5 Вывод

В данной части были описаны объекты сцены, проанализированы способы представления объектов сцены, алгоритмы обнаружения коллизий, модели освещения; было принято решение использовать поверхностную модель, модель освещения Фонга и реализовать алгоритм AABB обнаружения коллизий.

2 Конструкторская часть

В данной части будут приведены требования к программному обеспечению, на формальном языке будут описаны алгоритмы, которые будут реализованы при разработке программного обеспечения, а также будет обоснован выбор типов и структур, которые будут использованы при разработке.

2.1 Требования к программному обеспечению

Разрабатываемое программное обеспечение должно предоставлять пользователю следующую функциональность:

— добавление объекта на сцену (куб, сфера, чайник);

— выбор объекта сцены с помощью клавиатуры;

— изменение цвета выбранного объекта;

— изменение геометрических свойств выбранного объекта (положение в пространстве, поворот, увеличение);

— изменение физических свойств выбранного объекта (масса, скорость, ускорение, сила);

— изменение значения гравитации;

— перемещение и поворот камеры с помощью клавиатуры и мыши.

При этом разрабатываемая программа должна удовлетворять следующим требованиям:

— программа должна генерировать кадр не менее, чем за $\frac{1}{30}$ секунды при сцене, состоящей не более, чем из 10 объектов и размере окна 1920x1080 пикселей;

— никакие действия пользователя не должны приводить к аварийному завершению программы.

2.2 Разработка алгоритмов

Далее на формальном языке будут описаны алгоритмы, которые будут реализованы при разработке проргаммного обеспечения.

2.2.1 Общий алгоритм работы программы

Далее представлен общий алгоритм работы разрабатываемого программного обеспечения.

1) Инициализировать используемые объекты (окно, камера, сцена, объекты сцены, графический интерфейс, шейдерная программа).
2) Пока приложение запущено:
   - обработать события от мыши и клавиатуры;
   - обнаружить и разрешить коллизии;
   - обновить местоположения объектов сцены;
   - сгенерировать и отобразить кадр.
3) Освободить ресурсы.

2.2.2 Алгоритм AABB

На рисунке 1 представлена схема алгоритма AABB обнаружения коллизий.

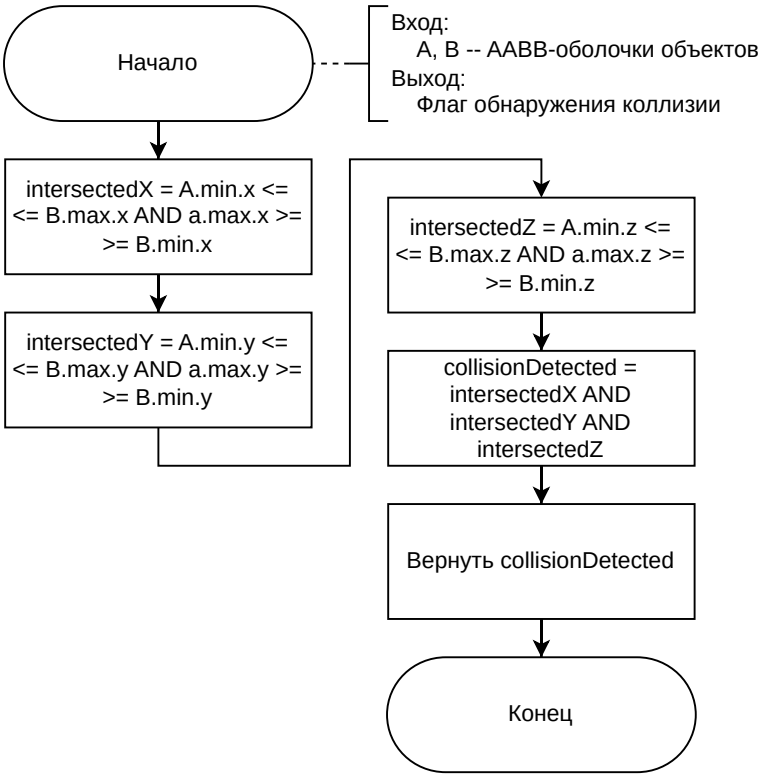

Рисунок 1 – Алгоритм AABB обнаружения коллизий

2.2.3 Модель освещения Фонга

В модели освещения Фонга [8, 9, 10] учитываются три составляющих отражённого света:

1) рассеянная,
2) фоновая,
3) зеркальная.

Рассеянный свет

Рассеянный свет является светом, отражаемым во всех направлениях, и его интенсивность во всех направлениях постоянна, следовательно, для его расчёта не требуется учитывать направление взгляда камеры, но должен быть известен угол между направлением света и нормалью к поверхности объекта [9].

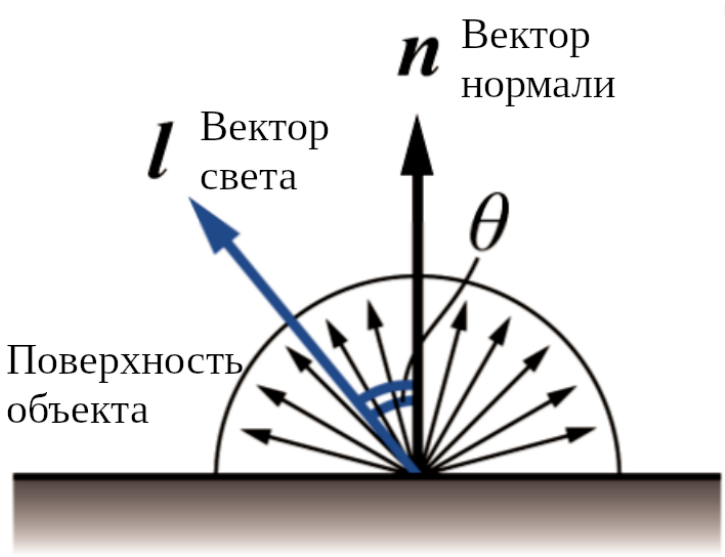

Рисунок 2 – Модель Фонга: Рассеянная составляющая света (Источник: [9])

Рассеянная составляющая света, согласно [9], рассчитывается по формуле 1, приведённой ниже.

$$I_d = k_d I_l \cos\theta = k_d I_l \left|\boldsymbol{n} \cdot \boldsymbol{l}\right|, \tag{1}$$

где

— I_l — интенсивность источника света,

— k_d — коэффициент рассеивания света,

— $\boldsymbol{n}$ — вектор нормали к поверхности объекта,

— $\boldsymbol{l}$ — вектор направления света,

— $\cos\theta$ — угол между вектором направления света и нормалью к поверхности объекта.

Фоновое освещение

Если учитывать только рассеяный свет при освещении объекта, будут видны абсолютно неосвещённые, чёрные грани. В реальном мире такое встречается редко, в связи с чем, для достижения большей реалистичности, было предложено ввести минимальный уровень освещённости — фоновое освещение, являющееся обычно результатом отражения света от других объектов сцены [9, 11].

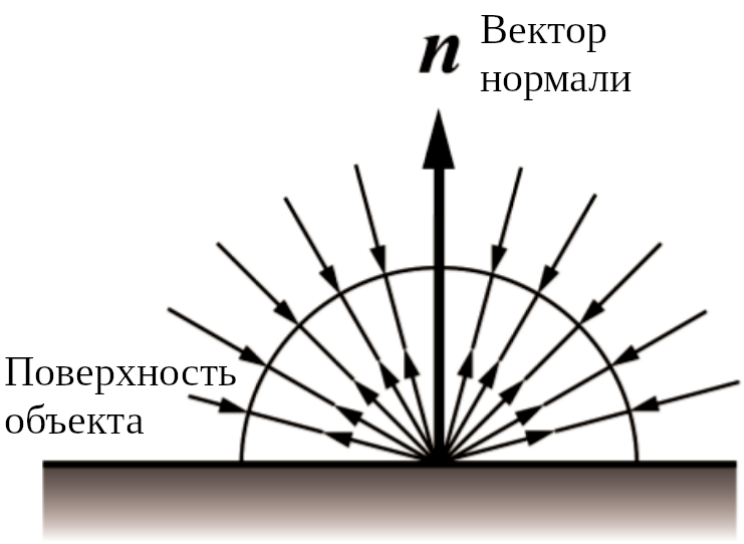

Рисунок 3 – Модель Фонга: Фоновая составляющая света (Источник: [9])

Фоновая составляющая света, согласно [9], рассчитывается по формуле 2, приведённой ниже.

$$I_a = k_a I_0, \tag{2}$$

где

— k_a — отражательная способность объекта,

— I_0 — интенсивность фонового освещения.

Зеркальный свет

В реальном мире гладкие объекты отражают свет подобно зеркалу — чем менее объект шероховатый, тем более отчётливо он отражает источник света, и виден блик. Для достижения подобного эффекта было предложено использовать функцию косинуса и регулировать размытость границ блика с помощью возведения косинуса угла между вектором взгляда и отражённым светом в различные степени [11].

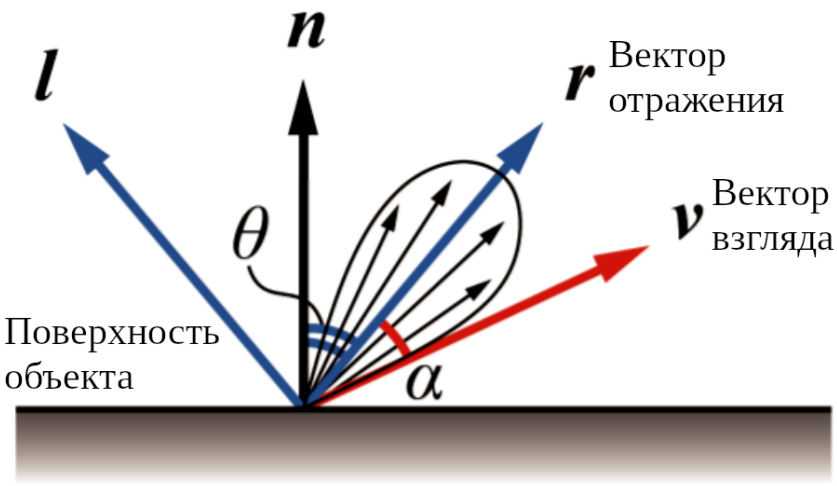

Рисунок 4 – Модель Фонга: Зеркальная составляющая света (Источник: [9])

Зеркальная составляющая света, согласно [9], рассчитывается по формуле 3, приведённой ниже.

$$I_s = k_s I_l \cos^n \alpha = k_s I_l \left| \boldsymbol{r} \cdot \boldsymbol{v} \right|^n , \tag{3}$$

где

— k_s — коэффициент зеркального отражения,

— $\boldsymbol{r}$ — вектор отражённого света,

— $\boldsymbol{v}$ — вектор взгляда,

— $\cos \alpha$ — угол между вектором взгляда и вектором отражённого света,

— n — коэффициент шероховатости поверхности.

Сумма рассеянной, фоновой и зеркальной составляющих даёт итоговый отражённый свет:

$$I = k_d I_l \left| \boldsymbol{n} \cdot \boldsymbol{l} \right| + k_a I_0 + k_s I_l \left| \boldsymbol{r} \cdot \boldsymbol{v} \right|^n \tag{4}$$

2.3 Выбор типов и структур данных

Далее будут выбраны типы и структуры данных для представления объектов в разрабатываемой программе.

Таблица 3 – Выбор типов и структур данных для представления объектов

Объект / структура	Представление / тип данных
Сцена	Структура Scene с полями: — objects типа массив структур Object (для хранения всех объектов сцены).
Объект сцены	Структура Object с полями: — mesh типа структура Mesh (для хранения информации о геометрии объекта), — transform типа структура Transform (для хранения информации о положении объекта в пространстве), — color типа структура Vec3 (для хранения информации о цвете объекта), — collider типа структура AABB (для хранения информации о коллайдере объекта), — rigidbody типа структура RigidBody (для хранения информации о физических свойствах объекта).

Таблица 3 – Выбор типов и структур данных для представления объектов

Структура Mesh	Структура Mesh с полями: — vertices типа массив структур Vertex (для хранения информации о вершинах объекта).
Структура Vertex	Структура Vertex с полями: — position типа структура Vec3 (для хранения информации о координатах вершины), — normal типа структура Vec3 (для хранения информации о нормали к вершине).
Структура Vec3	Структура Vec3 с полями: — x типа число с плавающей точкой, — y типа число с плавающей точкой, — z типа число с плавающей точкой.
Структура Transform	Структура Transform с полями: — position типа структура Vec3 (для хранения информации о смещении объекта в пространстве), — rotation типа структура Vec3 (для хранения информации о повороте объекта), — scale типа структура Vec3 (для хранения информации о масштабировании объекта).

Таблица 3 – Выбор типов и структур данных для представления объектов

Структура AABB	Структура AABB с полями: — min типа структура Vec3 (для хранения информации о вершине ограничивающего параллелепипеда с минимальными координатами по трём осям), — max типа структура Vec3 (для хранения информации о вершине ограничивающего параллелепипеда с максимальными координатами по трём осям).
Структура RigidBody	Структура RigidBody с полями: — velocity типа структура Vec3 (для хранения информации о скорости объекта), — acceleration типа структура Vec3 (для хранения информации об ускорении объекта), — force типа структура Vec3 (для хранения информации о силе, действующей на объект), — mass типа число с плавающей точкой (для хранения информации о массе объекта).

Таблица 3 – Выбор типов и структур данных для представления объектов

Камера	Структура Camera с полями: — position типа структура Vec3 (для хранения информации о местоположении камеры), — front типа структура Vec3 (для хранения информации, нужной для рассчёта направления вектора взгляда камеры), — up типа структура Vec3 (для хранения информации, нужной для рассчёта направления вектора взгляда камеры), — yaw типа число с плавающей точкой (для хранения информации о повороте камеры), — pitch типа число с плавающей точкой (для хранения информации о повороте камеры), — fov типа число с плавающей точкой (для хранения информации о соотношении сторон камеры).

2.4 Вывод

В данной части были приведены требования к программному обеспечению, на формальном языке были описаны алгоритмы, которые будут реализованы при разработке программного обеспечения, а также был обоснован выбор типов и структур, которые будут использованы при разработке.

3 Технологическая часть

В данной части будет обоснован выбор графического API, языка программирования и среды разработки, которые будут использоваться при разработке программного обеспечения. Также будет приведена UML-диаграмма классов, описывающая структуру программы. Будет продемонстрирован интерфейс, и будут приведены примеры работы программы.

3.1 Средства реализации

Далее будет обоснован выбор языка программирования и средств разработки, использованных при разработке программы.

3.1.1 Выбор графического API

В качестве используемого графического API был выбран OpenGL [12] по следующим причинам:

— кроссплатформенность (в отличие от DirectX [13]);

— простота инициализации (в отличие от Vulkan [14]);

— использование вычислительных мощностей графического ускорителя (в отличие от рендеринга на процессоре);

— спецификация OpenGL реализована в драйверах большинства массовых видеокарт (NVIDIA, AMD, Intel), в связи с чем приложение возможно будет запустить практически на любом персональном компьютере.

3.1.2 Выбор языка программирования

В качестве используемого языка программирования был выбран C++ по следующим причинам:

— язык широко используется при разработке графических приложений;

— наличие комплияторов, генерирующих высокопроизводительный исполняемый код;

— язык типизирован, в связи с чем в процессе разработки возникает меньше ошибок времени выполнения;

— наличие библиотек для обеспечения доступа к функциям OpenGL (glad [15]), а также для создания окон и управления вводом (GLFW [16]);

— наличие математических библиотек (glm [17]);

— язык поддерживается отладчиками gdb [18] и RenderDoc [19];

— наличие кроссплатформенной утилиты для автоматической сборки программы cmake.

3.1.3 Выбор среды разработки

В качестве среды разработки был выбран текстовый редактор Neovim [20] по следующим причинам:

— высокая отзывчивость в отличие от графических сред разработки, таких как Visual Studio, Visual Studio Code, Clion, QtCreator;

— полная поддержка vim-движений, что ускорит навигацию по исходному коду проекта;

— возможность использования протокола Language Server, что позволит определять наличие ошибок времени компиляции в коде без необходимости компиляции проекта;

— наличие расширений, ещё больше ускоряющих процесс разработки и навигации по исходному коду проекта (harpoon, vim-fugitive, nvim-tree, nvim-cmp, nvim-treesitter, nvim-lspconfig, telescope-nvim, luasnip, vim-surround, vim-commentary, friendly-snippets).

3.2 Структура программы

На рисунке 5 представлена диаграмма реализованых в процессе разработки ПО классов.

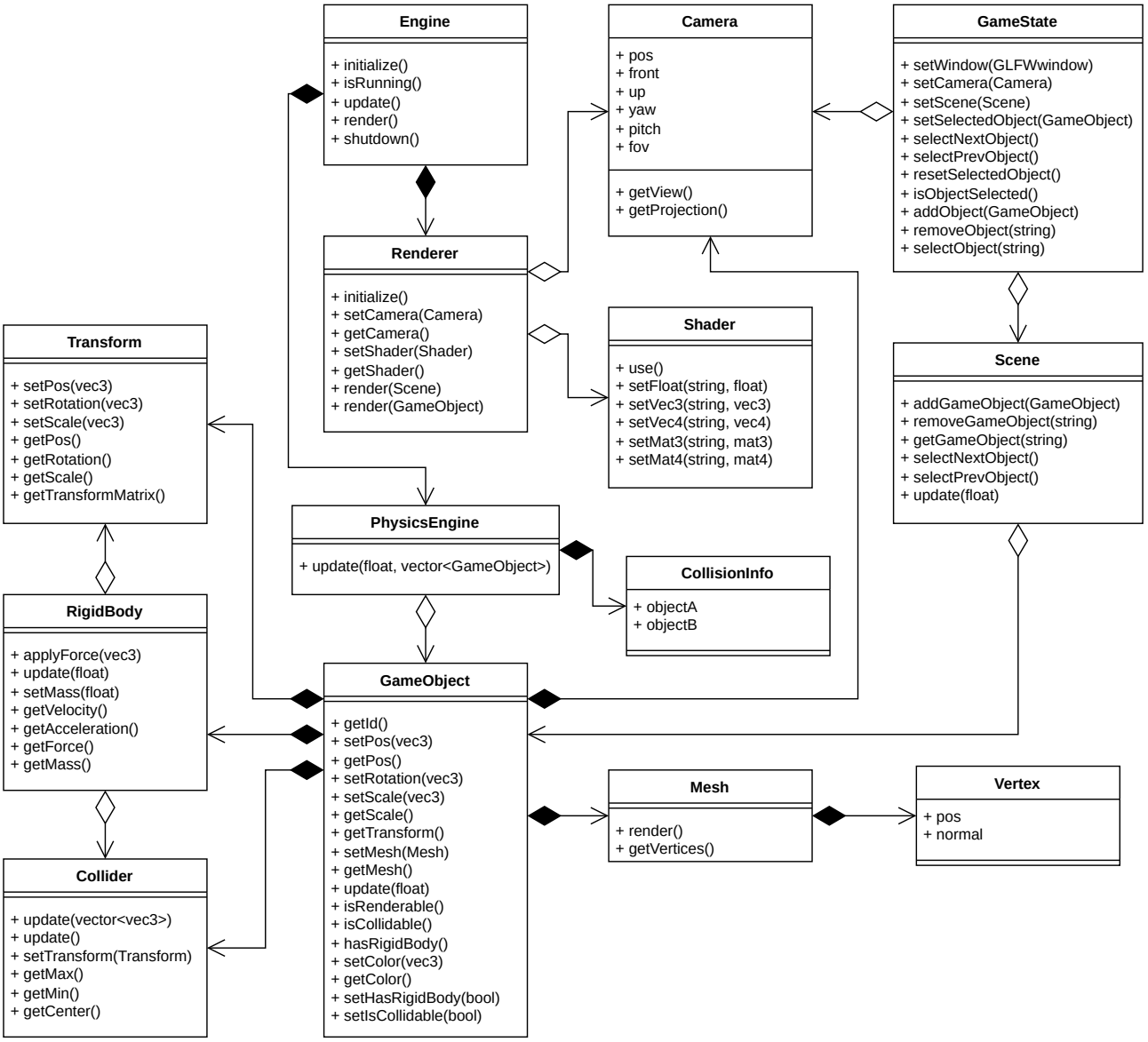

Рисунок 5 – Диаграмма классов разработанного ПО

Engine

Класс Engine является центральным компонентом разработанного ПО, он отвечает за создание всех объектов, нужных для запуска приложения. В главном цикле программы (см. Листинг 1) он перенаправляет запросы на обработку ввода, обновление объектов и рендеринг сцены своим компонентам (PhysicsEngine и Renderer).

```
int main() {
    Engine engine;
    engine.initialize();

    while (engine.isRunning()) {
        engine.processInput();
        engine.update();
        engine.render();
    }
    // ...
}
```

Листинг 1 – Точка входа и главный цикл

PhysicsEngine

Класс PhysicsEngine отвечает за обновление местоположения объектов в соответствии с атрибутами их компонента RigidBody (скорость, ускорение, сила и т.д.), обновление этих атрибутов на основе времени, прошедшего с момента генерации последнего кадра, а также обнаружение и разрешение коллизий между объектами. Для обнаружения коллизий используются компоненты Collider объектов. В случае обнаружения коллизий создаётся объект CollisionInfo, который хранит указатели на столкнувшиеся объекты. Процесс обнаружения коллизий выглядит следующим образом:

1) Для каждого объекта сцены, если он имеет компонент Collidable, добавить его во временный массив Collidables объектов, которые могут сталкиваться.

2) Для каждой пары объектов из массива Collidables проверить, сталкиваются они или нет.

3) Если объекты сталкиваются, создать объект CollisionInfo и добавить его во временный массив пар сталкивающихся объектов Collisions.

4) Для каждой пары объектов из массива Collisions разрешить коллизии, обновив местоположения объектов и их физические характеристики.

Renderer

Класс Renderer отвечает за рендеринг объектов сцены и генерацию кадра. Рендеринг происходит следующим образом:

1) Для каждого объекта сцены, вызывается функция render класса Renderer.
2) В функции render класса Renderer используются компоненты Mesh, Color и Transform объекта сцены: переменные шейдера инициализируются соответствующими данными компонентов Color и Transform, после чего вызывается функция render компонента Mesh.
3) В функции render компонента Mesh происходит вызов функций OpenGL, в результате чего происходит отрисовка объекта.

GameState

Класс GameState, реализующий паттерн Одиночка, предназначен для хранения глобального состояния программы. Получение указателей на текущую камеру, сцену и выбранный объект происходит по запросу к GameState. Часть интерфейса класса GameState представлена на листинге 2.

```
1  class GameState {
2  public:
3      static GameState& get() {
4          static GameState instance;
5          return instance;
6      }
7      // ...
8      std::shared_ptr<Camera> getCamera() const;
9      std::shared_ptr<Scene> getScene() const;
10     std::shared_ptr<GameObject> getSelectedObject() const;
11     // ...
12 };
```

Листинг 2 – Класс GameState

3.3 Интерфейс

На рисунках ниже представлен интерфейс программы.

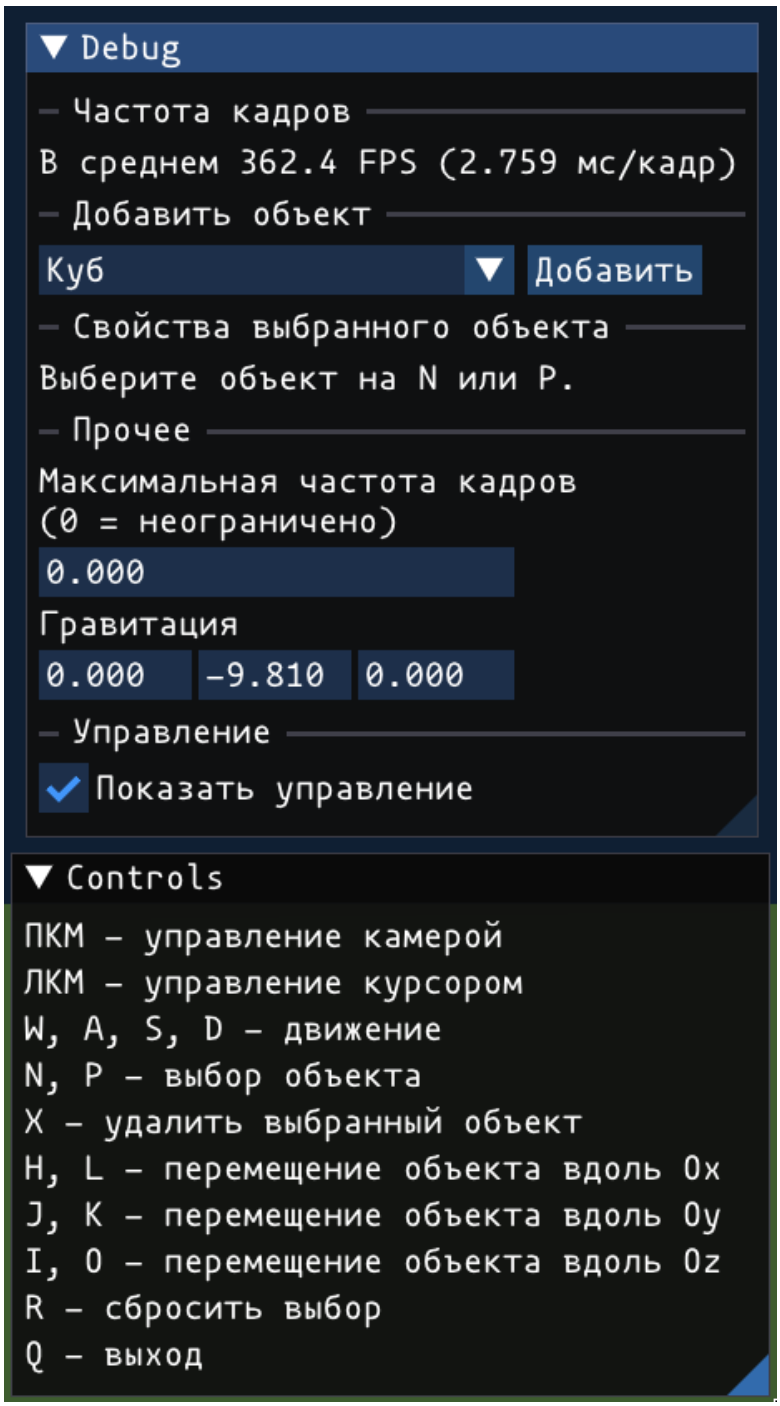

Рисунок 6 – Демонстрация интерфейса, объект не выбран

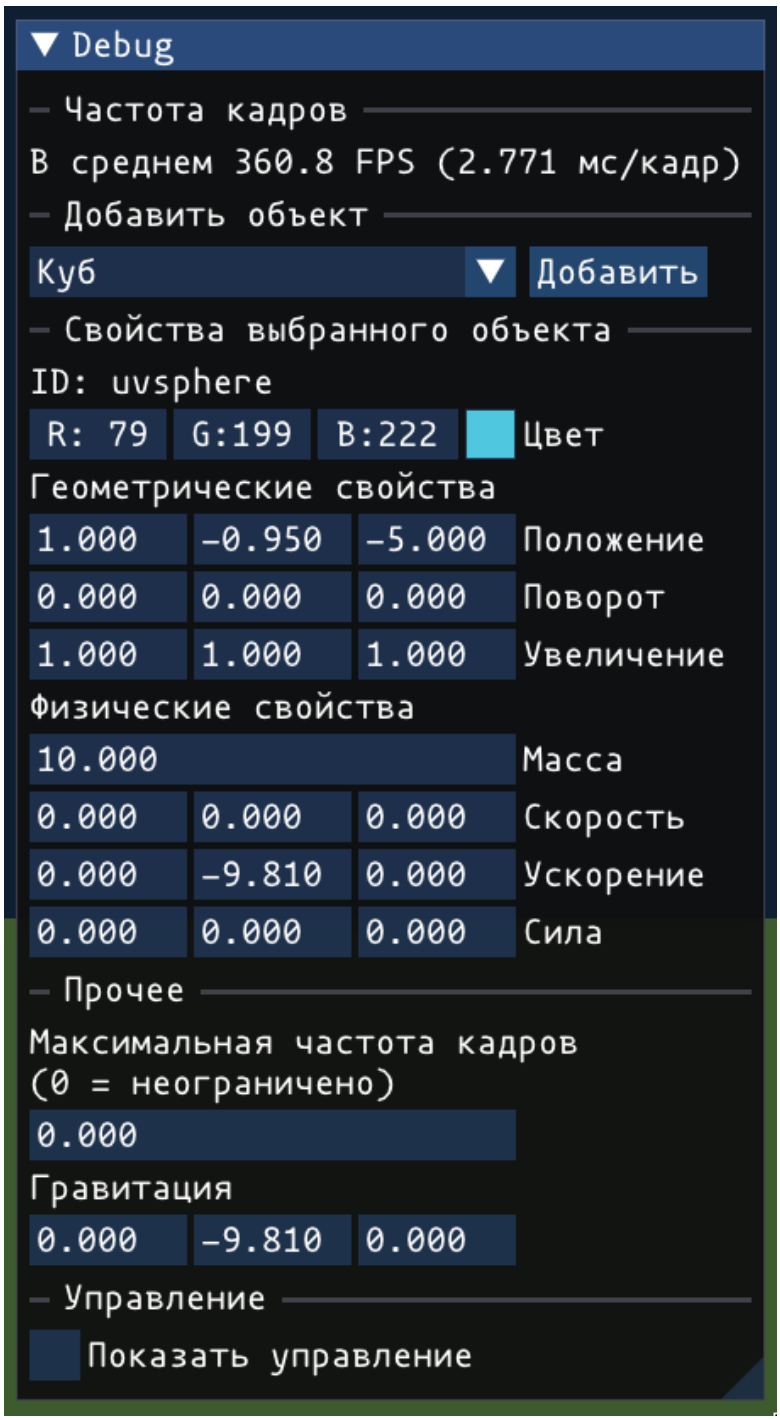

Рисунок 7 – Демонстрация интерфейса, объект выбран

3.4 Демонстрация работы программы

На рисунках ниже представлена демонстрация работы программы.

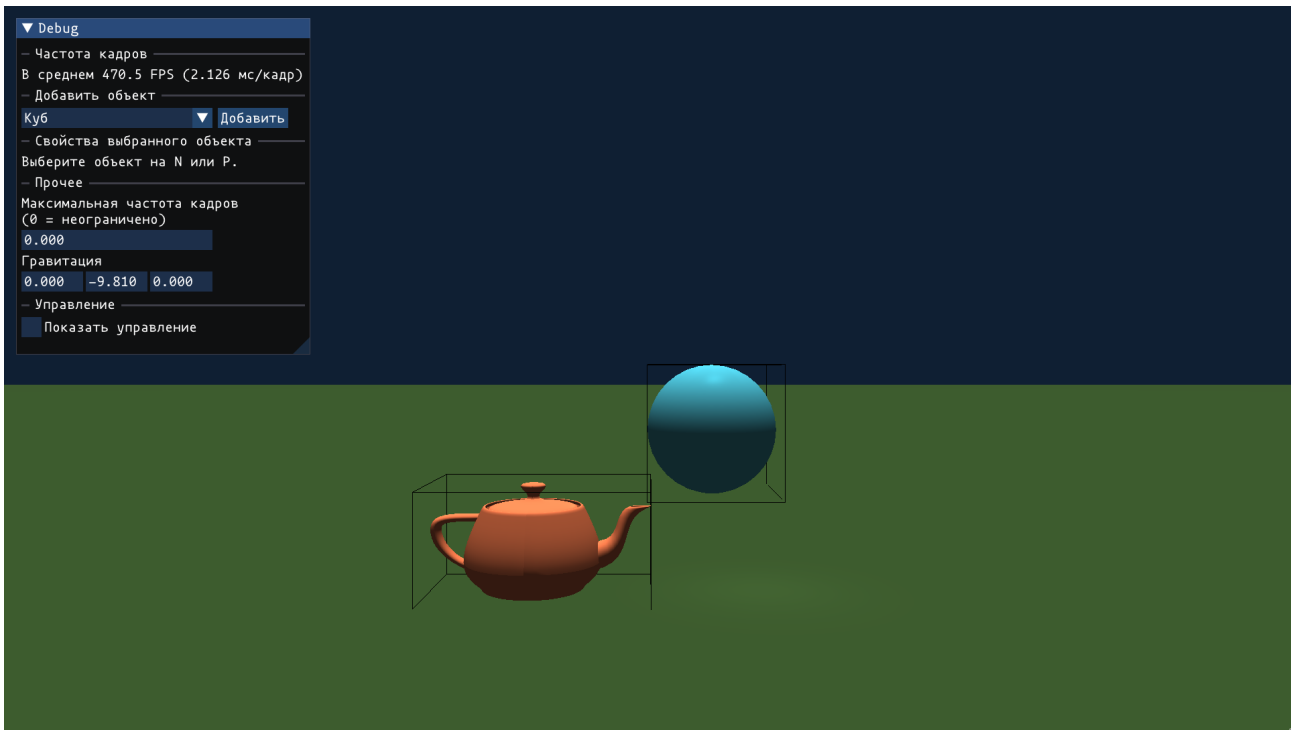

Рисунок 8 – Демонстрация работы программы сразу после запуска

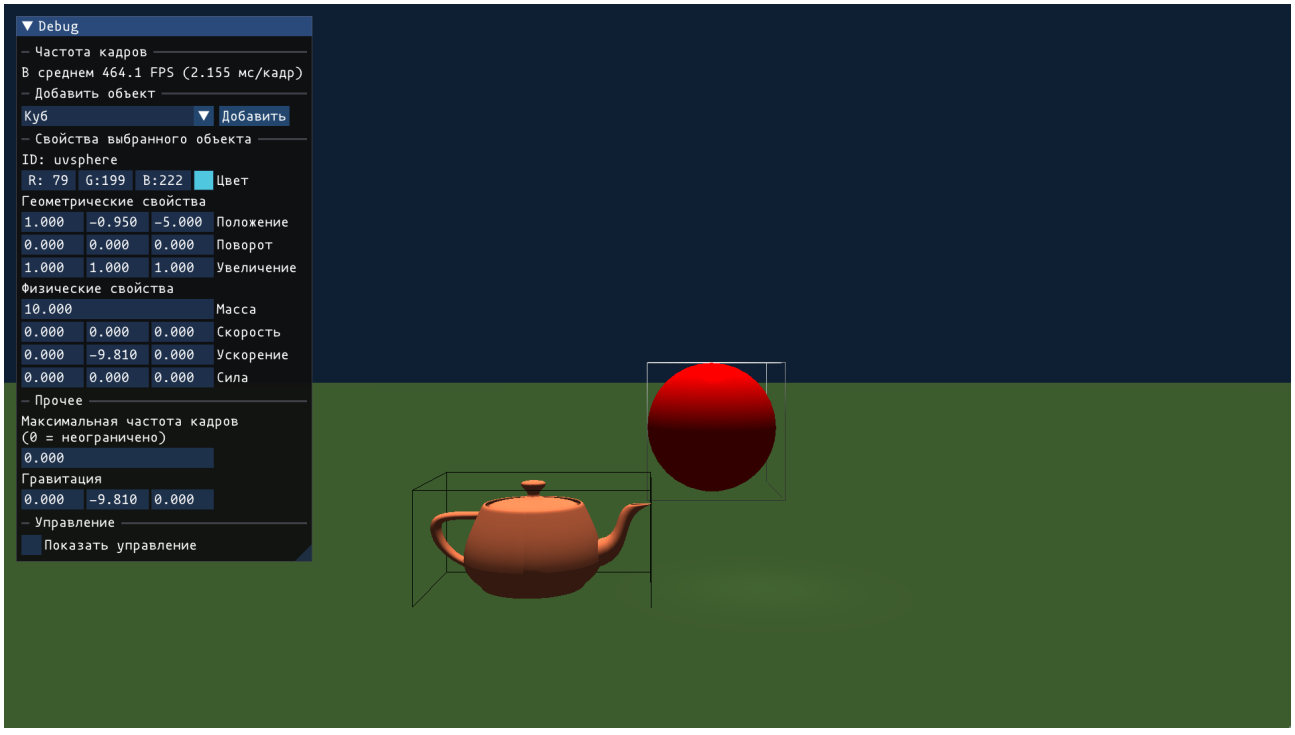

Рисунок 9 – Демонстрация выбора объекта

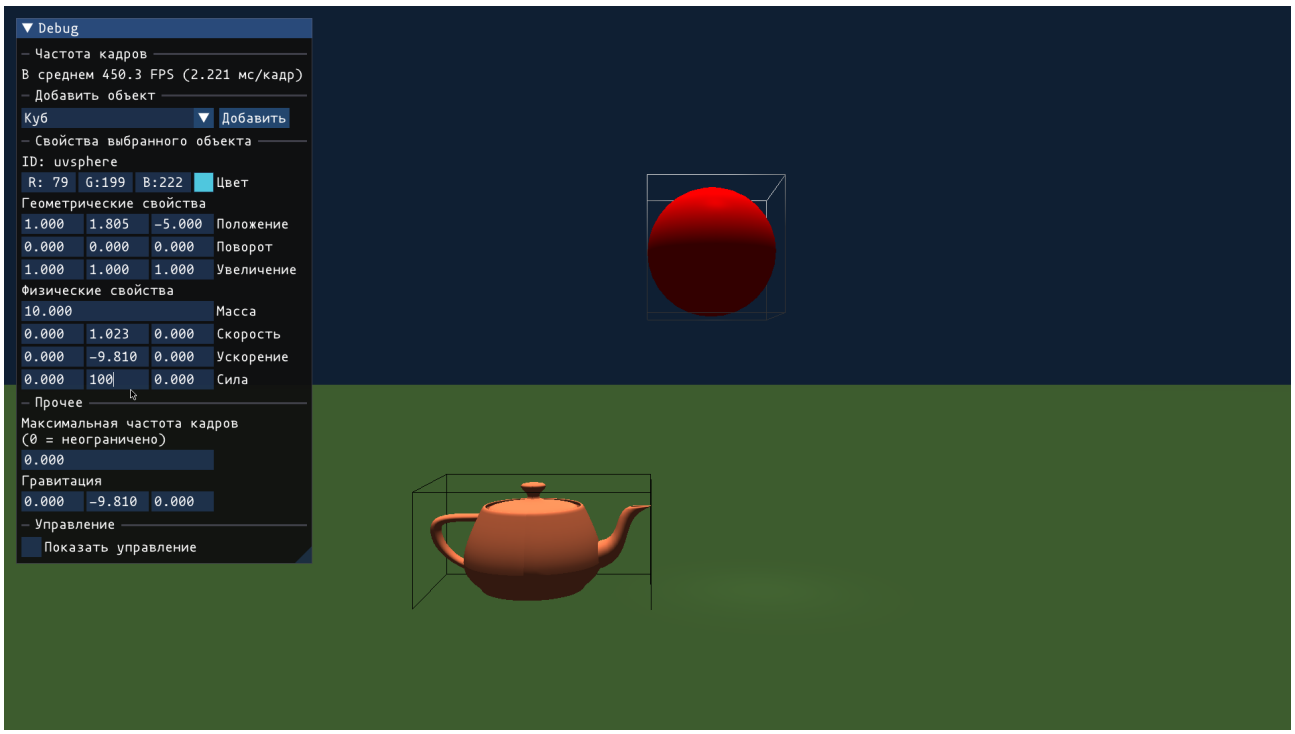

Рисунок 10 – Демонстрация приложения силы к объекту

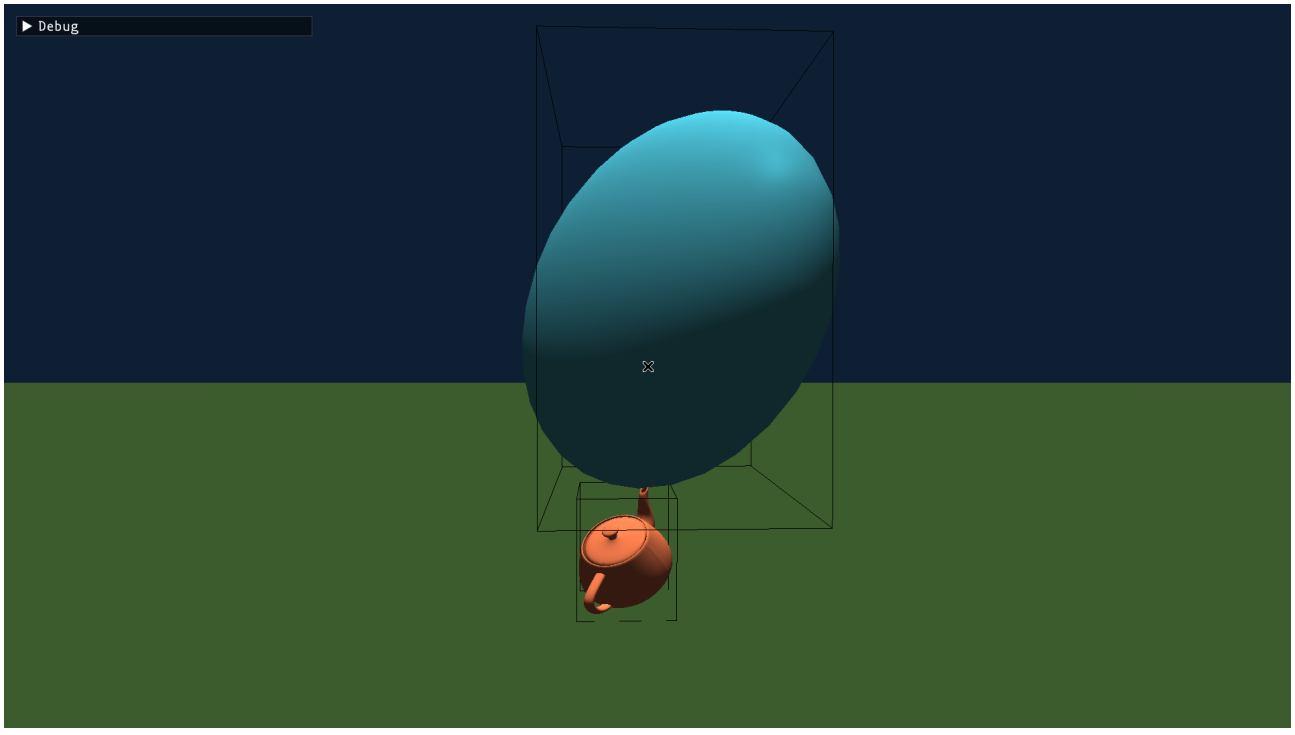

Рисунок 11 – Демонстрация перемещения, поворота и масштабироавния объектов

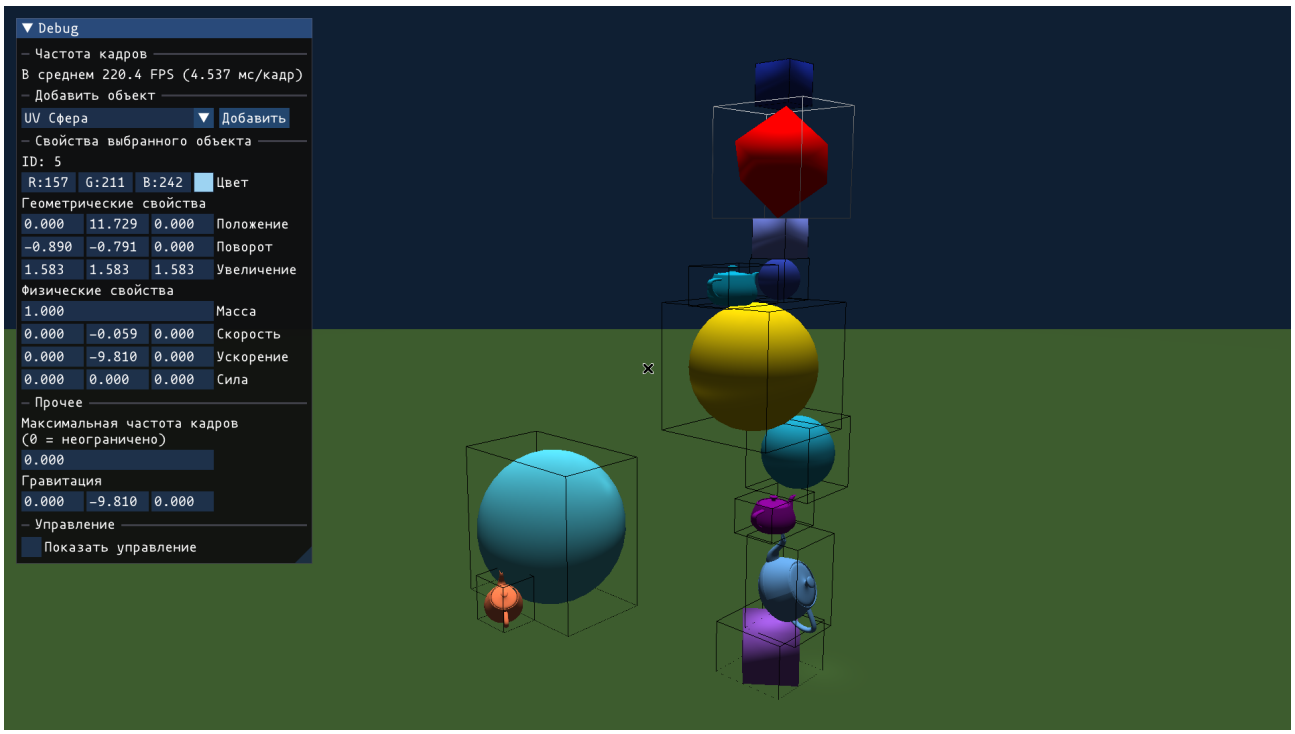

Рисунок 12 – Демонстрация создания объектов

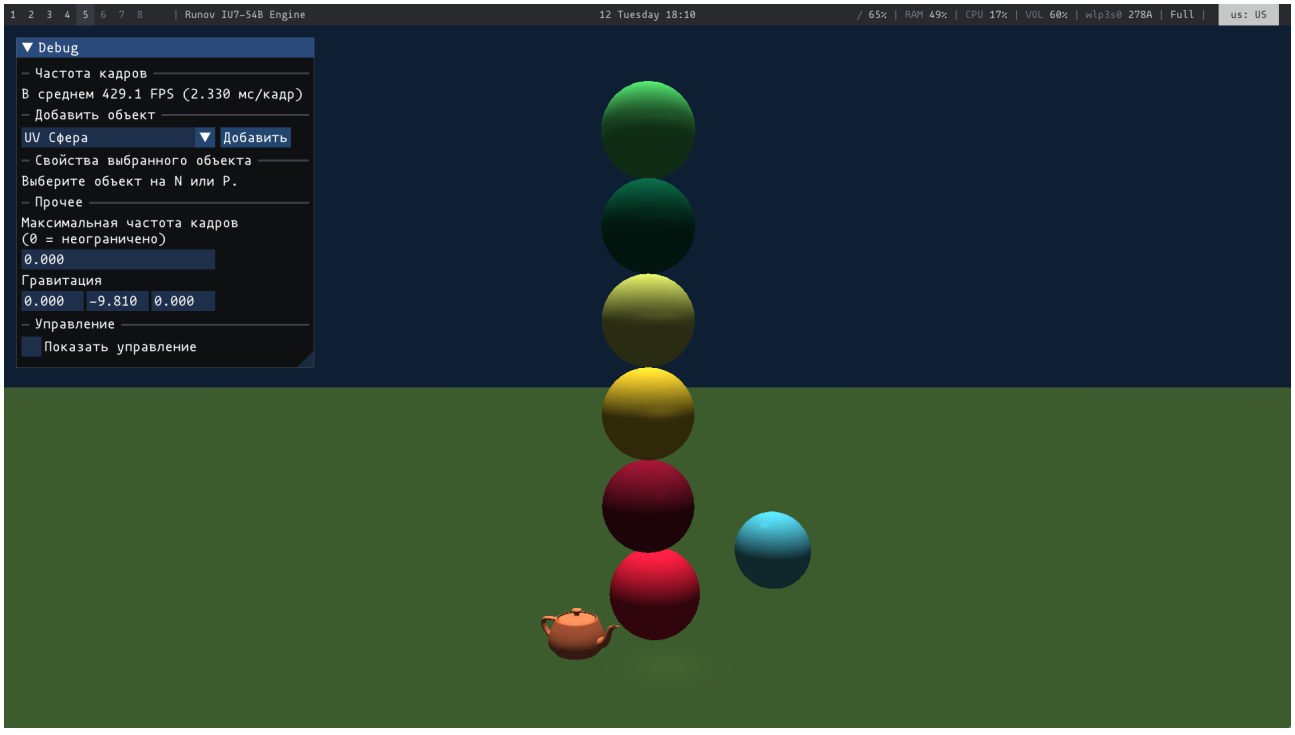

Рисунок 13 – Демонстрация отключения визуализации AABB-коллайдеров объектов

3.5 Вывод

В данной части был обоснован выбор графического API, языка программирования и среды разработки, которые были использованы при разработке программного обеспечения. Также была приведена UML-диаграмма классов, описывающая структуру программы. Был продемонстрирован интерфейс, и приведены примеры работы программы.

4 Исследовательская часть

4.1 Технические характеристики

Технические характеристики устройства, на котором выполнялось тестирование:

— Процессор: AMD Ryzen 7 4700U 2.0 ГГц [21];

— Видеокарта: Radeon Graphics (встроенная);

— Оперативная память: 8 ГБ, DDR4, 3200 МГц;

— Операционная система: NixOS 23.11.1209.993d7 [22];

— Версия ядра: 6.1.64.

Тестирование проводилось на компьютере, включенном в сеть электропитания. Во время тестирования ноутбук был нагружен только встроенными приложениями окружения, а также непосредственно самим тестируемым приложением.

4.2 Средства проведения опытов

В листинге 3 приведена функция, используемая для получения процессорного времени в рамках проведения опытов. В листинге 4 приведён пример её использования в рамках проведения опытов.

```
1 long bm::get_cpu_time_ns() {
2     struct timespec t;
3     if (clock_gettime(CLOCK_THREAD_CPUTIME_ID, &t)) {
4         std::cerr << "Невозможно получить время.\n";
5         return -1;
6     }
7     return t.tv_sec * 1000000000LL + t.tv_nsec;
8 }
```

Листинг 3 – Функция получения процессорного времени

```
int main() {
    Engine engine;
    engine.initialize();

    while (engine.isRunning()) {
        engine.processInput();
        engine.update();

        long start = bm::get_cpu_time_ns();
        engine.render();
        gRenderTime = bm::get_cpu_time_ns() - start;
    }
    // ...
}
```

Листинг 4 – Проведение замеров времени

gRenderTime — глобальная переменная, из которой по запросу проводилось считывание времени генерации кадра в процессе проведения опытов.

4.3 Описание проводимых опытов

Далее следует описание опытов, проведённых в рамках исследования.

4.3.1 Опыт 1

Цель опыта — получить «базовое» время генерации кадра, с которым впоследствии будет сравниваться времени генерации кадра из других опытов.

Характеристики опыта:

— Количество объектов на сцене: $1 + 5N, N = \overline{0; 25}$

— Типы объектов сцены: $1 + 5N$ кубов

— Общее количество треугольников объектов сцены: $12 \cdot (1 + 5N)$

— Столкновения: Нет

— Количество вызовов OpenGL-функций отрисовки: $1 + 5N$

4.3.2 Опыт 2

Цель опыта — определение влияния количества вызовов OpenGL-функций отрисовки на время генерации кадра, в случае, когда сцена состоит из одних кубов.

Предположение — чем больше вызовов OpenGL-функций, тем дольше будет генерироваться кадр.

Характеристики опыта:

— Количество объектов на сцене: $1 + 5N, N = \overline{0; 25}$

— Типы объектов сцены: $1 + 5N$ кубов

— Общее количество треугольников объектов сцены: $12 \cdot (1 + 5N)$

— Столкновения: Нет

— Количество вызовов OpenGL-функций отрисовки: $12 \cdot (1 + 5N)$

4.3.3 Опыт 3

Цель опыта — определение влияния количества столкновений на время рендеринга.

Предположение — количество столкновений не влияет на время рендеринга, потому что время выполнения функции $engine.update()$ в рамках проведения опыта не учитывается.

Характеристики опыта:

— Количество объектов на сцене: $1 + 5N, N = \overline{0; 25}$

— Типы объектов сцены: $1 + 5N$ кубов

— Общее количество треугольников объектов сцены: $12 \cdot (1 + 5N)$

— Столкновения: Да

— Количество вызовов OpenGL-функций отрисовки: $1 + 5N$

4.3.4 Опыт 4

Цель опыта — определение влияния количества треугольников моделей объектов сцены на время генерации кадра.

Предположение — чем больше треугольников в моделях объектов, тем дольше будет генерироваться кадр.

Характеристики опыта:

— Количество объектов на сцене: $1 + 5N, N = \overline{0; 25}$

— Типы объектов сцены: 1 куб, $5N$ чайников

— Общее количество треугольников объектов сцены: $12 + 6320 \cdot 5N$

— Столкновения: Нет

— Количество вызовов OpenGL-функций отрисовки: $1 + 5N$

4.3.5 Опыт 5

Цель опыта — определение влияния количества вызовов OpenGL-функций отрисовки на время генерации кадра, в случае, когда сцена состоит преимущественно из моделей с большим количеством треугольников.

Предположение — чем больше вызовов OpenGL-функций, тем дольше будет генерироваться кадр.

Характеристики опыта:

— Количество объектов на сцене: $1 + 5N, N = \overline{0; 25}$

— Типы объектов сцены: 1 куб, $5N$ чайников

— Общее количество треугольников объектов сцены: $12 + 6320 \cdot 5N$

— Столкновения: Нет

— Количество вызовов OpenGL-функций отрисовки: $12 + 6320 \cdot 5N$

4.3.6 Опыт 6

Цель опыта — определение влияния количества столкновений на время рендеринга, в случае, когда сцена состоит преимущественно из моделей с большим количеством треугольников.

Предположение — количество столкновений не влияет на время рендеринга, потому что время выполнения функции $engine.update()$ в рамках проведения опыта не учитывается.

Характеристики опыта:

— Количество объектов на сцене: $1 + 5N, N = \overline{0; 25}$

— Типы объектов сцены: 1 куб, $5N$ чайников

— Общее количество треугольников объектов сцены: $12 + 6320 \cdot 5N$

— Столкновения: Да

— Количество вызовов OpenGL-функций отрисовки: $1 + 5N$

4.4 Проведение опытов

На рисунках 14 – 17 представлены снимки экрана, сделанные во время проведения опытов. В таблицах 4 – 9 представлены количественные данные, полученные в результате проведения опытов. На рисунках 18 – 25 представлены графики, построенные по результатам проведённых опытов.

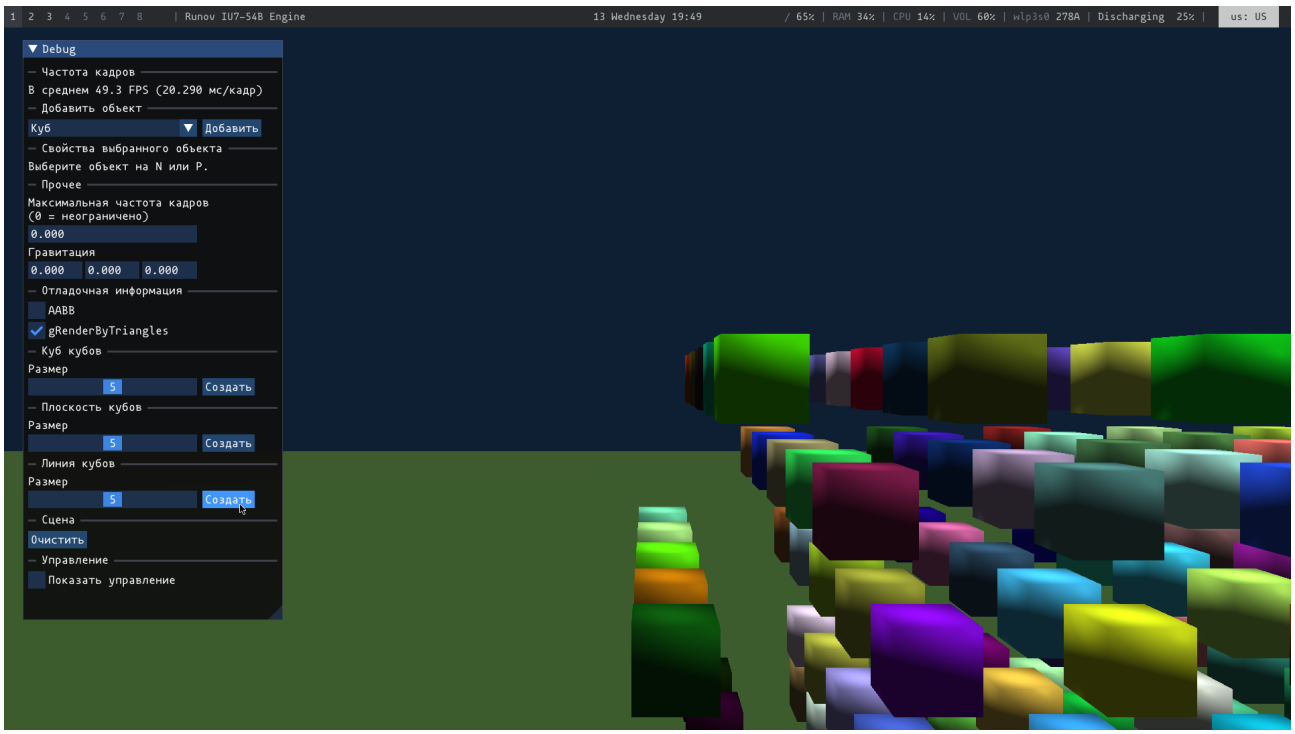

Рисунок 14 – Проведение опытов 1, 2

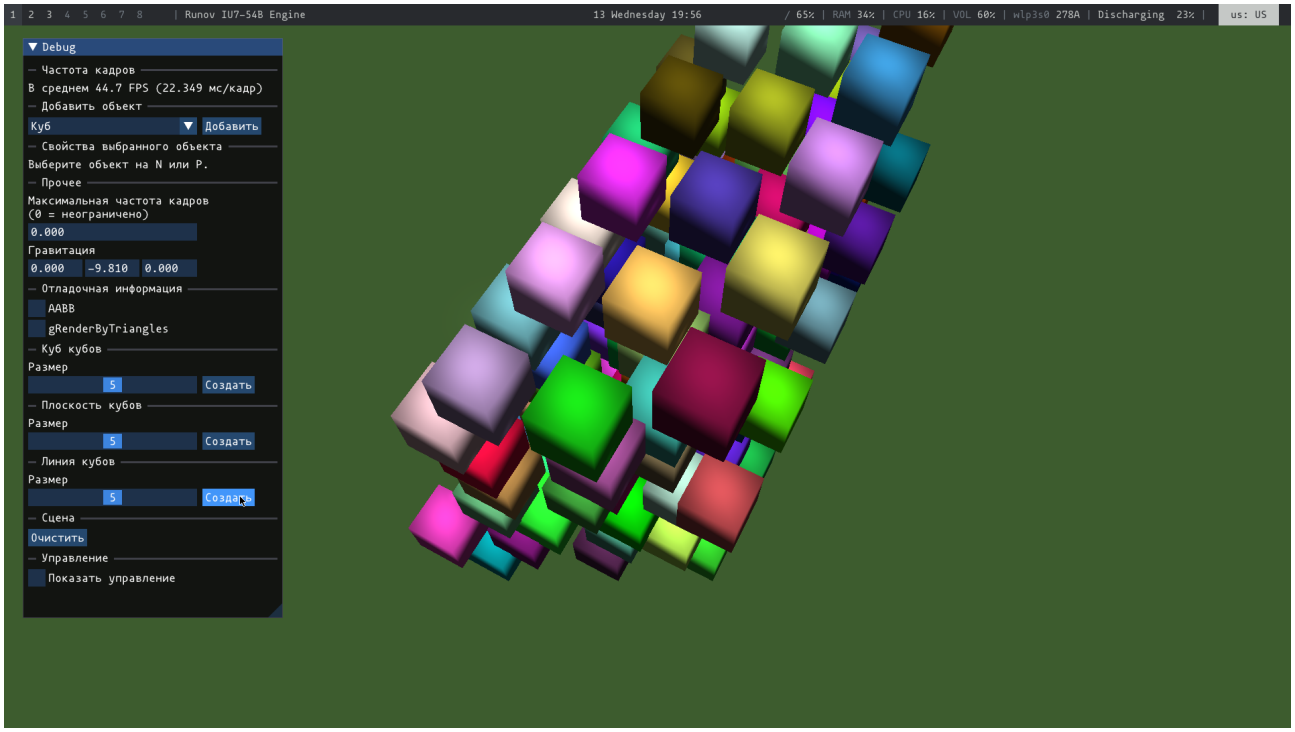

Рисунок 15 – Проведение опыта 3

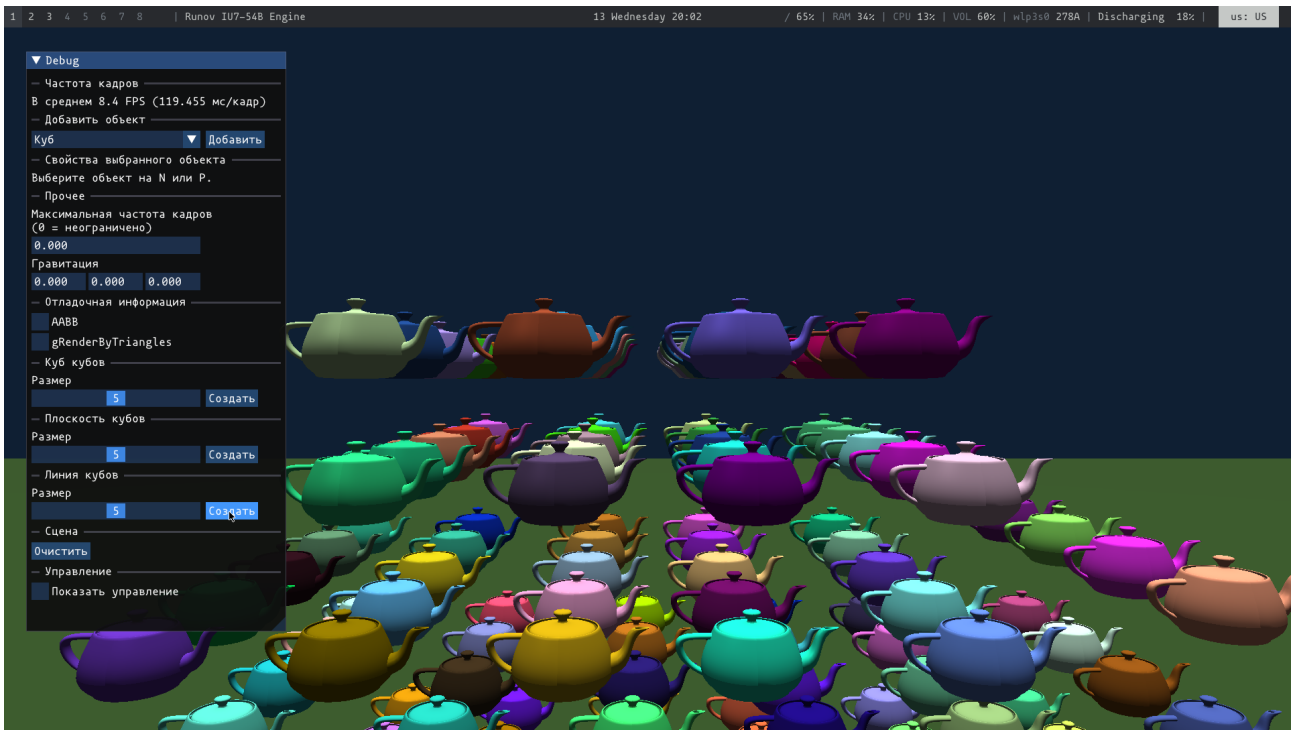

Рисунок 16 – Проведение опытов 4, 5

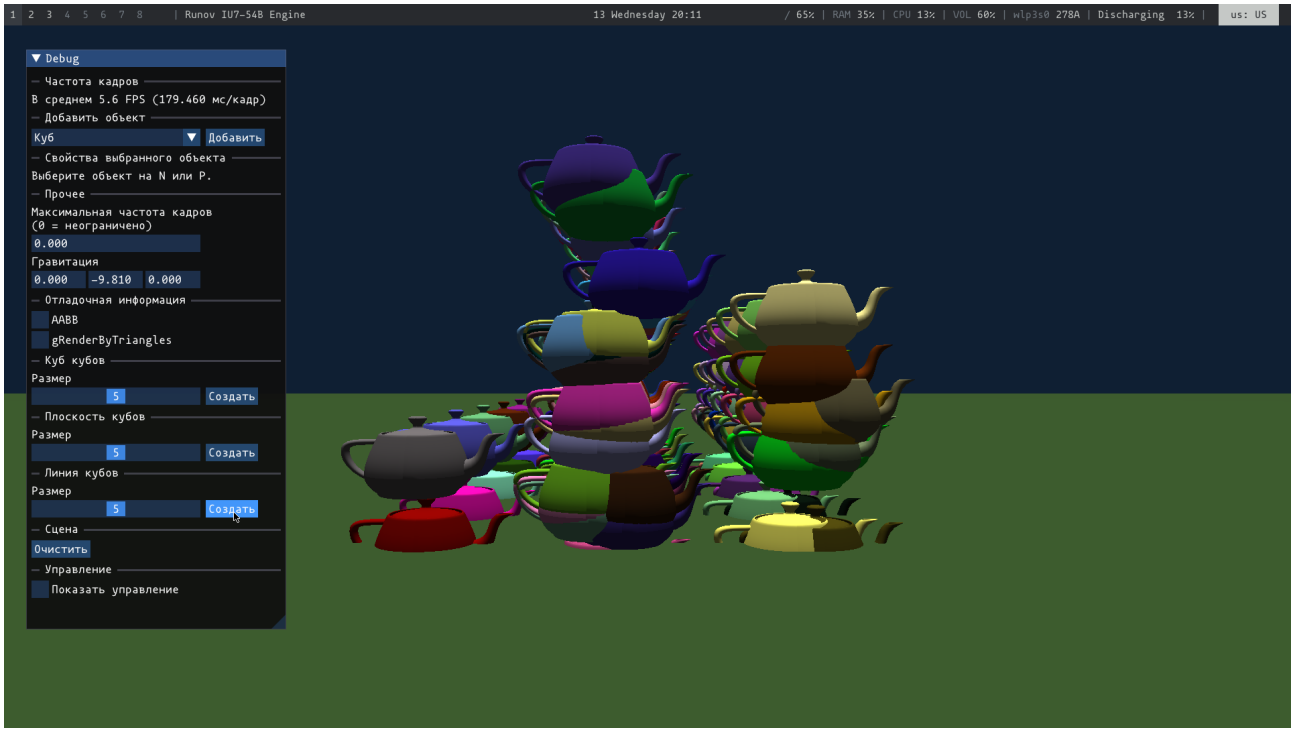

Рисунок 17 – Проведение опыта 6

Таблица 4 – Количественные данные, полученные в результате проведения опыта 1

Количество объектов	Количество треугольников	Количество коллизий	Количество вызовов функций отрисовки OpenGL	Время генерации кадра, мс
26	312	0	26	1.62
46	552	0	46	2.67
86	1032	0	86	4.73
106	1272	0	106	5.75
126	1512	0	126	7.12

Таблица 5 – Количественные данные, полученные в результате проведения опыта 2

Количество объектов	Количество треугольников	Количество коллизий	Количество вызовов функций отрисовки OpenGL	Время генерации кадра, мс
26	312	0	312	1.70
46	552	0	552	2.64
86	1032	0	1032	4.77
106	1272	0	1272	5.87
126	1512	0	1512	7.20

Таблица 6 – Количественные данные, полученные в результате проведения опыта 3

Количество объектов	Количество треугольников	Количество коллизий	Количество вызовов функций отрисовки OpenGL	Время генерации кадра, мс
26	312	36.32	26	1.71
46	552	63.25	46	2.75
86	1032	117.39	86	4.61
106	1272	136.74	106	5.57
126	1512	176.61	126	6.99

Таблица 7 – Количественные данные, полученные в результате проведения опыта 4

Количество объектов	Количество треугольников	Количество коллизий	Количество вызовов функций отрисовки OpenGL	Время генерации кадра, мс
26	158012	0	26	1.85
46	284412	0	46	2.90
86	537212	0	86	5.00
106	663612	0	106	5.98
126	790012	0	126	7.38

Таблица 8 – Количественные данные, полученные в результате проведения опыта 5

Количество объектов	Количество треугольников	Количество коллизий	Количество вызовов функций отрисовки OpenGL	Время генерации кадра, мс
26	158012	0	158012	45.7
46	284412	0	284412	72.9
86	537212	0	537212	125
106	663612	0	663612	159
126	790012	0	790012	190

Таблица 9 – Количественные данные, полученные в результате проведения опыта 6

Количество объектов	Количество треугольников	Количество коллизий	Количество вызовов функций отрисовки OpenGL	Время генерации кадра, мс
26	158012	25.00	26	1.77
46	284412	45.00	46	2.88
86	537212	122.50	86	4.90
106	663612	207.22	106	5.87
126	790012	355.00	126	7.43

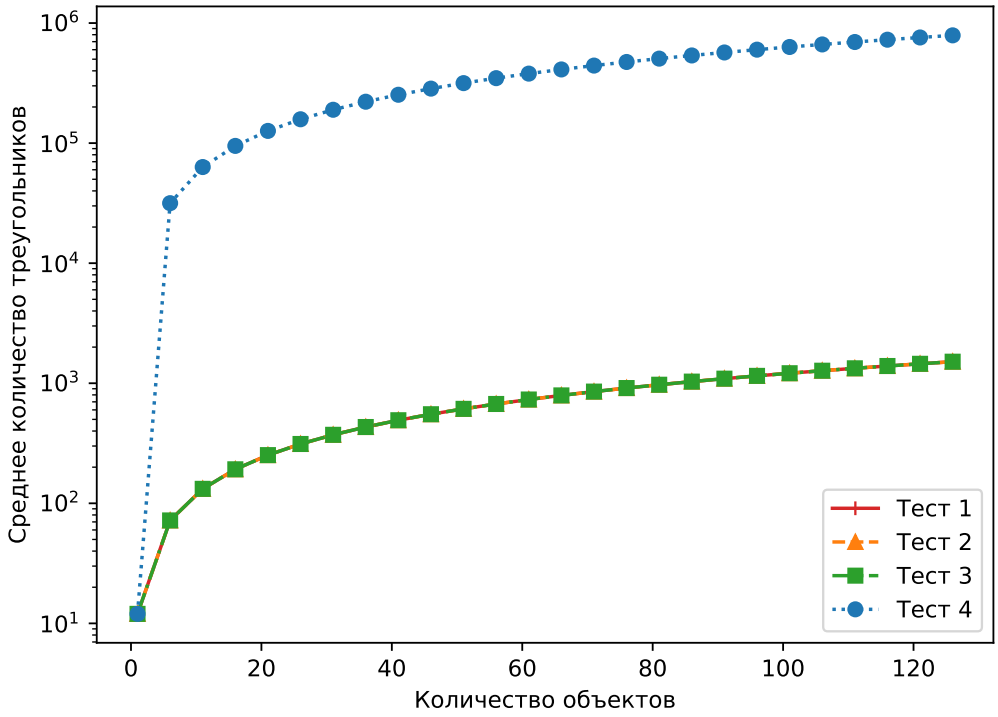

Рисунок 18 – Зависимость количества треугольников от количества объектов в опытах 1 – 4

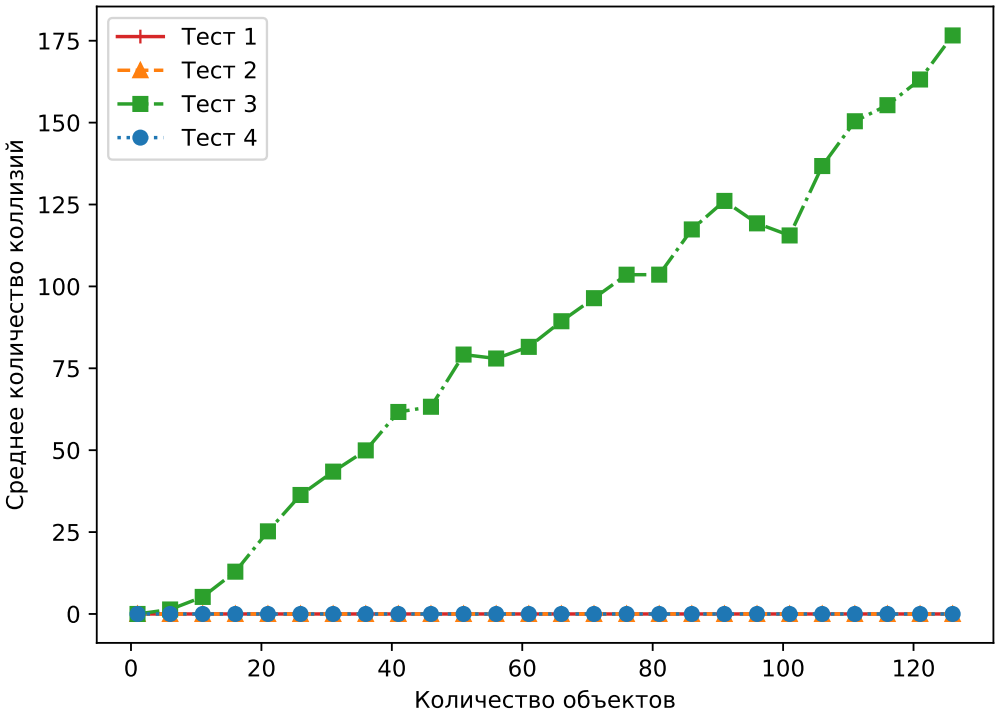

Рисунок 19 – Зависимость количества коллизий от количества объектов в опытах 1 – 4

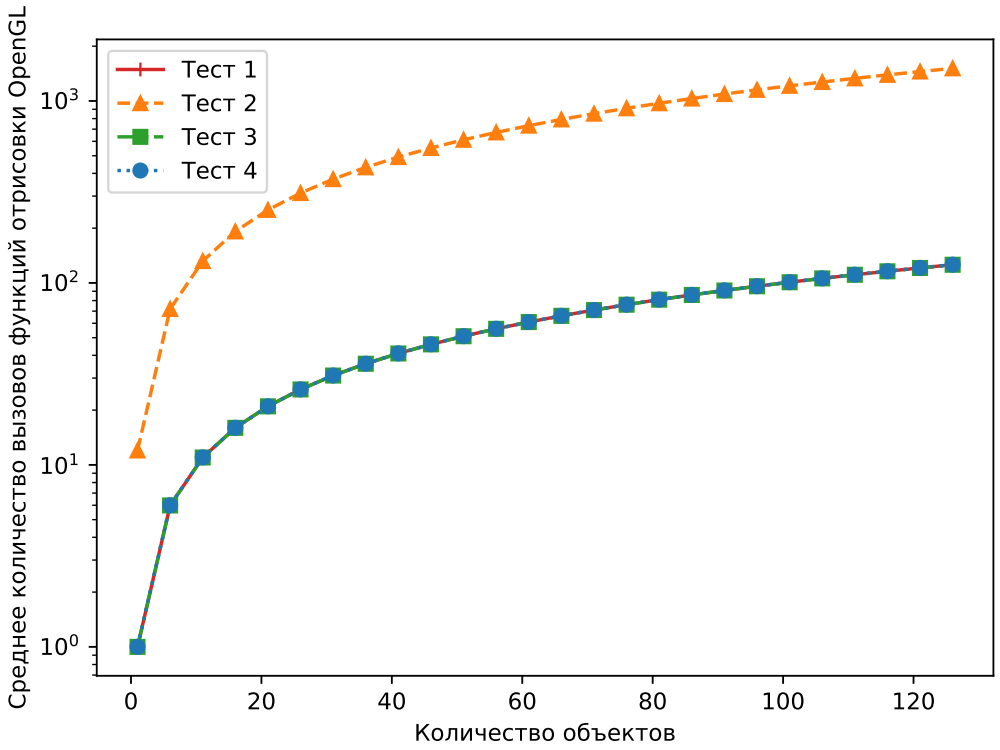

Рисунок 20 – Зависимость количества вызовов OpenGL-функций отрисовки от количества объектов в опытах 1 – 4

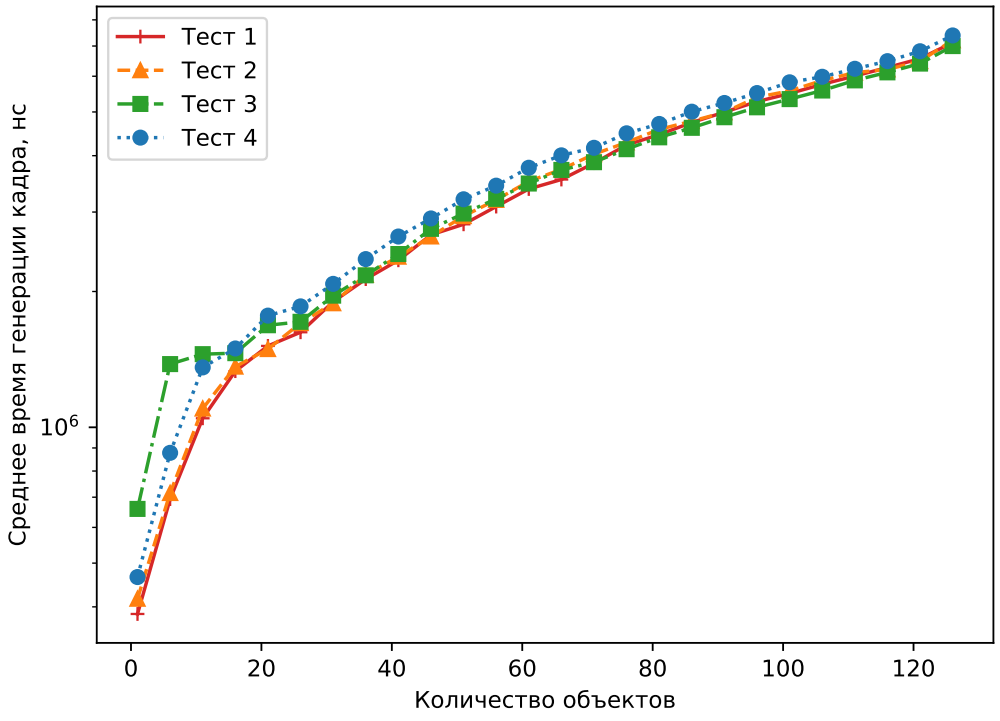

Рисунок 21 – Зависимость времени генерации кадра от количесва объектов в опытах 1 – 4

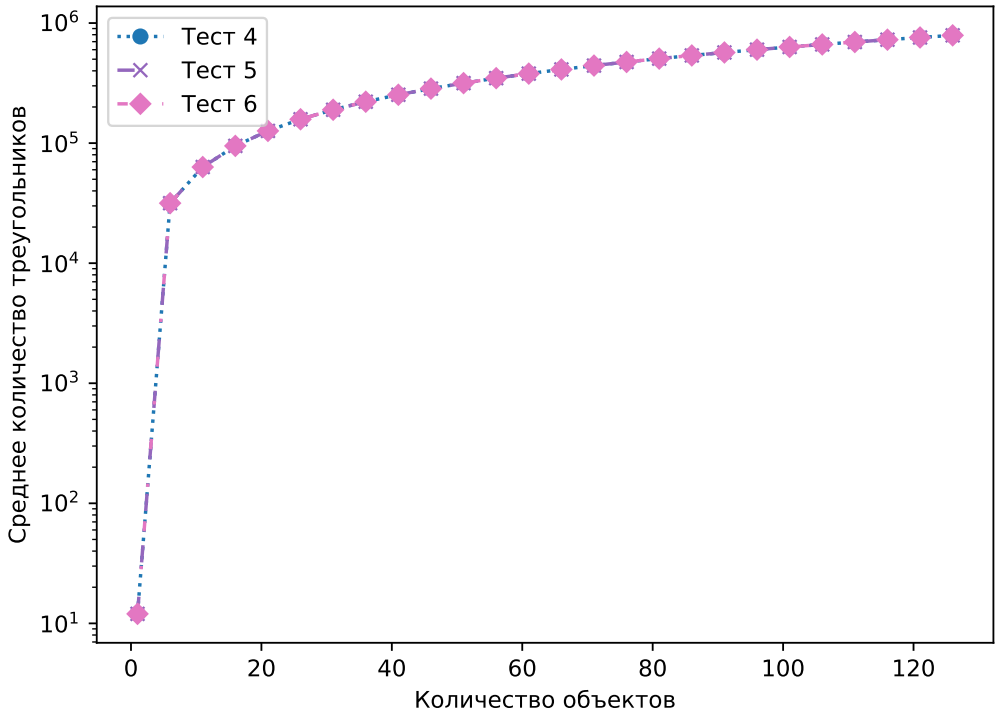

Рисунок 22 – Зависимость количества треугольников от количества объектов в опытах 4 – 6

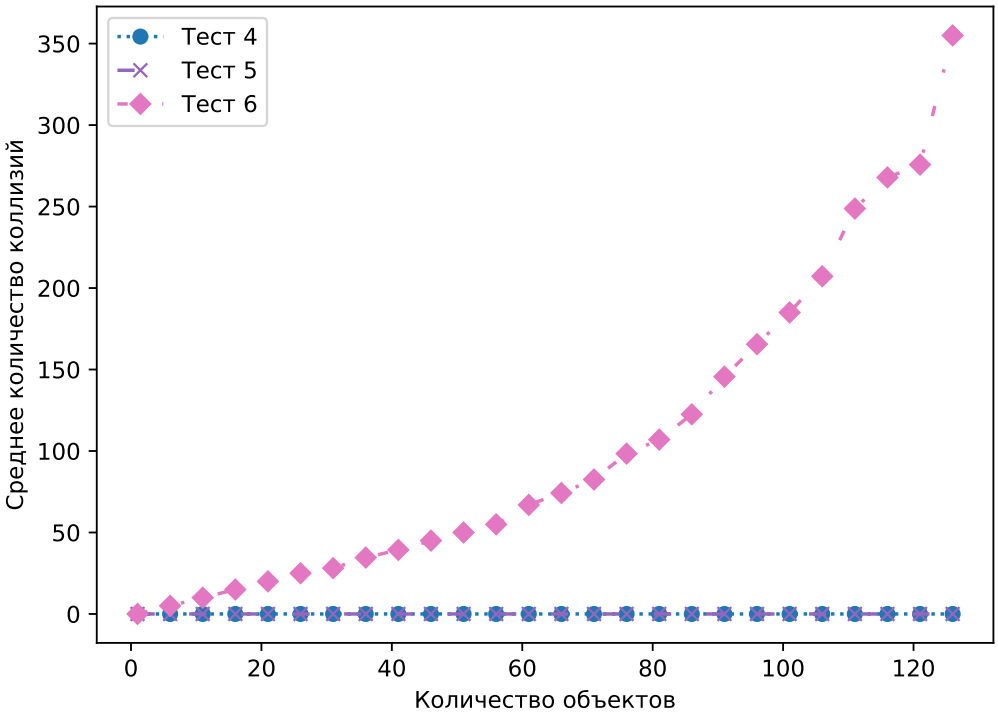

Рисунок 23 – Зависимость количества коллизий от количества объектов в опытах 4 – 6

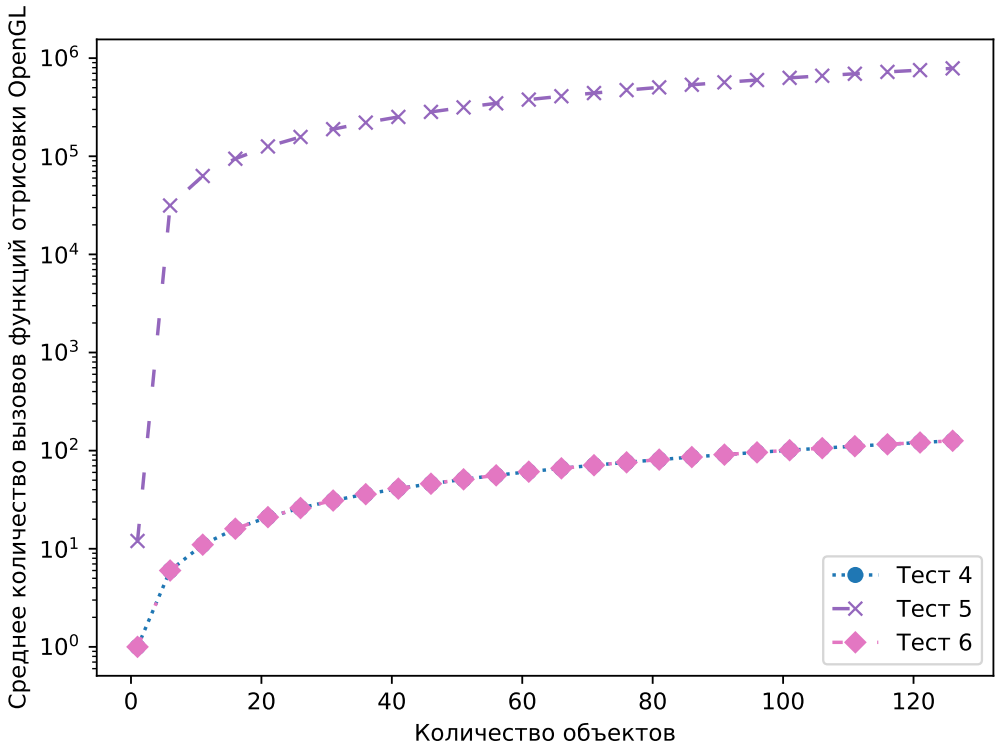

Рисунок 24 – Зависимость количества вызовов OpenGL-функций отрисовки от количества объектов в опытах 4 – 6

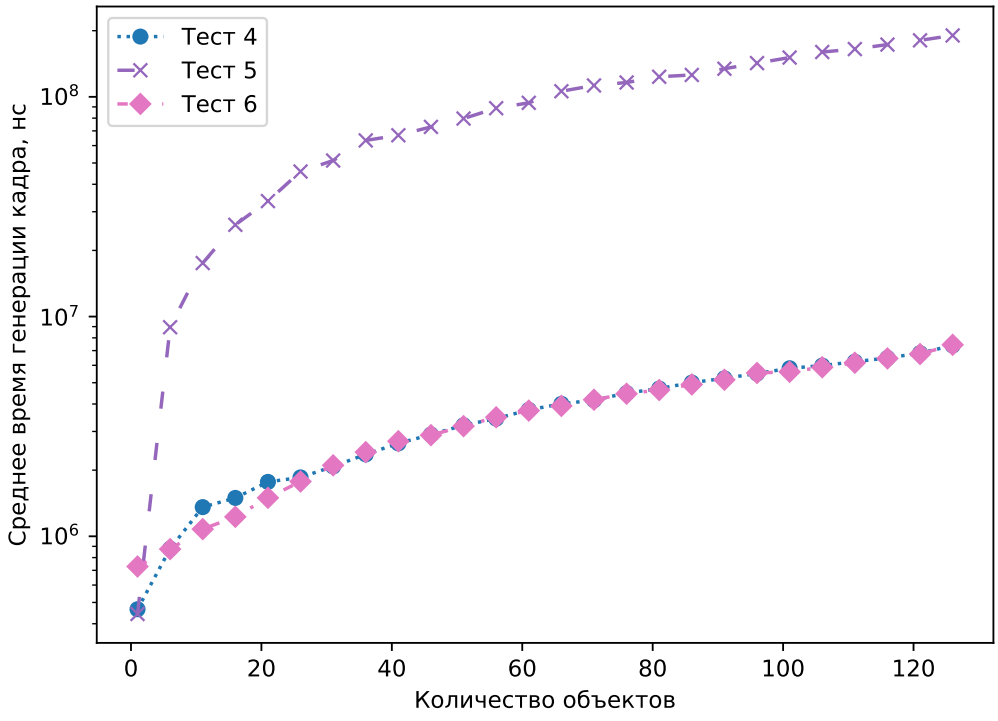

Рисунок 25 – Зависимость времени генерации кадра от количесва объектов в опытах 4 – 6

Вывод

ЗАКЛЮЧЕНИЕ

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Куров А. В. Курс лекций по дисциплине «Компьютерная графика». 2023.
2. Методы 3D моделирования [Электронный ресурс]. – URL: https://studbooks.net/1413342/bzhd/metody_modelirovaniya (дата обращения: 15.07.2023).
3. Методы твердотельного моделирования [Электронный ресурс]. – URL: https://studref.com/605454/tehnika/metody_tverdotelnogo_modelirovaniya (дата обращения: 15.07.2023).
4. Грегори Дж. Игровой движок. Программирование и внутреннее устройство. Третье издание. – СПб.: Питер, 2022 – 1136 с.
5. Mamatov M., Nuritdinov J. Some Properties of the Sum and Geometric Differences of Minkowski // Journal of Applied Mathematics and Physics. 2020. Т. 8. С. 2241 – 2255.
6. IMPLEMENTING GJK (2006) [Электронный ресурс]. – URL: https://caseymuratori.com/blog_0003 (дата обращения: 16.07.2023).
7. Gutiérrez A. Mario A., Vexo Frédéric, Thalmann Daniel. Computer Graphics // Stepping into Virtual Reality. Springer Nature Switzerland, 2023. С. 15 – 54.
8. Phong Bui Tuong. Illumination for computer generated pictures // Communications of the ACM. 1975. Т. 18. С. 311 – 317.
9. Kurihara Takayuki, Takaki Yasuhiro. Shading of a computer-generated hologram by zone plate modulation // Optics express. 2012. Т. 20. С. 3529 – 40.
10. Роджерс Д. Алгоритмические основы машинной графики. Пер. с англ. – М.: Мир, 1989 – 512 с.
11. Carmack J. QuakeCon 2013: The Physics of Light and Rendering [Электронный ресурс]. – URL: https://youtu.be/P6UKhR0T6cs (дата обращения: 11.12.2023).

12. OpenGL. The Industry’s Foundation for High Performance Graphics [Электронный ресурс]. – URL: https://www.opengl.org/ (дата обращения: 11.12.2023).

13. DirectX graphics and gaming [Электронный ресурс]. – URL: https://learn.microsoft.com/en-us/windows/win32/directx (дата обращения: 11.12.2023).

14. Vulkan Documentation [Электронный ресурс]. – URL: https://docs.vulkan.org/spec/latest/index.html (дата обращения: 11.12.2023).

15. Glad. Multi-Language GL/GLES/EGL/GLX/WGL Loader-Generator based on the official specs [Электронный ресурс]. – URL: https://glad.dav1d.de (дата обращения: 11.12.2023).

16. GLFW [Электронный ресурс]. – URL: https://www.glfw.org (дата обращения: 11.12.2023).

17. GLM. OpenGL Mathematics [Электронный ресурс]. – URL: https://glm.g-truc.net/0.9.2/api/index.html (дата обращения: 11.12.2023).

18. GDB: The GNU Project Debugger [Электронный ресурс]. – URL: https://www.sourceware.org/gdb (дата обращения: 11.12.2023).

19. RenderDoc [Электронный ресурс]. – URL: https://renderdoc.org/ (дата обращения: 11.12.2023).

20. Neovim [Электронный ресурс]. – URL: https://neovim.io (дата обращения: 11.12.2023).

21. AMD Ryzen™ 7 4700U [Электронный ресурс]. – URL: https://www.amd.com/en/product/9096 (дата обращения: 12.12.2023).

22. NixOS [Электронный ресурс]. – URL: https://nixos.org (дата обращения: 12.12.2023).