|  |  |
| --- | --- |
| Gerb-BMSTU_01 | **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 КУРСОВОМУ ПРОЕКТУ***

***НА ТЕМУ:***

Программа моделирования взлета ракеты с земли.

Студент \_\_**ИУ7-51 Б\_\_\_\_**  **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_** Зейналов Зейнал Габибович

(Группа) (Подпись, дата) (И.О.Фамилия)

Руководитель курсового проекта **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_** Куров Андрей.Владимирович

(Подпись, дата) (И.О.Фамилия)

*2019 г.*

Оглавление

[Введение 4](#_Toc27249162)

[1. Аналитическая часть 5](#_Toc27249163)

[1.1. Постановка задачи 5](#_Toc27249164)

[1.2. Формализация объектов синтезируемой сцены 6](#_Toc27249165)

[1.3. Критерии выбора алгоритма 7](#_Toc27249166)

[1.4. Ограничения 7](#_Toc27249167)

[1.5. Алгоритмы удаления невидимых линий и поверхностей 7](#_Toc27249168)

[1.6. Алгоритмы закрасок 10](#_Toc27249169)

[1.7. Алгоритмы построения теней 13](#_Toc27249170)

[1.8. Алгоритм генерации огня 15](#_Toc27249171)

[1.9. Алгоритм удаления нелицевых граней 16](#_Toc27249172)

[1.10. Алгоритм построения конусов, цилиндров, усеченных конусов 17](#_Toc27249173)

[2. Конструкторский раздел 20](#_Toc27249174)

[2.1 Проектирование ПО 20](#_Toc27249175)

[2.2 Схемы алгоритмов 21](#_Toc27249176)

[2.3 Диаграмма классов 24](#_Toc27249177)

[Вывод 25](#_Toc27249178)

[3 Технологический раздел 26](#_Toc27249179)

[3.1 Средства реализации 26](#_Toc27249180)

[3.2 Инструкция по запуску программного обеспечения 26](#_Toc27249181)

[3.3 Описание интерфейса программы 26](#_Toc27249182)

[3.4 Описание основных этапов реализации 29](#_Toc27249183)

[Вывод 30](#_Toc27249184)

[4 Исследовательский раздел 31](#_Toc27249185)

[4.1 Зависимость времени рендеринга от числа объектов и количества граней. 31](#_Toc27249186)

[4.2 Зависимость времени взлета ракеты 32](#_Toc27249187)

[Вывод 32](#_Toc27249188)

[Заключение 33](#_Toc27249189)

[Литература 34](#_Toc27249190)

# Введение

Компьютерная графика неразрывно связана с нашей повседневной жизнью. Важной составляющей в использование компьютеров является обработка и синтез изображения таким образом, чтобы человеческий глаз не обнаруживал разницы между переданной картиной и реальностью. Ввиду растровой природы дисплеев появилась необходимость разрабатывать алгоритмы, которые позволят достичь подобного успеха. Ежегодно с ростом производительности современных устройств растут и требования, предъявляемые к реалистичности смоделированного трехмерного изображения. Такие модели используются в графике, компьютерных играх, создании сценических эффектов, виртуальной и дополненной реальности.

Целью курсового проекта является разработка программы, моделирующей в реальном времени взлет космической ракеты с земли.

В рамках реализации проекта должны быть решены следующие задачи:

* Изучение и анализ алгоритмов компьютерной графики, использующихся для создания реалистичной модели взаимно перекрывающихся объектов и выбор наиболее подходящего для решения поставленной задачи.
* Проектирование архитектуры программы и ее интерфейса.
* Реализация выбранных алгоритмов и структур данных.
* Проведение исследования на основе разработанной программы

1. **Аналитическая часть**

В данном разделе представлены постановка задачи, критерии выбора алгоритмов, ограничения, анализ алгоритмов и методов, средства для реализации поставленной задачи.

* 1. **Постановка задачи**

Необходимо смоделировать взлет ракеты с земли с учетом теней при построении динамического изображения. Данная программа предоставляет возможность моделирования взлета ракеты для космических станций, а также развитие данного проекта до полноценной компьютерной игры. Формализацией задачи служит рис.1 представляющий входные и выходные данные на idef0 диаграмме.

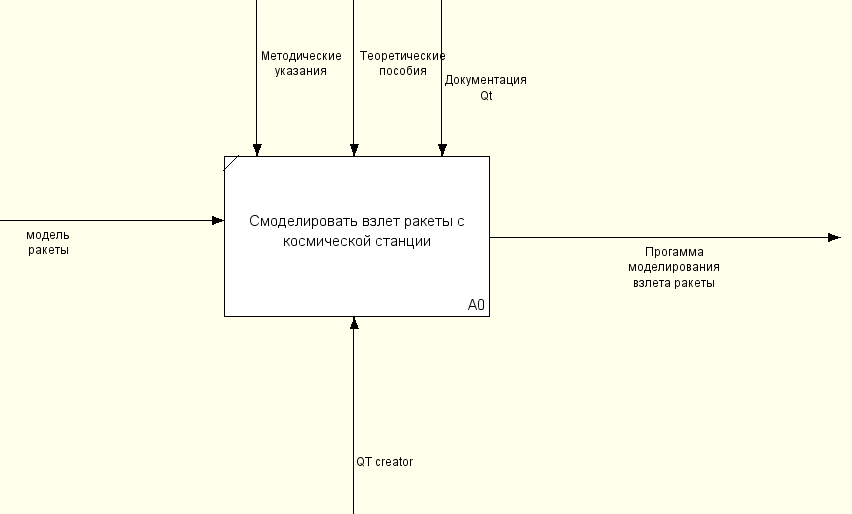


Рисунок 1 – idef0 диаграмма поставленной задачи.

* 1. **Формализация объектов синтезируемой сцены**

Существует несколько видов геометрических моделей [1]:

1. Каркасная модель.
2. Поверхностная модель.
3. Объемная модель.

Объекты сцены наилучшим образом описываются через поверхностную модель, так как каркасные модели не обладают достаточной реалистичностью, а объемные модели слишком информативны, что будет затруднять вычисления и будет более затратным по памяти [1]. Для представления объектов сцены используются такие геометрические примитивы как точка и треугольник.

Сцена состоит из следующих объектов.

Ракета, стартовый стол, а они представляются из конусов, цилиндров и усеченных конусов, которые в свою очередь представляются из треугольников.

* Конусы, цилиндры, усеченные конусы – полигональные объекты, в основе которых лежат правильные многоугольники, составленные из треугольников (рисунок 2).
* Космическая ракета, составленная из конусов, цилиндров и усеченных конусов.
* Стартовая площадка, составленная из цилиндра.
* Источник света - представляет собой материальную точку, из которой исходят лучи света во все стороны.

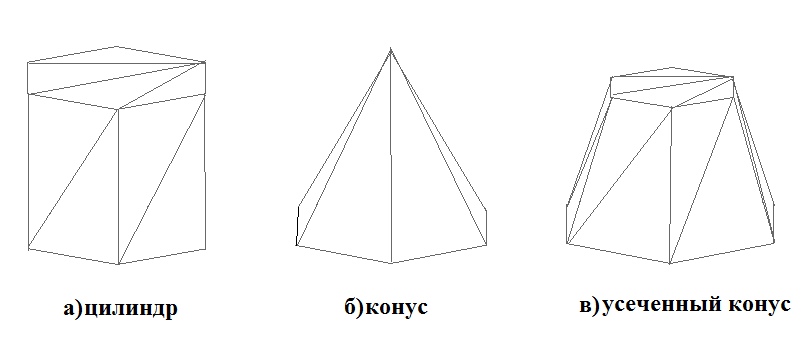


Рисунок 2 - Представление объектов из 6 граней с помощью треугольников

* 1. **Критерии выбора алгоритма**

Критериями выбора алгоритма служат простота реализации, скорость работы и возможность его оптимизации.

* 1. **Ограничения**

Ограничениями реализации является недопустимость использования сторонних библиотек.

* 1. **Алгоритмы удаления невидимых линий и поверхностей**

Для выбора подходящего алгоритма построения изображения, необходимо провести обзор известных алгоритмов и осуществить выбор наиболее подходящего для реализации поставленной задачи.

**Алгоритм Робертса**

Алгоритм Робертса[1] представляет собой первое известное решение задачи об удалении невидимых линий. Это метод, работающий в объектном пространстве. В соответствии с алгоритмом, прежде всего удаляются из каждого тела те ребра или грани, которые перекрываются самим телом. Затем каждое из видимых ребер каждого тела сравнивается с каждым из оставшихся тел для определения того, какая его часть или части, если таковые есть, перекрываются этими телами.

Преимущества данного алгоритма в том, что математические методы, используемые в нем просты, мощны и точны. Более поздние реализации алгоритма, например, использующие предварительную сортировку вдоль оси z, демонстрируют почти линейную зависимость от числа объектов.

Минус этого алгоритма в том, что теоретически вычислительная трудоемкость алгоритма Робертса растет как квадрат числа объектов. Реализация оптимизированных алгоритмов весьма сложна.

**Алгоритм трассировки лучей**

В этом методе для каждого пикселя картинной плоскости определяется ближайшая к нему грань, для чего через этот пиксель выпускается луч, находятся все его пересечения с гранями и среди них выбирается ближайшая.

К достоинствам данного алгоритма можно отнести возможность получения изображения гладких объектов без аппроксимации их примитивами (например, треугольниками). Вычислительная сложность метода линейно зависит от сложности сцены. Нетрудно реализовать наложение света и тени на объекты. Полученное изображение получается очень реалистичным, этот метод отлично подходит для создания фотореалистичных картин.

Серьёзным недостатком алгоритма трассирования является производительность. Для получения изображения необходимо создавать огромное число лучей, проходящих через сцену и отражаемых от объекта. Это приводит к существенному снижению скорости работы программы.

**Алгоритм Варнока**

Алгоритм Варнока[1] работает в пространстве изображений. В основу алгоритма положен принцип разбиения области рисунка на более мелкие подобласти (окна). Для каждой подобласти (окна) определяются связанные с ней многоугольники и те из них, видимость которых определить "легко", изображаются на экране. В противном же случае разбиение повторяется, и для каждой из вновь полученных подобластей рекурсивно применяется процедура принятия решения. Предполагается, что с уменьшением размеров области ее перекрывает все меньшее и меньшее количество многоугольников. Считается, что в пределе будут получены области, содержащие не более одного многоугольника, и решение будет принято достаточно просто. Если же в процессе разбиения будут оставаться области, содержащие не один многоугольник, то следует продолжать процесс разбиения до тех пор, пока размер области не станет совпадать с одним пикселем. В этом случае для полученного пикселя необходимо вычислить глубину (значение координаты Z) каждого многоугольника и визуализировать тот из них, у которого максимальное значение этой координаты.

**Алгоритм, использующий z-буфер**

Данный алгоритм удаления невидимых поверхностей является одним из простейших. Этот алгоритм работает в пространстве изображения. Здесь обобщается идея о буфере кадра. Буфер кадра используется для заполнения атрибутов (интенсивности) каждого пикселя в пространстве изображения[1]. Наряду с буфером кадра вводится Z-буфер, представляющий собой специальный буфер глубины, в котором запоминаются координаты Z (глубина) каждого видимого пикселя в пространстве изображения. В процессе работы глубина (значение координаты Z) каждого нового пикселя, который надо занести в буфер кадра, сравнивается с глубиной того пикселя, который уже занесен в Z-буфер. Если это сравнение показывает, что новый пиксель расположен ближе к наблюдателю, чем пиксель, уже находящийся в буфере кадра, то новый пиксель заносится в буфер кадра. Помимо этого, производится корректировка Z-буфера: в него заносится глубина нового пикселя. Если же глубина (значение координаты Z) нового пикселя меньше, чем хранящегося в буфере, то никаких действий производить не надо. В сущности, алгоритм для каждой точки (x,y) находит наибольшее значение функции Z(x,y).

Несмотря на свою простоту, этот алгоритм позволяет удалять сложные поверхности и визуализировать пересечения таких поверхностей. Сцены могут быть произвольной сложности, а поскольку размеры изображения ограничены размером экрана дисплея, то трудоемкость алгоритма имеет линейную зависимость от числа рассматриваемых поверхностей. Элементы сцены заносятся в буфер кадра в произвольном порядке, поэтому в данном алгоритме не тратится время на выполнение сортировок, необходимых в других алгоритмах.

**Выбор алгоритма удаления невидимых линий и поверхностей**

Таблица 1 – Сравнение алгоритмов удаления невидимых поверхностей

|  |  |  |  |
| --- | --- | --- | --- |
| Алгоритм | Простота реализации | Скорость работы | Возможность оптимизаций |
| Алгоритм Робертса | **☒** | **☒** | **☑** |
| Алгоритм трассировки лучей | **☒** | **☒** | **☑** |
| Алгоритм Варнока | **☑** | **☒** | **☒** |
| Алгоритм, использующий z-буфер | **☑** | **☑** | **☑** |

* 1. **Алгоритмы закрасок**

Существуют три основных алгоритма, позволяющих закрасить полигональную модель: простая закраска, закраска по Гуро и закраска по Фонгу.

**Простая закраска**

Суть данного алгоритма заключается в том, что для каждой грани объекта находится вектор нормали, и с его помощью в соответствии с выбранной моделью освещения вычисляется значение интенсивности, с которой закрашивается вся грань. [2]

Данный метод закраски обладает большим быстродействием, однако все пиксели грани имеют одинаковую интенсивность, и сцена выглядит нереалистично.

**Закраска по Гуро**

Метод Гуро является одним из способов устранения дискретности интенсивностей закрашивания.

Данный алгоритм предполагает следующие шаги:

* Вычисление векторов нормалей к каждой грани.
* Вычисление векторов нормали к каждой вершине грани путем усреднения нормалей к граням (см. рисунок 3)
* Вычисление интенсивности в вершинах грани.
* Интерполяция интенсивности вдоль ребер грани.
* Линейная интерполяция интенсивности вдоль сканирующей строки.

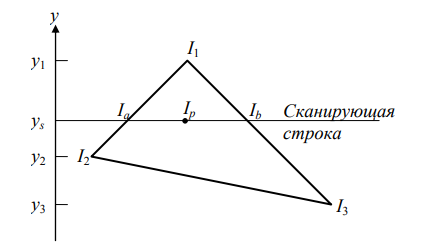


Рисунок 3. Интерполяция интенсивностей

Интерполяция производится следующим образом: для всех ребер запоминается начальная интенсивность, а также изменение интенсивности при каждом единичном шаге по координате y. Заполнение видимого интервала на сканирующей строке производится путем интерполяции между значениями интенсивности на двух ребрах, ограничивающих интервал (рис. 3) [2]

Закраска по Гуро хорошо сочетается с диффузным отражением. Данный метод интерполяции обеспечивает лишь непрерывность значений интенсивности вдоль границ многоугольников, но не обеспечивает непрерывность изменения интенсивности, следовательно, возможно появление полос Маха.

На рисунке 4 Nv – нормаль к вершине – усреднённое значение нормалей N1…NN.

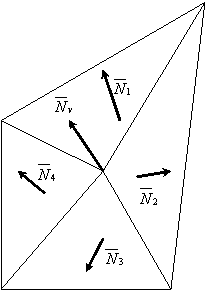


Рисунок 4. Усреднение нормалей в вершинах

Закраска по Фонгу

Закраска по Фонгу требует больших вычислительных затрат, чем Гуро. При такой закраске, в отличие от метода Гуро, вдоль сканирующей строки интерполируется значение вектора нормали, а не интенсивности. Используя закраску по Фонгу, можно достичь лучшей локальной аппроксимации кривизны поверхности.

Шаги алгоритма:

* Вычисление векторов нормалей в каждой грани.
* Вычисление векторов нормали к каждой вершине грани.
* Интерполяция векторов нормалей вдоль ребер грани.
* Линейная интерполяция векторов нормалей вдоль сканирующей строки.
* Вычисление интенсивности в очередной точке сканирующей строки.

Изображение получается более реалистичным. Закраска по Фонгу хорошо сочетается с зеркальным отражением.

В методе закраски, разработанном Фонгом, используется интерполяция вектора нормали N к поверхности вдоль видимого интервала на сканирующей строке внутри многоугольника, а не интерполяция интенсивности [2]. Интерполяция выполняется между начальной и конечной нормалями, которые сами тоже являются результатами интерполяции вдоль ребер многоугольника между нормалями в вершинах. Нормали в вершинах, в свою очередь, вычисляются так же, как в методе закраски, построенном на основе интерполяции интенсивности (формулы 2, 3, 4).

Основными недостатками данного алгоритма являются ресурсоемкость и большая вычислительная сложность.

**Выбор алгоритма закраски**

На рисунке 5 наглядно показаны различия рассмотренных методов закраски.



Рисунок 5 – Методы закрасок (простая, Гуро, Фонга)

Таблица 2 - Сравнение алгоритмов закраски

|  |  |  |  |
| --- | --- | --- | --- |
| Алгоритм | Простота реализации | Скорость работы | Возможность оптимизаций |
| Простая закраска | **☑** | **☑** | Не требуется |
| Закраска по Гуро | **☒** | **☑** | **☑** |
| Закраска по Фонгу | **☒** | **☒** | **☑** |

* 1. **Алгоритмы построения теней**

**Построение теней в алгоритме трассировки лучей.**

Изначально алгоритм трассировки лучей был задуман как средство

удаления невидимых поверхностей, но затем он был усовершенствован: были добавлены возможности построения теней и учтены эффекты отражения и пропускания света. Таким образом, метод трассировки лучей позволяет получать изображения наивысшей реальности, моделируя прозрачность, отражения, преломления, другие оптические эффекты.

Кратко рассмотрим основные принципы работы алгоритм трассировки лучей. Наиболее часто используется обратная трассировка лучей—от наблюдателя к сцене, которая позволяет существенно уменьшить вычислительные затраты по сравнению с прямой трассировкой. Лучи трассировки проводятся из точки наблюдения через каждый пиксель экрана и проецируются на модель. Пересечение лучей трассировки и объектов определяет видимые поверхности. Кроме того, из точки пересечения испускаются дополнительные лучи. Если поверхность отражает свет, то генерируется отраженный луч. Если поверхность пропускает свет, то генерируется пропущенный луч с учетом того, что при переходе из одной среды в другую луч меняет направление (преломляется). Пути этих лучей отслеживаются и при пересечении их с объектами сцен формируются новые лучи. Тени в алгоритме трассировки лучей учитываются следующим образом. Из каждой точки пересечения луча трассировки с поверхностью строятся дополнительные лучи по направлению к каждому источнику света. Если такой луч пересекает на своем пути какую-нибудь поверхность, значит, эта точка находится в тени. Таким образом, определяется цвет и его интенсивность для каждой точки изображения [6].

**Построение теней с использованием алгоритма z-буфера**

Алгоритм удаления невидимых поверхностей с использованием z-буфера легко может быть модифицирован и для того, чтобы учесть эффект отбрасывания теней. При построении теней с использованием алгоритма z-буфера выполняется два прохода: один - относительно источника света, другой – относительно наблюдателя. Для этого выделяется отдельный "теневой" z-буфер. Первый проход необходим для того, чтобы определить, какие точки видны со стороны источника света. При втором проходе сцена визуализируется из положения наблюдателя с учетом того, что точки, которые оказались невидимыми со стороны источника света, находятся в тени. Таким образом, алгоритм работает в два этапа. При первом проходе сцена рассчитывается при совмещении точки наблюдения с положением источника света. Значения глубины пикселей для данного вида заносятся в "теневой" z- буфер, а значения интенсивности не учитываются. При втором проходе сцена строится из точки, в которой на самом деле находится наблюдатель. При сканировании каждой поверхности значение ее глубины каждого ее пикселя сравнивается со значением глубины в z-буфере. Если поверхность видима, то необходимо проверить, видима ли данная точка со стороны источника света. Для этого координаты точки x, y, z из вида наблюдателя линейно преобразуются в координаты x', y', z' на виде из источника света. Проверка на видимость осуществляется сравнением значения, которое хранится в "теневом" z-буфере для координат x', y', и значения z'. Если точка невидима для источника света (значение в "теневом" z-буфере больше значения z'), значит, она находится в тени и ее свечение вычисляется с учетом затенения. Если же точка видима из положения источника света, то она изображается без изменений [6].

**Выбор алгоритма построения теней**

Таблица 3 - Сравнение алгоритмов построения теней

|  |  |  |  |
| --- | --- | --- | --- |
| Алгоритм | Простота реализации | Скорость работы | Возможность оптимизаций |
| Построение теней с использованием теневого буфера | **☑** | **☑** | **☑** |
| Построение теней в алгоритме трассировки лучей. | **☒** | **☑** | **☑** |

* 1. **Алгоритм генерации огня**

Для реализации огня потребуется дополнительный буфер кадра с указанными размерами ширины и высоты огня. В основе данного алгоритма используется карта цветовых высот, представленная на рисунке 6.

https://habrastorage.org/webt/zb/-j/42/zb-j42e-s0dm4uo-yhwazmuzbcy.png

Рисунок 6 – цветовая карта высот.

Массив размером с выделенный буфер заполняется 37 значениями в интервале от 0 до 36. Каждое значение связывается с цветом от белого до чёрного, и захватывает по дороге между ними жёлтый, оранжевый и красный. Идея заключается в моделировании температуры частицы пламени, которая по мере удаления от источника постепенно охлаждается. Буфер кадра инициализируется полностью чёрным (заполненным нулями) с единственной белой строкой белых пикселей вверху (36), которая является «источником» пламени. При каждом обновлении экрана «тепло» опускается вниз. Для каждого пикселя в буфере кадра вычисляется новое значение. Каждый пиксель обновляется с учётом значения, расположенного непосредственно над ним. В коде нижний левый угол это нулевой индекс массива, а верхний правый угол имеет индекс FIRE\_HEIGHT \* FIRE\_WIDTH — 1 ( Ширина огня \* высоту огня – 1).

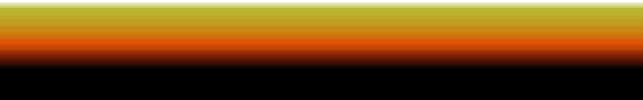


Рисунок 7 - Изображения огня без добавления случайности

Для придания реалистичности используется генерация случайных чисел, т.е распространение пикселей вниз и по ширине имеет случайный характер. При отображении огня получаем картину, представленную на рисунке 8 [5].

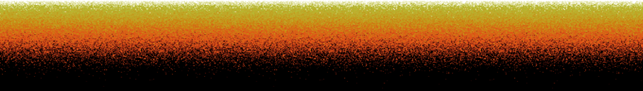


Рисунок 8 – Изображение огня

* 1. **Алгоритм удаления нелицевых граней**

Для оптимизации работы алгоритмов закраски необходимо прежде удалить нелицевые грани. Идея заключается в том, что с точки зрения наблюдателя, некоторые грани объекта оказываются невидимыми и их можно определить с помощью скалярного произведения векторов взгляда и нормали к этой грани. В случае, если скалярное произведение оказывается больше нуля, то угол между вектором взгляда и вектором нормали > 90 градусов, значит грань – невидима, соответственно растеризацию для этой грани проводить не нужно. Для вычисления нормали так же необходимо учитывать направление обхода граней объекта. В зависимости от направления обхода можно получить либо внутреннюю нормаль, либо внешнюю (рис. 9).

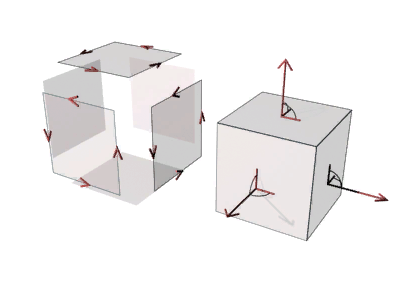


Рисунок 9. Внешние нормали к граням куба

* 1. **Алгоритм построения конусов, цилиндров, усеченных конусов**

Конусы, цилиндры, и усеченные конусы имеют одинаковую основу. Построение этих геометрических объектов можно задать с помощью 5 входных параметров:

* Радиус верхней окружности
* Радиус нижней окружности
* Высота
* Точка – центр объекта в трехмерном пространстве
* Количество граней объекта

В первую очередь необходимо соединить множество точек, задающих верхнюю и нижнюю окружности. Затем, для каждой точки первой окружности соединить с соответствующей точкой второй окружности. В итоге, задача построения усеченного конуса сводится к задаче построения правильных многоугольников (рис.10) с разными радиусами. Конусы и цилиндры являются частными случаями усеченных конусов, при равенстве радиусов друг другу и равенстве одного из радиусов нулю соответственно. Идея построения правильных многоугольников состоит в том, что через угол на расстоянии r откладывается точка. Угол высчитывается как , где n – количество граней.

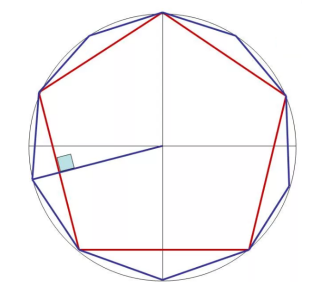
**

Рисунок 10. Изображение правильных многоугольников

* 1. Алгоритмы Моделей освещения

Все модели освещения делятся на два вида: глобальные и локальные. Глобальные модели учитывают возможности отражения и преломления света от объектов, не являющихся прямыми источниками освещения, поэтому они требуют значительных затрат.

Существуют более простые, локальные модели освещения, которые учитывают только свет от источника. Именно этот тип моделей используются в данном курсовом проекте.

**Модель Ламберта**

Данная модель является простейшей моделью освещения, так как учитывает только идеальное диффузное отражение света от тела [2]. На рисунке 11 показано, что согласно этой модели, освещенность в точке определяется только плотностью света в точке поверхности, а она линейно зависит от косинуса угла падения. При этом положение наблюдателя не имеет значения, т.к. диффузно отраженный свет рассеивается равномерно по всем направлениям.

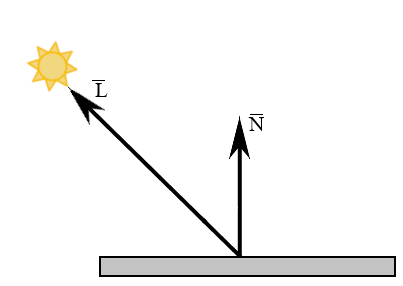


Рисунок 11 - Модель освещения Ламберта

Пусть

– вектор от точки до источника,

– вектор нормали,

I – результирующая интенсивность света в точке,

I0 – интенсивность источника,

Kd – коэффициент диффузного освещения.

Формула расчёта интенсивности имеет следующий вид:

Из формулы (5) следует главный недостаток модели Ламберта – одинаковая интенсивность во всех точках, принадлежащих одной грани.

## Конструкторский раздел

В данном разделе будут рассмотрены проектирование ПО, диаграмма классов, схемы алгоритмов для Z – буфера, простой модели освещения, удаления невидимых граней.

Программа должна обладать следующей функциональностью:

1. Визуализировать трехмерную сцену, состоящую из объектов, представленных в пункте 1.2, в режиме реального времени.
   1. В зависимости от положения источника света тени должны менять свою длину, угол и положение.
   2. Ракета при нажатии на кнопку должна взлетать с сопровождающими эффектами (Огонь из сопла ракеты)
2. Предоставлять в интерфейсе возможность задавать следующие настройки:
   1. Углы поворота ракеты относительно осей х и у.
   2. Углы поворота сцены.
   3. Выбор цвета источника освещения.
   4. Использование/Неиспользование теней при построении изображения

### Проектирование ПО

На рисунке 12 приведена детализация постановки задачи, приведенной в п. 1.1.

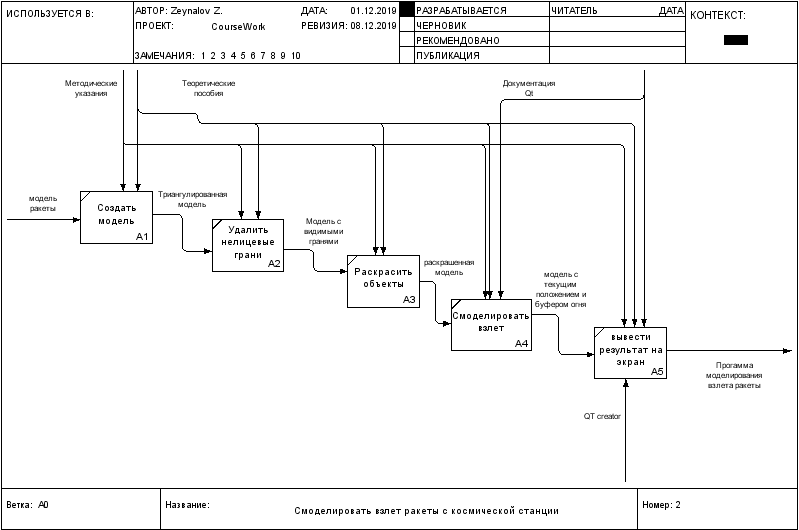


Рисунок 12. Детализация поставленной задачи

### Схемы алгоритмов

На рисунках 13 - 15 приведены схемы алгоритмов.



Рисунок 13. Схема алгоритма удаления нелицевых граней

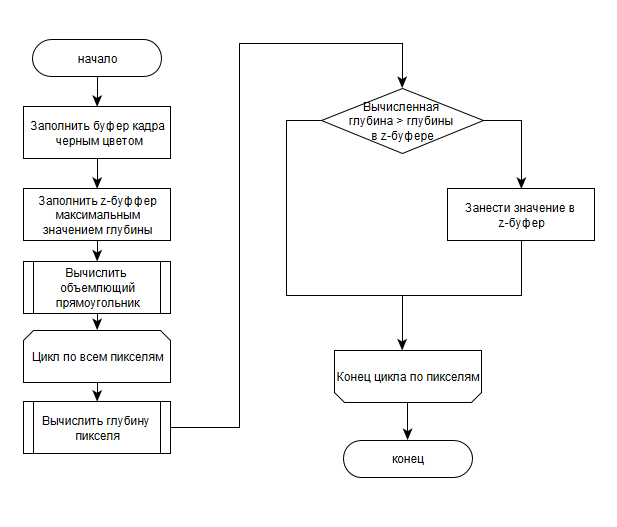


Рисунок 14 - Схема алгоритма, использующего z-буфер

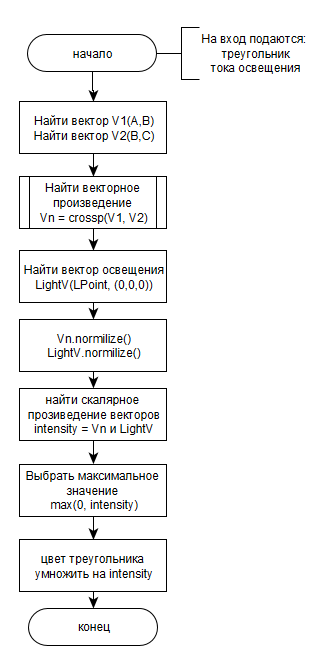


Рисунок 15 - Схема алгоритма простой модели освещения

Цвета представляются с помощью трех компонент – red, green, blue, значения которых лежат в диапазоне от 0 до 255. Вычисленное значение интенсивности лежит в диапазоне от 0 до 1. Умножение интенсивности на цвет происходит путем умножения каждой компоненты на значение интенсивности.

### Диаграмма классов

На рисунке 16 приведена диаграмма классов.

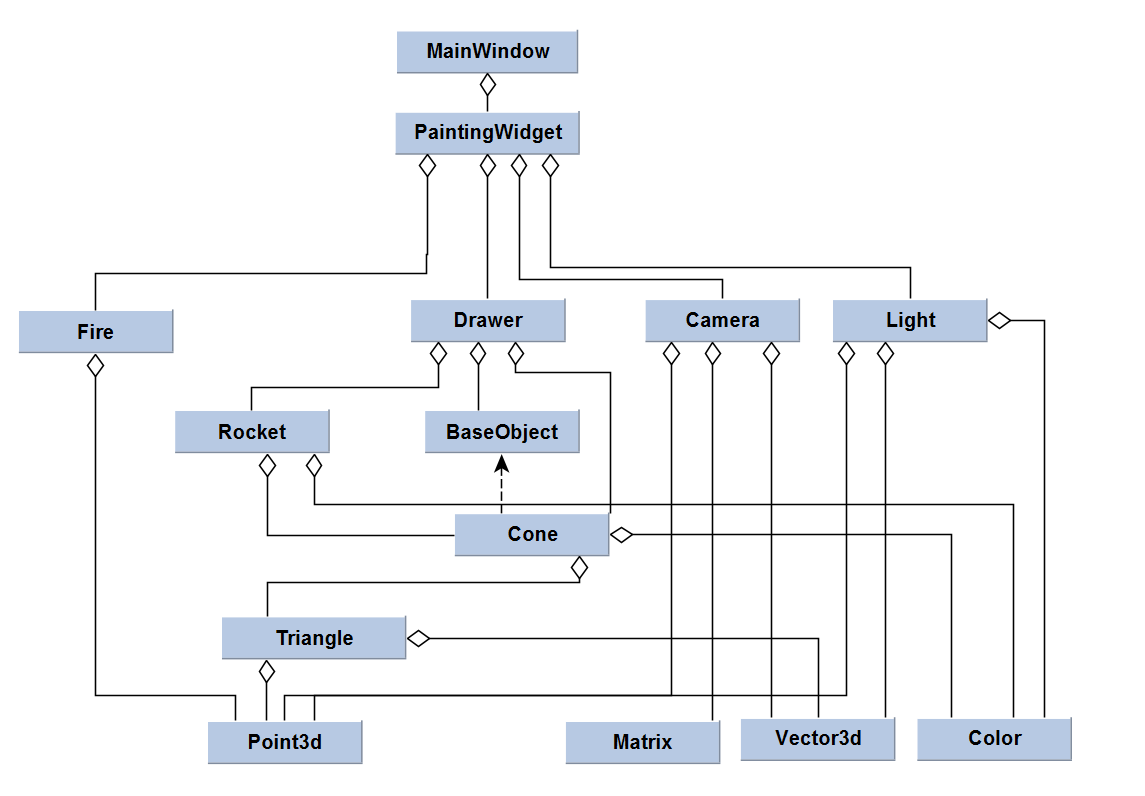


Рисунок 16 - Схема классов программы

Разработанная программа состоит из следующих классов:

* Базовые математические классы
  + Matrix – класс матриц;
  + Vector3D – класс векторов трехмерного пространства;
  + Point3D – класс точки трехмерного пространства.
* Классы объектов
  + BaseObject – базовый класс объектов с возможностью перемещения по сцене;
  + Cone – класс конуса с возможностью задания радиусов верхнего и нижнего оснований, высоты и числа боковых граней.
  + Rocket – класс ракеты, состоящий из объектов класса Cone, с возможностью добавления модулей, перемещения по сцене, поворотов, задания точки-центра в пространстве, масштаба, количества граней .
  + Triangle – класс треугольника, состоящий из 3 точек и вектора нормали к данному треугольнику.
* Вспомогательные классы сцены
  + Camera – класс камеры с возможностью перемещения по сцене;
  + Light – класс источника освещения с возможностью перемещения по сцене и изменения цвета.
* Классы интерфейса
  + RenderWidget – класс виджета сцены для вывода буфера кадра;
  + MainWindow – класс главного окна сцены.
* Классы анимации
  + Fire – класс огня, состоящий из буфера кадра;

### Вывод

В данном разделе было спроектировано ПО, рассмотрены схемы алгоритмов, диаграмма классов, а также их назначение.

## Технологический раздел

В данном разделе рассмотрен выбор средств реализации, описаны основные этапы программной реализации.

### Средства реализации

Для написания данного курсового проекта в качестве языка программирования был выбран С++ (компилятор g++) Этот язык поддерживает объектно-ориентированную модель разработки, что позволяет четко структурировать программу и легко модифицировать отдельные ее компоненты независимо от других. Язык C++ позволяет эффективно использовать ресурсы системы благодаря широкому набору функций и классов из стандартной библиотеки. Язык программирования С++ содержит библиотеку «thread.h», которая позволяет использовать многопоточность. Для реализации проекта выбрана среда программирования Qt Creator 4.8.2[5] Данная среда обладает редактором кода, отладчиком а также широким функционалом по работе с интерфейсами.

### Инструкция по запуску программного обеспечения

Для запуска программного обеспечения необходима среда разработки Qt Creator версии не ниже 4.8.2 и компилятор g++ . Запуск должен осуществляться в режиме «Выпуск».

### Описание интерфейса программы

При запуске программы (рисунок 17) на виджете сцены отображается система координат, ось z которой направлена на нас. Для управления сценой используется панель, в правой части экрана. (рисунок 18).

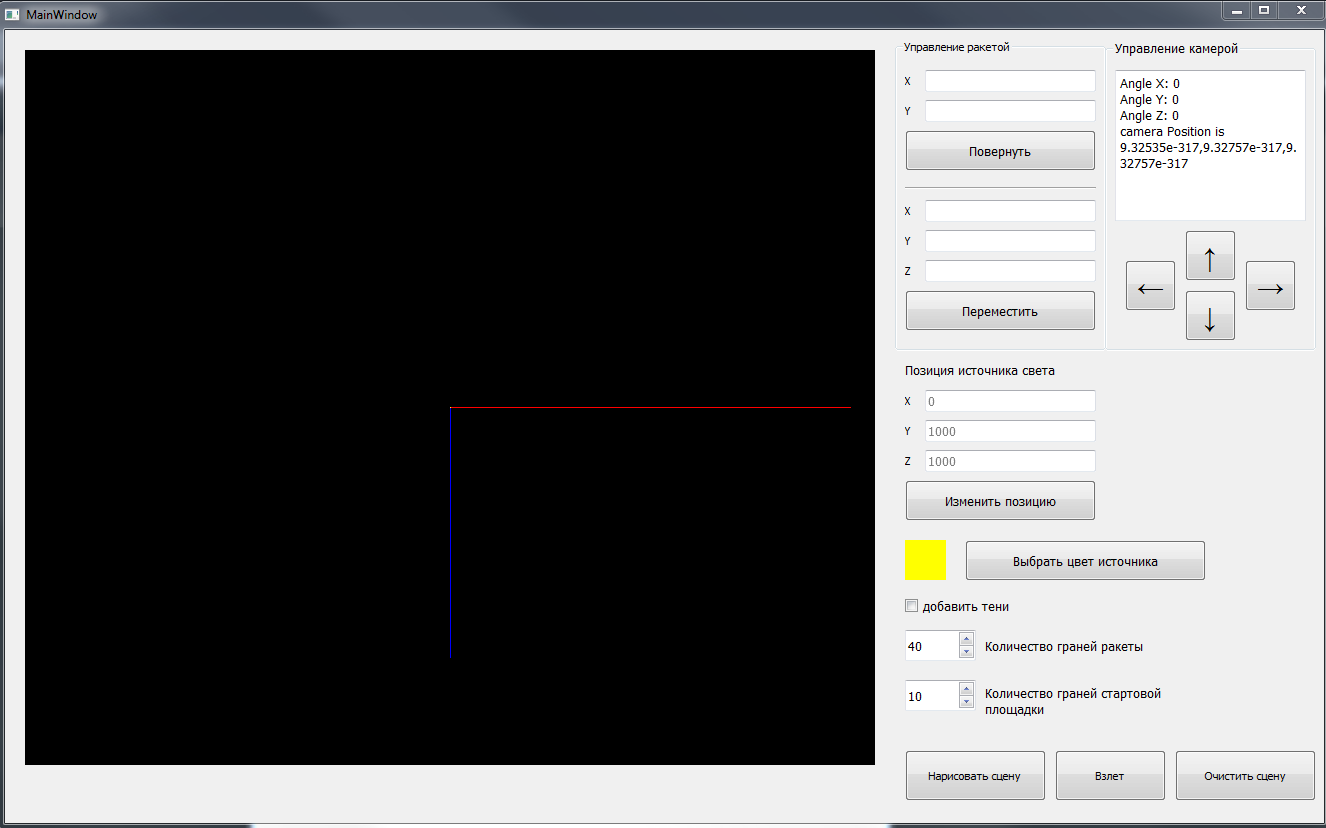


Рисунок 17 - Интерфейс программы

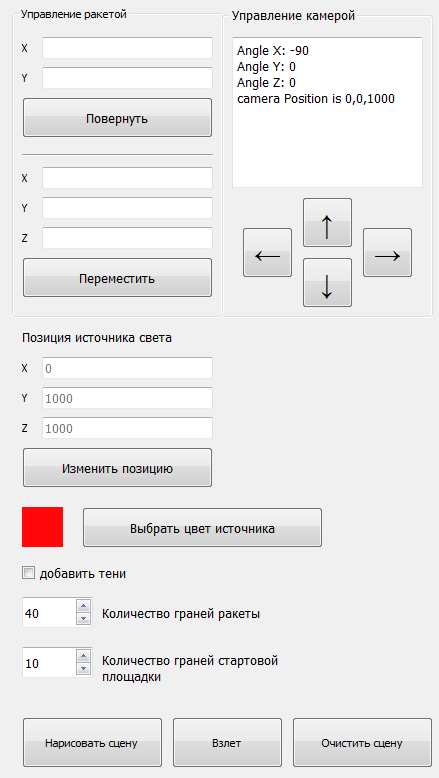


Рисунок 18 - Панель управления

На панели присутствует возможность изменять углы поворотов, а так же положения ракеты относительно текущего положения. В правой части находится окно выдачи сообщений о текущем повороте и положении камеры, а также кнопки для поворотов сцены. Можно изменять положение источника освещения и его цвет. Ниже расположено поле для выбора использования теней. Количество граней у объектов сцены так же можно изменить. Для взлета ракеты необходимо нажать на кнопку “Взлет”. Для очистки сцены нужно нажать на кнопку “Очистить сцену”. Получить изображение ракеты можно с помощью нажатия кнопки “Нарисовать сцену”. После нажатия, отобразится сцена (рисунок 19), управлять которой можно с помощью кнопок на панели или с помощью клавиш: W – повернуть сцену вниз вокруг оси x (красная), S - повернуть сцену вверх вокруг оси x , A – повернуть сцену влево вокруг оси y (синяя), D повернуть сцену вправо вокруг оси y, E и Z – против и по часовой стрелке вокруг оси z (желтая) соответственно.

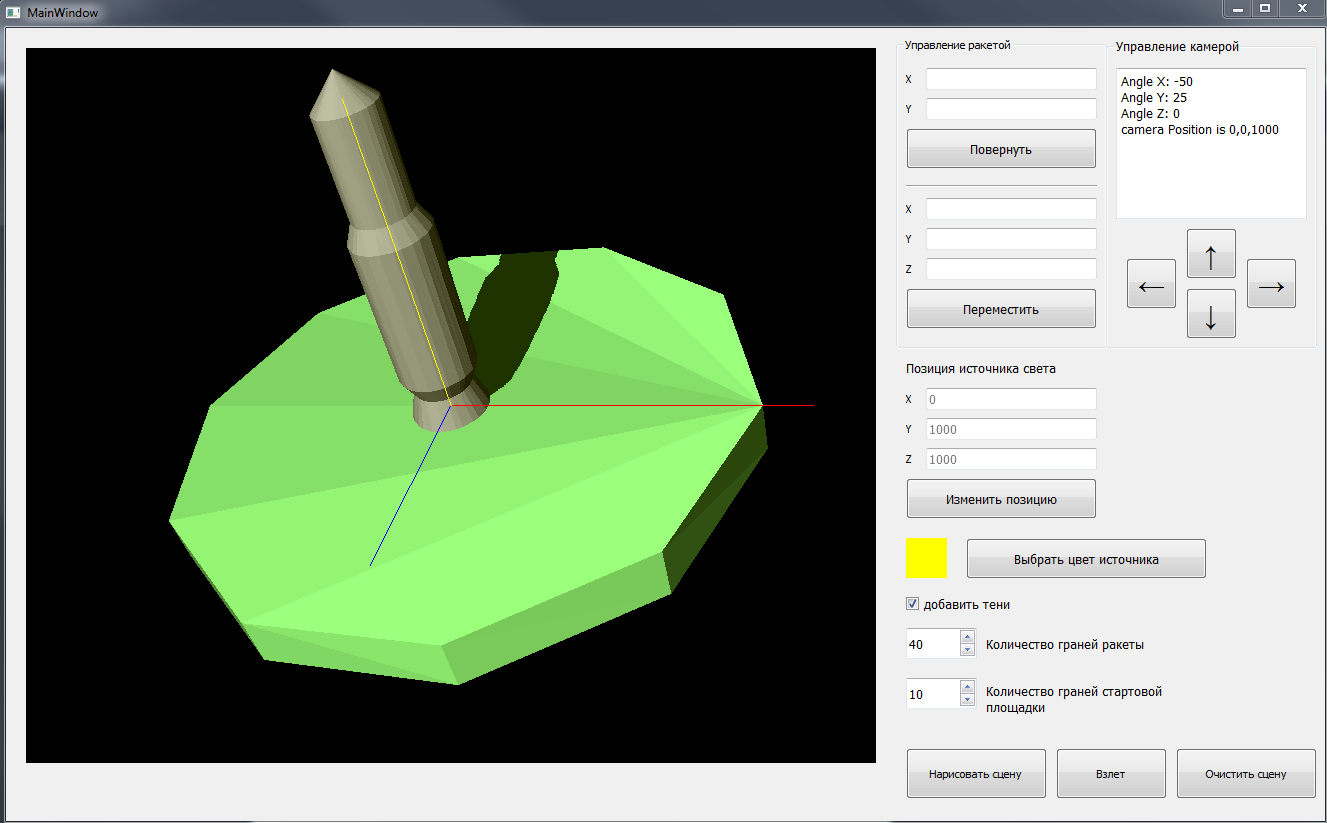


Рисунок 19 - Построенная сцена

### Описание основных этапов реализации

Ниже на листинге 1 описана реализация закраски с использованием z-буфера.

**Листинг1 – Реализация алгоритма z-буфера с помощью барицентрических координат**

void PaintWidget::**fillObject**(Point3D A, Point3D B, Point3D C)

{

double x1 = A.x();

double x2 = B.x();

double x3 = C.x();

double y1 = A.y();

double y2 = B.y();

double y3 = C.y();

double xmax, xmin, ymax, ymin, z;

xmax = getMaxFor3(x1, x2, x3);

xmin = getMinFor3(x1, x2, x3);

ymax = getMaxFor3(y1, y2, y3);

ymin = getMinFor3(y1, y2, y3);

Point3D P, ShadowPoint;

double square = (A.y() - C.y()) \* (B.x() - C.x()) + (B.y() - C.y()) \* (C.x() - A.x());

A.setZ(1. / A.z());

B.setZ(1. / B.z());

C.setZ(1. / C.z());

*for* (int y = int(ymin); y <= int(ymax); y++)

{

*for* (int x = int(xmin); x <= int(xmax); x++)

{

P.changeAll(x, y, 0);

ComputeBarycentric(A, B, C, P, square);

*if* (BarCoor.b1 >= 0 && BarCoor.b1 <= 1 && BarCoor.b2 >= 0 && BarCoor.b2 <= 1 && BarCoor.b3 >= 0 && BarCoor.b3 <=1)

{

z = 1 / (BarCoor.b1 \* A.z() + BarCoor.b2 \* B.z() + BarCoor.b3 \* C.z());

*if* (x + y \* WIDTH >= 0 && x + y \* WIDTH < WIDTH \* HEIGHT)

{

*if* (z > ZBuffer.zbuffer[x + y \* WIDTH])

{

ZBuffer.zbuffer[x + y \* WIDTH] = z;

*if* (fillShadow == *true*)

{

ShadowPoint.changeAll(x - X\_SIZE, y - Y\_SIZE, z - 10000);

ShadowPoint.setW((A.w() + B.w() + C.w()) / 3);

drawShadow(ShadowPoint);

}

painter->drawPoint(x, y);

}

}

}

}

}

На рисунке 20 приведено изображение ракеты, составленной из объектов, представленных на рисунке 2 (6 граней).

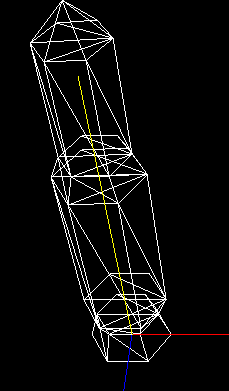


Рисунок 20 - Полигональная ракета из 6 граней

### Вывод

В данном разделе были выбраны средства реализации, рассмотрен интерфейс программы, инструкция по запуску, а так же листинг основных моментов реализации.

## Исследовательский раздел

В данном разделе будут приведены постановка эксперимента и сравнительный анализ алгоритмов на основе экспериментальных данных. При исследовании временных характеристик разработанной программы использовался компьютер на базе 4-х ядерного процессора Intel Core i5 частотой 1,6 - 4,1 MГц с 8 ГБ оперативной памяти типа DDR4 частотой 2300 МГц. Для замеров времени была использована библиотека chrono.

### Зависимость времени рендеринга от числа объектов и количества граней.

Для исследования зависимости времени рендеринга изображения от числа объектов на сцене, использовались объекты с различным числом граней, каждый объект имел освещенную и затененную части. Количество объектов менялось на сцене от 1 до 20 с шагом 10, были рассмотрены случаи для разных входных параметров. Количество граней у объектов менялось от 4 до 50. Результаты проведенного исследования представлены на рисунке 21.

Рисунок 21 - Сравнительный анализ времени рендеринга сцены от разных входных параметров

Как видно из графика, время рендеринга сцены линейно зависит от количества объектов.

### Зависимость времени взлета ракеты

Для исследования зависимости времени взлета ракеты от числа объектов на сцене с использованием теней, использовались объекты с одинаковым числом граней, каждый объект имел освещенную и затененную части. Количество объектов менялось на сцене от 1 до 5 с шагом 1. Результаты проведенного исследования представлены на рисунке 22.

Рисунок 22 - Сравнительный анализ времени рендеринга с использованием и без использования теней

Как видно из графика, время взлета ракеты зависит от использования теней. Каждое значение времени рендеринга сцены с использованием теней пропорционально увеличивается относительно значений времени рендеринга без использования теней.

### Вывод

В результате исследования временных затрат на рендеринг сцены с разными входными параметрами обнаружено, что от усложнения видимых объектов сцены, увеличения числа граней, добавления эффектов линейно увеличивается и время рендеринга изображения.

# Заключение

В ходе работы были проанализированы существующие алгоритмы удаления невидимых линий и поверхностей, построения теней и закраски, указаны их преимущества и недостатки.

Разработаны собственные и адаптированы существующие структуры данных и алгоритмы, необходимые для решения поставленной задачи.

Спроектировано и реализовано программное обеспечение, моделирующее взлет ракеты с земли.

# Литература

1. Роджерс Д. Математические основы машинной графики. / Роджерс Д., Адамс Дж. – М.: Мир, 1989. – 512с.
2. Fletcher Dunn, Ian Parberry. 3D Math Primer for Graphics and Game Development / Fletcher Dunn, Ian Parberry. – Second edition. – Taylor and Francis Group, LLC, 2011. – 845с.
3. Eric Lengyel. Mathematics for 3D Game Programming and Computer Graphics / Eric Lengyel. – Third edition. - Course Technology, a part of Cengage Learning, 2012. – 566с.
4. Шлее М. Qt 5.10. Профессиональное программирование на C++ / Шлее М. – СПб: БХВ-Петербург, 2018. – 1072с.
5. Огонь – GAME ENGINE BLACK BOOK DOOM[Электронный ресурс]. – Режим доступа: URL: http://fabiensanglard.net/doom\_fire\_psx/index.html – (Дата обращения: 10.11.2019).
6. А.Н.Романюк, М.В.Куринный. Журнал "КОМПЬЮТЕРЫ+ПРОГРАММЫ", № 8-9, 2000 г.