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:

«Трехмерная визуализация конструктора сцены»

Студент _____ИУ7-52Б_____ _____ _____Н.А.Гарасев _____

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

Руководитель курсового проекта _____ _____К.А.Кивва _____

(Подпись, дата) (И.О.Фамилия)

2020 г.

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)

УТВЕРЖДАЮ
Заведующий кафедрой ___ИУ7____
(Индекс)
_____ __И.В.Рудаков__
(И.О.Фамилия)
« _____ » _____ 2020 г.

З А Д А Н И Е
на выполнение курсового проекта

по дисциплине _____Компьютерная графика_____

Студент группы ____ИУ7-52_____

_____Гарасев Никита Алексеевич_____
(Фамилия, имя, отчество)

Тема курсового проекта ___Трехмерная визуализация конструктора сцены_____
_____
_____

Направленность КП (учебный, исследовательский, практический, производственный, др.)
_____учебный_____
Источник тематики (кафедра, предприятие, НИР) _____кафедра_____

График выполнения проекта: 25% к _4_ нед., 50% к _7_ нед., 75% к 11 нед., 100% к 14 нед.

Задание разработать программу для построения трехмерной сцены, состоящей из некоторых примитивов таких, как сфера, правильная четырехугольная пирамида, цилиндр и параллелограмм, обладающих изменяемыми атрибутами такие, как положение в пространстве, размеры, гладкость и отполированность. Необходимо предоставить возможность добавлять, удалять, а также изменять объекты сцены. Исследовать и выбрать алгоритм для построения трехмерного изображения и моделирования сцены.

Оформление курсового проекта:

Расчетно-пояснительная записка на 25-30 листах формата А4.
Перечень графического (иллюстративного) материала (чертежи, плакаты, слайды и т.п.)
На защиту проекта должна быть представлена презентация, состоящая из 15-20 слайдов. На слайдах должны быть отражены: постановка задачи, использованные методы и алгоритмы, расчетные соотношения, структура комплекса программ, диаграмма классов, интерфейс, характеристики разработанного ПО, результаты проведенных исследований.

Дата выдачи задания « ___ » _____ 20__ г.

Руководитель курсового проекта _____ ___К.А.Кивва._____
(Подпись, дата) (И.О.Фамилия)
Студент _____ ___Н.А.Гарасев,_____
(Подпись, дата) (И.О.Фамилия)

Примечание: Задание оформляется в двух экземплярах: один выдается студенту, второй хранится на кафедре.

Оглавление

Введение

Компьютерная графика – это совокупность методов и способов преобразования информации в графическую форму и из графической формы в ЭВМ[1].

В современном мире компьютерная графика является неотъемлемой частью человеческой жизни. Область применения компьютерной графики не ограничивается художественными эффектами. Она используются во всех отраслях науки, техники, медицины, в коммерческой деятельности, где используются построенные с помощью компьютера схемы, графики, диаграммы, предназначенные для наглядного отображения информации. Конструкторы, разрабатывая новые модели, используют трехмерные графические объекты, чтобы представить окончательный вид изделия. С помощью программных редакторов архитекторы проектируют здания, а затем просматривают их объемное изображение, что позволяет дать предварительные оценки будущего сооружения. Вследствие этого перед людьми, создающими трехмерные сцены, встает задача создания реалистичных изображений, которые будут учитывать оптические явления такие, как отражение, преломление и рассеивание света.

Существует множество алгоритмов компьютерной графики, которые решают данную задачу. Однако такие алгоритмы являются ресурсозатратными. Для получения более качественного изображения требуется большое количество времени и памяти[1].

Целью курсового проекта является разработка программы для создания трехмерных изображений сооружений, состоящих из сфер, цилиндров, параллелепипедов и правильных четырехугольных пирамид.

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 программного обеспечения,
- реализация выбранных алгоритмов и структур данных,
- разработка программного обеспечения, которое позволит отобразить трехмерную сцену,
- проведение исследования на основе разработанной программы.

Итогом работы является программное обеспечение, предоставляющее визуализацию и сцены. В программе предусмотрена возможность поворота камеры, приближения и отдаления объектов сцены, создания трехмерных изображений сооружений, состоящих из сфер, цилиндров, параллелепипедов и правильных четырехугольных пирамид.

1. Аналитическая часть

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

1.1. Постановка задачи

Необходимо предоставить возможность моделирования трехмерных изображений сооружений, состоящих из сфер, цилиндров, параллелепипедов и правильных четырехугольных пирамид. Данная программа является инструментом для проектирования конструкции и может быть использована архитекторами, конструкторами и дизайнерами.

Сцена состоит из следующих объектов:

- точечных источников света – представляют собой фиксированную точку в пространстве, называемой его позицией, из которой свет испускается равномерно во всех направлениях. Точечный источник полностью характеризуется его позицией и яркостью,
- сооружения, состоящие из фигур.

1.2. Формализация объектов синтезируемой сцены

Перечень фигур, необходимые для конструкции сцен в программе:

- сфера,
- параллелепипед,
- четырехугольная пирамида,
- цилиндр.

Список данных фигур достаточен для реализации цели курсового проекта.

Существует несколько видов геометрических моделей:

- каркасная модель;
- поверхностная модель;
- объемная модель.

Объекты сцены наилучшим образом описываются с помощью поверхностной модели, так как каркасные модели не обладают достаточной реалистичностью, а в объемной модели добавляется информация о том, где расположен материал, что в данной работе не нужно. Поэтому объекты в программе описываются с помощью поверхностной модели[1].

Для описания поверхности фигур выбран аналитический метод, потому что данный метод подходит для описания фигур вращения (сфера и цилиндр), четырехугольную пирамиду можно разбить на 6 треугольников, а параллелепипед двумя вершинами.

1.3. Выбор алгоритма удаления невидимых линий и поверхностей

Критериями выбора алгоритма удаления невидимых линий и поверхностей служат возможность работы с телами вращения и демонстрацией зеркального отражения.

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

1.3.1. Алгоритм Робертса

Алгоритм Робертса работает в объектном пространстве и может быть применен для построения изображения множества выпуклых многогранников. Алгоритм состоит из трех больших этапов: удаления нелицевых граней для каждого тела, удаления видимых ребер данного тела, экранируемых другими

телами сцены, удаление видимых ребер при взаимном “протыкании” тел, это продемонстрировано на рисунке 1.

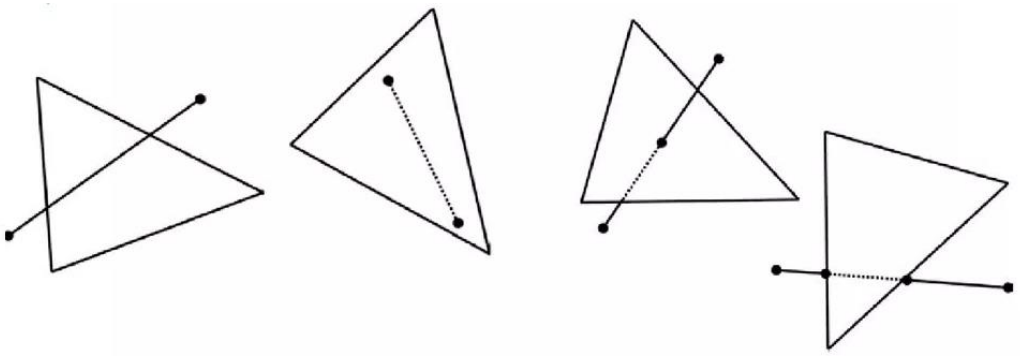

Рисунок 1. Этапы алгоритма Робертса

Серьезным недостатком является вычислительная трудоемкость алгоритма. В теории она растет как квадрат количества объектов. Поэтому при большом количестве фигур, находящихся в сцене, этот алгоритм будет показывать себя, как недостаточно быстрый. Можно использовать разные оптимизации для повышения эффективности, например сортировку по z. Минусом является и то, что алгоритм не позволяет визуализировать тени, зеркальные эффекты и преломление света. Использование растровых дисплеев, также снижает интерес к данному алгоритму, т.к. он работает в объектном пространстве[2].

Преимуществом данного алгоритма является точность вычислений. Она достигается за счет работы в объектном пространстве, в отличии от большинства других алгоритмов.

Алгоритм Робертса является очень точным в силу того, что вычисления производятся в объектном пространстве, однако его вычислительная сложность слишком велика. В силу того, что асимптотическая сложность алгоритма равна $\mathbf{O(n^2)}$, где n – количество объектов, при большом количестве квантов жидкости алгоритм будет слишком сложен для использования.

Плюсы:

- Очень точен

Минусы:

- Сильно теряет в производительности при большом количестве объектов
- Сложен в реализации
- Недостаточно детально моделирует отражение и преломление

1.3.2. Алгоритм Варнока

Алгоритм Варнока основывается на рекурсивном разбиении экрана. Алгоритм работает в пространстве изображения и анализирует область на экране дисплея (окно) на наличие в них видимых элементов. Если в окне нет изображения, то оно просто закрашивается фоном. Если же в окне имеется элемент, то проверяется, достаточно ли он прост для визуализации. Если объект сложный, то окно разбивается на более мелкие, для каждого из которых выполняется тест на отсутствие или простоту изображения. Рекурсивный процесс разбиения может продолжаться до тех пор, пока не будет достигнут предел разрешения экрана. На рисунке 2 представлено примерное разбиение сложной фигуры.

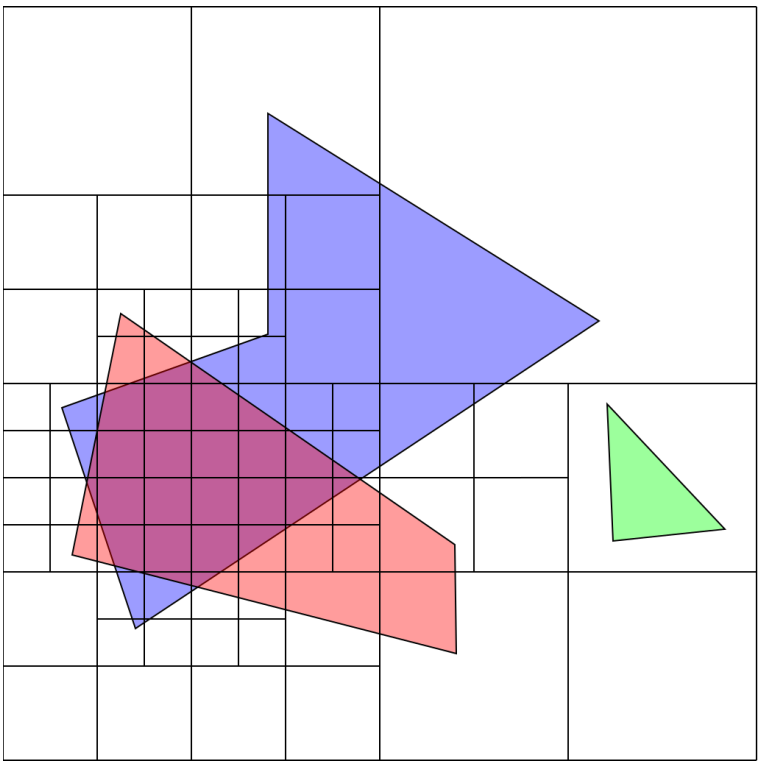

Рисунок 2. Разбиение сложной фигуры алгоритмом Варнока

Алгоритм Варнока не может быть применим для решения поставленной задачи, т.к. не моделирует оптические свойства объекта.

Плюсы:

- Весьма эффективен

Минусы:

- Не моделирует оптические эффекты

1.3.3. Алгоритм, использующий z-буфер

Данный алгоритм удаления невидимых поверхностей является одним из самых простых и широко используемых. Этот алгоритм работает в пространстве изображения. Его идея заключается в использовании двух буферов: буфера кадра и буфера глубины, также называемого Z-буфером. Буфер кадра используется для хранения интенсивности каждого пикселя в пространстве изображения. В буфере глубины запоминается значение координаты Z (глубины) каждого

видимого пикселя в пространстве изображения. В ходе работы алгоритма значение глубины каждого нового пикселя, заносимого 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 и производится корректировка Z-буфера: в него заносится глубина нового пикселя. Если же значение глубины нового пикселя меньше, чем хранящееся в буфере, то осуществляется переход к следующей точке. На рисунке 2 представлен пример работы z-буфера.

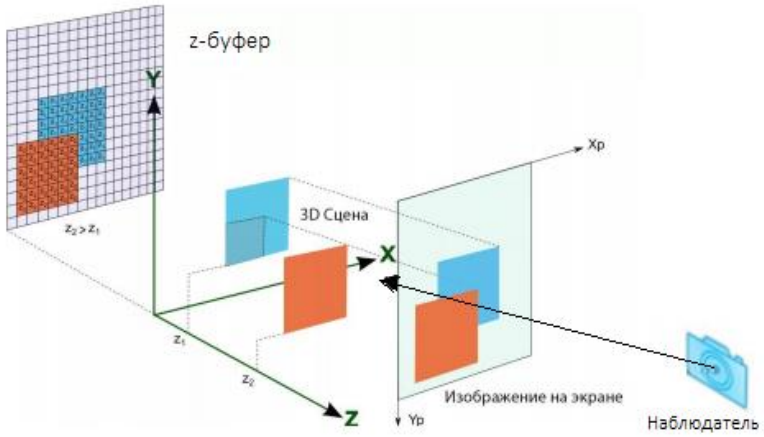

Рисунок 2. *Работа алгоритма z-буфера.*

Основными достоинствами данного алгоритма являются простота его реализации, корректная обработка случаев взаимных пересечений объектов, линейная зависимость трудоемкости от числа объектов на сцене, а также отсутствие необходимости предварительной сортировки объектов по глубине, то есть они могут обрабатываться в произвольном порядке.

К недостаткам данного алгоритма относят необходимость выделения памяти под два буфера, каждый из которых имеет размер, равный количеству пикселей на экране.

Благодаря модификации алгоритма z-буфера, можно учитывать тени и прозрачность. Однако алгоритм не может визуализировать эффект зеркального отражения, которое попадает на другие поверхности.

Плюсы:

- Прост в реализации
- Не требуется сортировка элементов сцены

Минусы:

- Требует хранение информации о каждом пикселе
- Не моделирует оптические эффекты

1.3.4. Алгоритм обратной трассировки лучей

Алгоритм является улучшенной модификацией алгоритма прямой трассировки. Из камеры испускаются лучи, проходящие через каждый пиксель вглубь сцены, затем идет поиск пересечений первичного луча с объектами сцены, как показано на рисунке 4, в случае обнаружения пересечения, рассчитывается интенсивность пикселя, в зависимости от положения источника света, при отсутствии пересечения, пиксель закрашивается цветом фона[4].

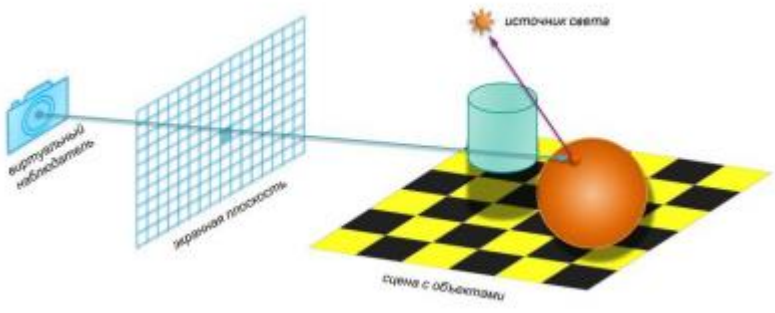

Рисунок 4. *Работа алгоритма обратной трассировки лучей.*

К достоинствам данного алгоритма можно отнести возможность получения изображения гладких объектов без аппроксимации их примитивами

(например, треугольниками). Трассировка лучей позволяет визуализировать тени, эффекты прозрачности, преломления, отражения. Вычислительная сложность метода линейно зависит от сложности сцены. Полученное изображение получается очень реалистичным[5].

Серьёзным недостатком алгоритма трассирования является производительность. Для получения изображения необходимо создавать большое количество лучей, проходящих через сцену и отражаемых от объекта. Это приводит к существенному снижению скорости работы программы. Однако для ускорения работы программы лучи можно трассировать параллельно, поскольку каждый луч, исходящий из камеры, независим от всех остальных[8].

Плюсы:

- Хорошо моделирует оптические эффекты
- Может быть ускорен параллельными вычислениями

Минусы:

- Очень ресурсозатратен
- Сложен в реализации

1.3.5. Вывод

Алгоритм **обратной трассировки лучей** выбран для удаления невидимых линий и поверхностей. Он позволяет получить изображение высокого качества, учитывая эффект отражения, а также предоставляет возможность работы с телами вращения[6].

1.4. Анализ алгоритмов построения теней

В силу выбора метода **трассировки лучей** вопрос построения теней оказывается решённым: пиксел затенён, если луч попадает на объект, и позже не попадает ни в объект, ни в источник света[3].

1.5. Анализ моделей освещения

Физические модели, которые не учитывают перенос света между поверхностями (не используют вторичное освещение), называются локальными. В противном случае модели называются глобальными или моделями глобального освещения. Для решения поставленной задачи имеет смысл использовать только локальную модель освещения, потому что глобальная модель освещения затрагивает ненужные физические явления, например преломление лучей. В данном подразделе будут рассмотрены локальные модели освещения Ламберта и Фонга.

1.5.1. Модель Ламберта

Модель Ламберта является одной из самых простых моделей освещения. Модель Ламберта моделирует идеальное диффузное освещение. Считается, что свет, падающий в точку, одинакового рассеивается по всем направлениям полупространства. Сила освещения зависит исключительно от угла α между вектором падения света L и вектором нормали N, как показано на рисунке 5. Максимальная сила света будет при перпендикулярном падении света на поверхность и будет убывать с увеличением угла α.

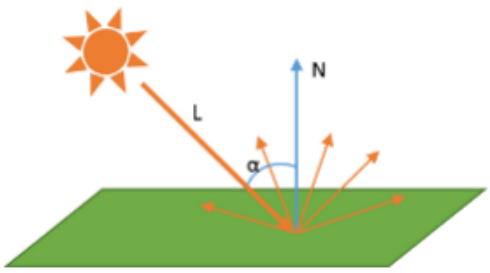

Рисунок 5. *Модель освещения Ламберта.*

Модель проста в реализации, но не позволяет передавать блики на телах сцены.

1.5.2. Модель Фонга

Основная идея модели Фонга заключается в предположении, что освещенность каждой точки тела разлагается на 3 компоненты:

1. фоновое освещение (ambient);

2. рассеянный свет (diffuse);

3. бликовая составляющая (specular).

Свойства источника определяют мощность излучения для каждой из этих компонент, а свойства материала поверхности определяют ее способность воспринимать каждый вид освещения.

Фоновое освещение присутствует в любом уголке сцены и никак не зависит от каких-либо источников света, поэтому для упрощения расчетов оно задается константой. Диффузное освещение рассчитывается аналогично модели Ламберта. Отраженная составляющая освещенности (блики) в точке зависит от того, насколько близки направления вектора, направленного на наблюдателя (вектор V на рисунке 6), и отраженного луча (вектор R на рисунке 6).

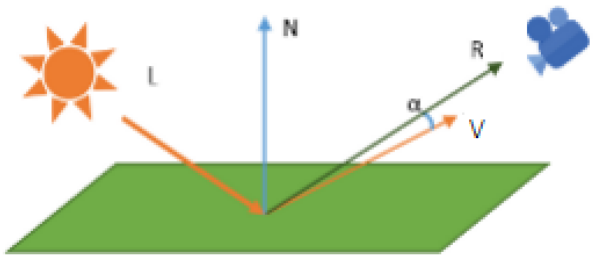

Рисунок 6. Получение бликов в модели освещения Фонга.

$$I = K_a I_a + K_d(\vec{N}, \vec{L}) + K_s(\vec{R}, \vec{V})^p$$, где

$\vec{N}$ – вектор нормали к поверхности в точке,

$\vec{L}$ – падающий луч (направление на источник света),

$\vec{R}$ – отраженный луч,

$\vec{V}$ – вектор, направленный к наблюдателю,

K_a – коэффициент фонового освещения,

K_d – коэффициент диффузного освещения,

K_s – коэффициент зеркального освещения.

p – степень, аппроксимирующая пространственное распределение зеркально отраженного света.

Все векторы являются единичными.

Модель Фонга улучшает визуальные качества сцены, по сравнению с моделью Ламберта, добавляя в нее блики (см. рис 7).

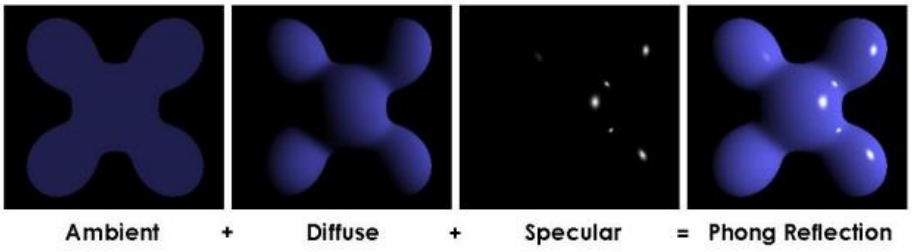

Рисунок 7. Модель освещения Фонга.

1.5.3. Вывод

В результате анализа алгоритмов, в соответствии с поставленной задачей была выбрана модель Фонга выбрана для расчета интенсивности в точке. Она позволяет учитывать матовые и блестящие поверхности.

2. Конструкторская часть

В данном разделе рассмотрены схемы алгоритмов для обратной трассировки лучей и расчета освещенности в соответствии с моделью Фонга, поиск пересечение луча с объектами сцены, диаграмма классов.

Программа должна обладать следующей функциональностью.

1) Визуализировать трехмерную сцену, состоящую из объектов, представленных в пункте 1.2, в режиме реального времени.
2) Предоставлять в интерфейсе возможность пользователю выполнять следующие действия:
    a. Добавлять и удалять объекты сцены.
    b. Изменять параметры объектов сцены.
    c. Изменять положение камеры и осуществлять её поворот.
    d. Добавлять точечные источники света.
    e. Изменять параметры источников света.

2.1. Схемы алгоритмов

На рисунке 8, 9 и 10 представлены схемы алгоритма трассировки лучей.

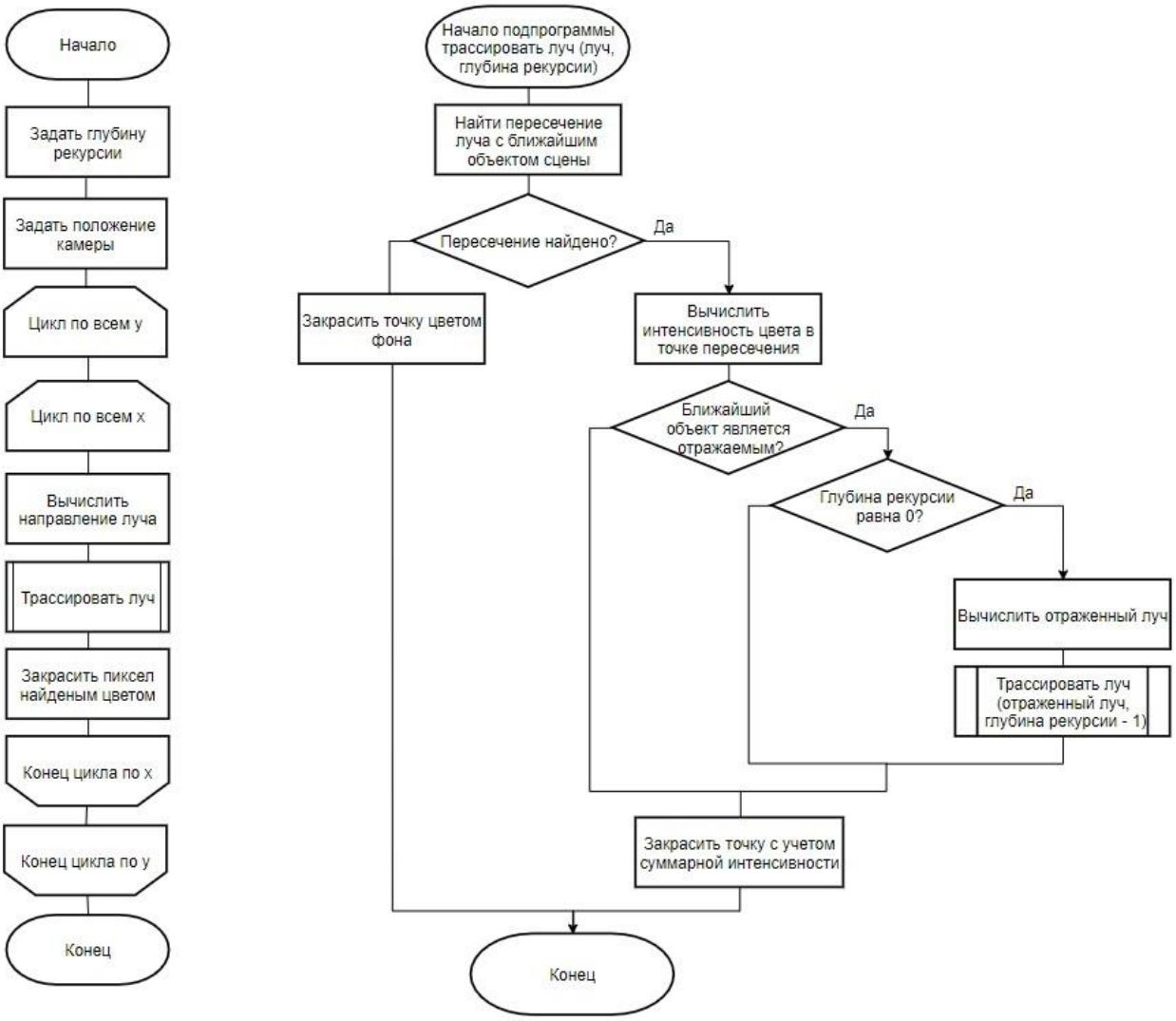

Рисунок 8. *Трассирование лучей.*

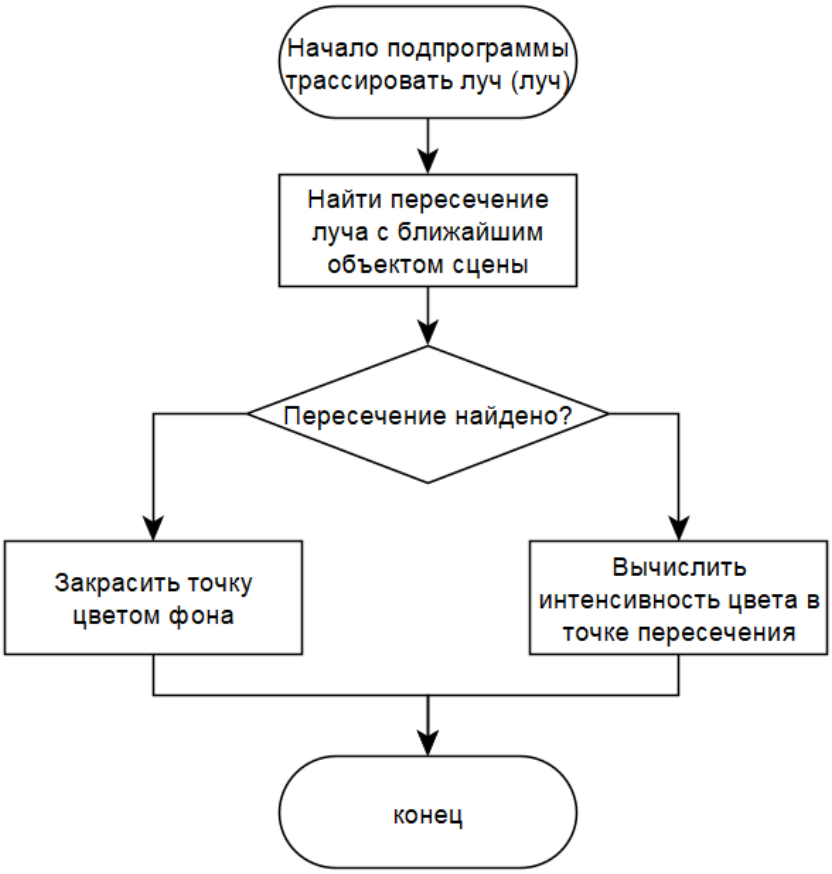

Рисунок 9. *Альтернативный алгоритм трассировки лучей для отображения в режиме построения сцены*

На рисунке 2 представлена схема алгоритма расчёта освещенности в соответствии с моделью Фонга.

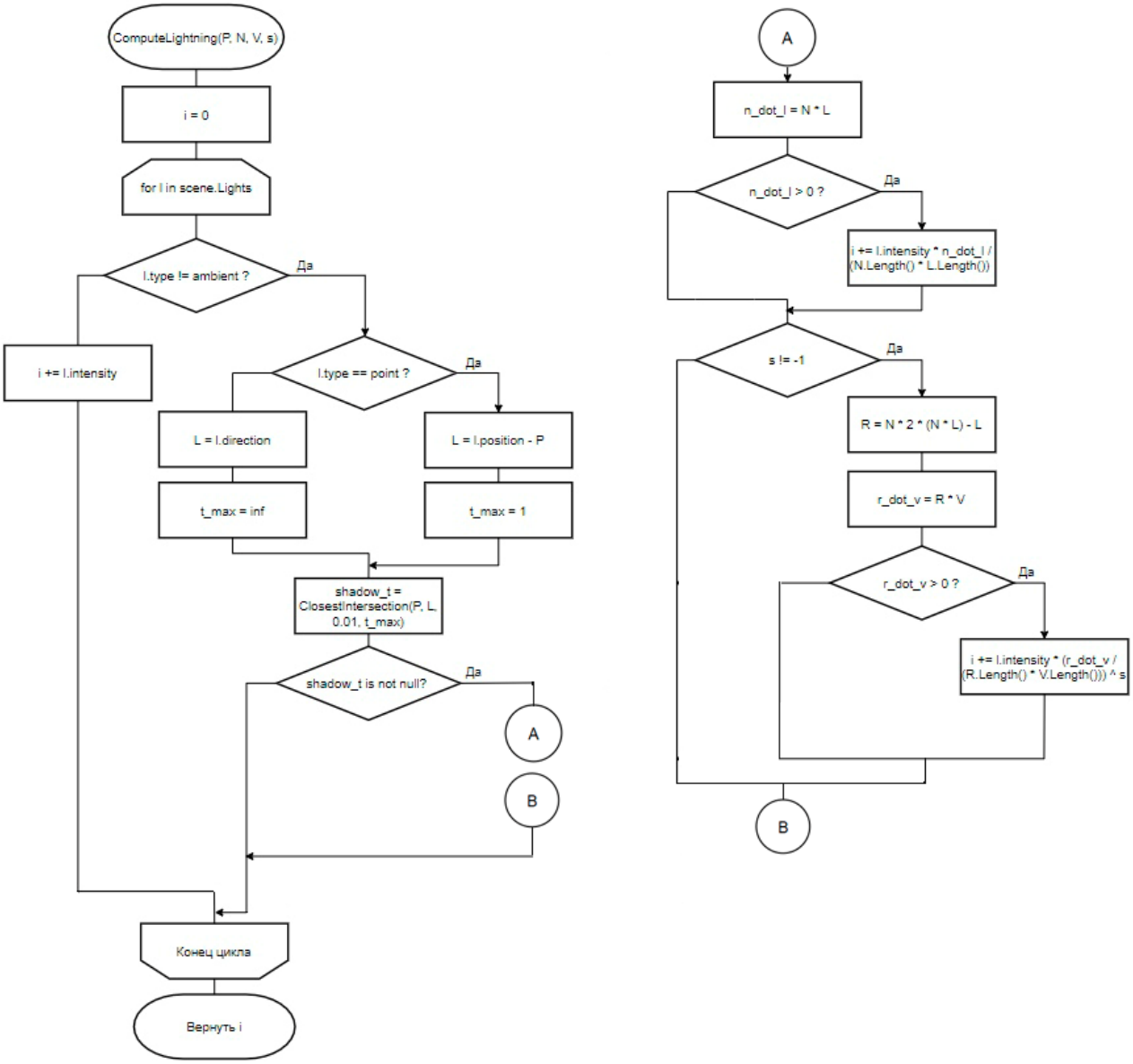

Рисунок 10. *Алгоритм закраски пикселей в соответствии с моделью освещения Фонга.*

2.2. Пересечение луча с объектами сцены

Наилучшим способом представления лучей является использование параметрического уравнения. Пусть O – начало луча, $\vec{D}$ – вектор, показывающий направление луча. Любую точку P луча можно представить как $P = O + t\vec{D}$, где t — произвольное действительное число.

Необходимо рассмотреть объекты сцены, с которыми лучи сталкиваются. В сцене присутствуют сферы, цилиндры, параллелепипеды и четырехугольные пирамиды, для представления которых используется такой геометрический примитив как треугольник, и плоскости. Для нахождения точки пересечения луча с произвольной поверхностью необходимо знать аналитические уравнения, определяющие оба эти объекта в трехмерном пространстве. Точка пересечения удовлетворяет всем уравнениям, так как принадлежит и лучу, и поверхности. Поэтому, сводя уравнения в систему и находя ее решения, можно получить координаты этой точки.

Для нахождения пересечения с объектами сцены используется скалярное произведение векторов, для которого принято следующее обозначение:

$dot(\vec{a}, \vec{b})$ – скалярное произведение векторов $\vec{a}$ и $\vec{b}$.

2.2.1. Пересечение луча со сферой

Пусть C – центр сферы, r – ее радиус, P – точка пересечения луча со сферой (см. рис 11).

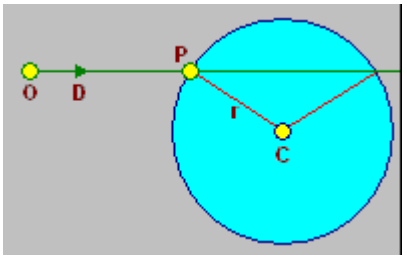

Рисунок 11. *Пересечение луча со сферой*

Тогда есть два уравнения, одно из которых описывает точки сферы, а другое – точки луча:

$$dot(P - C, P - C) = r^2 \quad (1.1)$$

$$P = O + t\vec{D} \quad (1.2)$$

Точка P, в которой луч падает на сферу, является одновременно и точкой луча, и точкой на поверхности сферы, поэтому она должна удовлетворять обоим уравнениям одновременно. Получается:

$$dot\left(\overrightarrow{CO} + t\vec{D}, \overrightarrow{CO} + t\vec{D}\right) = r^2 \quad (1.3)$$

$$t^2 dot\left(\vec{D}, \vec{D}\right) + 2 * t * dot\left(\overrightarrow{CO}, \vec{D}\right) + dot\left(\overrightarrow{CO}, \overrightarrow{CO}\right) - r^2 = 0 \quad (1.4)$$

Следовательно, для нахождения точки пересечения луча со сферой, необходимо решить квадратное уравнение, в котором отсутствие корней означает отсутствие пересечения, два корня – луч проходит через сферу, один корень – луч касается сферы. При получении двух корней выбирается меньший из них, он и будет расстоянием от начала луча до первого пересечения.

2.2.2. Пересечение луча с плоскостью

Пусть $\vec{V}$ – вектор нормали, P – точка пересечения луча с плоскостью, C – начальная точка (см. рис 12).

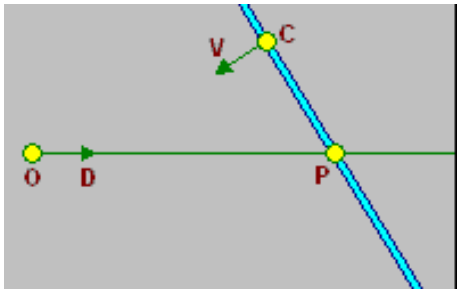

Рисунок 12. *Пересечение луча с плоскостью*

Уравнение плоскости:

$$Ax + By + Cz = 0 \quad (2.1)$$

Данное уравнение можно записать следующим образом:

$$dot(\vec{V}, P - C) = 0 \ (2.2)$$

Тогда пересечение уравнения, описывающего точки плоскости с уравнением, описывающим точки луча, находится следующим образом:

$$dot(\vec{V}, O + t\vec{D}) = 0 \quad (2.3)$$

$$dot(\vec{V}, O) + t * dot(\vec{V}, \vec{D}) - dot(\vec{V}, \vec{C}) = 0 \quad (2.4)$$

Отсюда t:

$$t = dot(\vec{V}, \vec{C}) - dot(\vec{V}, O)/dot(\vec{V}, \vec{D}) \ (2.5)$$

2.2.3. Пересечение луча с цилиндром

Пусть C – центр основания цилиндра, r – радиус его основания, $\vec{V}$ – вектор единичной длины, определяющий ось цилиндра, maxm – высота цилиндра, P – точка пересечения луча с цилиндром (см. рис 13).

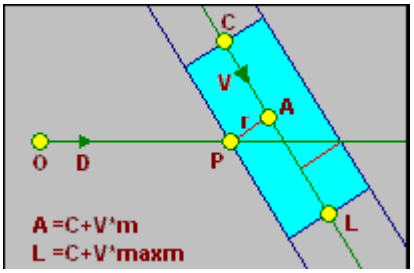

Рисунок 13. *Пересечение луча с цилиндром*

Исходя из условия:

$$A = C + m\vec{V} \quad (3.1)$$

$$dot\left(P - A, \vec{V}\right) = 0 \quad (3.2)$$

$$len(P - A) = r \quad (3.3)$$

где m – скаляр, определяющий ближайшую точку на оси к точке пересечения луча и боковой поверхности цилиндра. Вектор $P - A$ перпендикулярен V, что гарантирует самое близкое расстояние до оси. $P - A$ – это радиус цилиндра.

Тогда получается:

$$dot\left(P - C - \vec{V} * m, \vec{V}\right) = 0 \quad (3.4)$$

$$dot\left(P - C, \vec{V}\right) = m * dot\left(\vec{V}, \vec{V}\right) = m, \text{т.к } \vec{V} - \text{единичный вектор} \quad (3.5)$$

$$m = dot\left(\vec{D} * t + \overrightarrow{CO}, \vec{V}\right) \quad (3.6)$$

$$m = dot\left(\vec{D}, \vec{V}t\right) + dot\left(\overrightarrow{CO}, \vec{V}\right) \quad (3.7)$$

$$len\left(P - C - \vec{V} * m\right) = r \quad (3.8)$$

Получается:

$$t^2\left(dot\left(\vec{D},\vec{D}\right) - dot\left(\vec{D},\vec{V}\right)^2\right) + 2 * t * \left(dot\left(\vec{D},\overrightarrow{CO}\right) - dot\left(\vec{D},\vec{V}\right) * dot\left(\overrightarrow{CO},\vec{V}\right)\right)$$
$$+ \ dot\left(\overrightarrow{CO},\overrightarrow{CO}\right) - dot\left(\overrightarrow{CO},\vec{V}\right)^2 - r^2 = 0 \quad (3.9)$$

Следовательно, для нахождения точки пересечения луча с боковой поверхностью цилиндра, необходимо решить квадратное уравнение, в котором отсутствие корней означает отсутствие пересечения, два корня – луч проходит через цилиндр, один корень – луч касается цилиндра. При получении двух корней выбирается меньший из них, он и будет расстоянием от начала луча до первого пересечения.

Для того чтобы найти точки пересечения с основаниями цилиндра необходимо рассмотреть пересечения луча и пары плоскостей $\left(C, -\vec{V}\right)$ и $\left(C + \vec{V} * maxm, \vec{V}\right)$, ограниченные радиусом цилиндра.

2.2.4. Пересечение луча с параллелепипедом.

Параллелепипед задаётся координатами двух своих вершин: той, у которой все три координаты минимальны, и той, у которой все три координаты максимальны. Таким образом, получается шесть плоскостей, ограничивающих параллелепипед (см. рис 14).

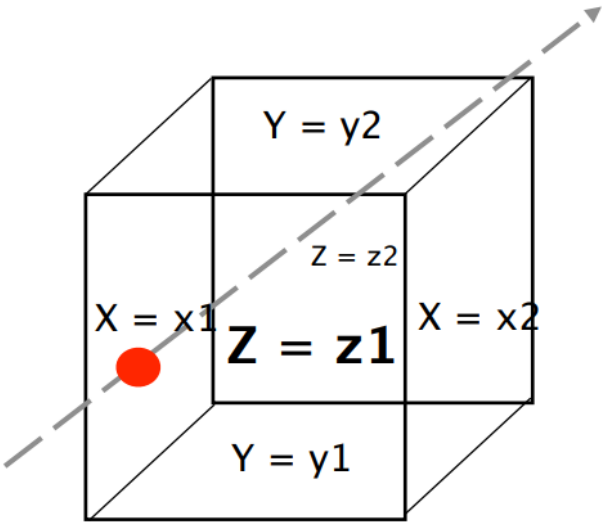

Рисунок 14. *Пересечение луча с параллелепипедом.*

Тогда пару плоскостей, параллельных плоскости yz: x = x1 и x = x2. Пусть $\vec{D}$ – вектор направления луча.

Если координата x вектора $\vec{D}$ равна 0, то заданный луч параллелен этим плоскостям и, если x0 < x1 или x0 > x1, то он не пересекает рассматриваемый прямоугольный параллелепипед. Если же D.x не равно 0, то вычисляем отношения:

Можно считать, что найденные величины связаны неравенством t1x < t2x.

Пусть tn = t1x, tf = t2x. Считая, что D.y не равно 0, и рассматривая вторую пару плоскостей, несущих грани заданного параллелепипеда, y = y1 и y = y2, находим величины:

$$t1y = (y1 - y0)/D.y \quad (5.3)$$

$$t2y = (y2 - y0)/D.y \quad (5.4)$$

Если t1y > tn, тогда пусть tn = t1y. Если t2y < tf, тогда пусть tf = t2y.

При tn > tf или при tf < 0 заданный луч проходит мимо прямоугольного параллелепипеда.

Считая, что D.z не равно 0, и рассматривая вторую пару плоскостей, несущих грани заданного параллелепипеда, z = z1 и z = z2, находим величины:

$$t1z = (z1 - z0)/D.z \quad (5.5)$$

$$t2z = (z2 - z0)/D.z \quad (5.6)$$

и повторяем предыдущие сравнения.

Если в итоге всех проведенных операций получается, что 0 < tn < tf или 0 < tf, то заданный луч пересечет исходный параллелепипед со сторонами, параллельными координатным осям.

Если начальная точка луча лежит вне параллелепипеда, то расстояние от начала луча до точки его входа в параллелепипед равно tn, а до точки выхода луча из параллелепипеда tf.

2.2.5. Пересечение луча с треугольником.

Для нахождения пересечения луча с треугольником используется алгоритм Моллера — Трумбора, который использует только вершины треугольника и не требует предварительное вычисление уравнения плоскости, содержащей треугольник. Пусть P — луч, $\vec{V}$ — его направление, v0, v1, v2 — вершины треугольника, z — точка пересечения, t —расстояние от точки вылета луча до точки пересечения луча и треугольника, u, v, t1 — барицентрические координаты, $cross(\vec{a}, \vec{b})$ — векторное произведение.

Барицентрические координаты представляют собой отношения площадей маленьких треугольников к большому треугольнику (см. рис 15).

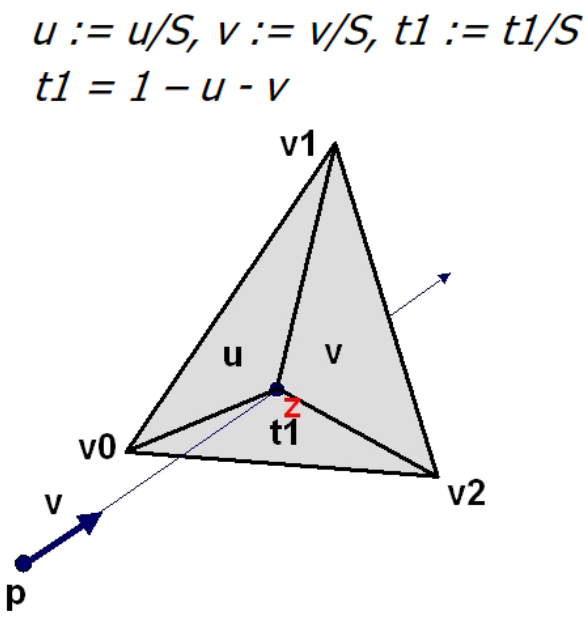

Рисунок 15. *Пересечение луча с треугольником.*

Имея 3 точки на плоскости, можно выразить любую другую точку через ее барицентричечкие координаты. Первое уравнение берется из определения барицентрических координат, выражая точку пересечения z. С другой стороны, эта же точка z лежит на прямой. Второе уравнение — параметрическое уравнение прямой. Прировняв правые части уравнений 1 и 2 получаем третье уравнение, которое, по сути, является системой 3-х уравнений (p, v, v1, v2, v3 - векторы) с 3-мя неизвестными (u, v, t).

$$z(u,v) = (1-u-v) * v1 + u * v2 + v * v0 \quad (6.1)$$

$$z(t) = p + t * d \quad (6.2)$$

$$p + t * d \; = (1-u-v) * v1 + u * v2 + v * v0 \quad (6.3)$$

После алгебраические преобразования, получается ответ в следующем виде:

$$E1 = v1 - v0 \quad (6.4)$$

$$E2 = v2 - v0 \quad (6.5)$$

$$t = p - v0 \quad (6.7)$$

$$P = cross(D, E2) \quad (6.8)$$

$$Q = cross(T, E1) \quad (6.9)$$

$$D = v \quad (7.0)$$

$$\begin{bmatrix} t \\ u \\ v \end{bmatrix} = \frac{1}{dot(P, E1)} * \begin{bmatrix} dot(Q, E2) \\ dot(P, T) \\ dot(Q, D) \end{bmatrix} \quad (7.1)$$

2.3. Нахождение отраженного луча

Для нахождения направления отраженного луча достаточно знать направление падающего луча $\vec{L}$ и нормаль к поверхности $\vec{N}$ в точке падения луча.

Можно разложить $\vec{L}$ на два вектора $\vec{L}p$ и $\vec{L}n$, таких что $\vec{L} = \vec{L}p + \vec{L}n$, где $\vec{L}n$ параллелен $\vec{N}$, а $\vec{L}p$ перпендикулярен, как изображено на рисунке 16.

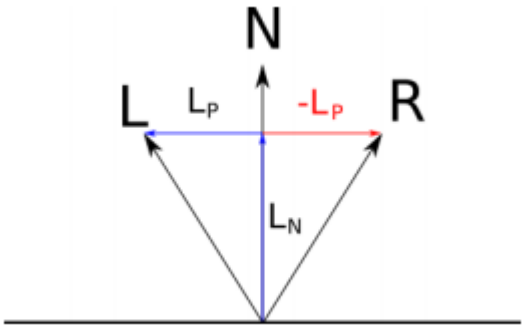

Рисунок 16. *Разложение вектора падающего луча.*

$\vec{L}n$ — проекция $\vec{L}$ на $\vec{N}$; по свойствам скалярного произведения и исходя из того, что $|\vec{N}| = 1$, длина этой проекции равна $(\vec{N}, \vec{L})$, поэтому

$$\vec{L}n = \vec{N}(\vec{N}, \vec{L}) \quad (8.1)$$

Отсюда:

$$\vec{L}p = \vec{L} - \vec{L}n = \vec{L} - \vec{N}(\vec{N}, \vec{L}) \quad (8.2)$$

Очевидно, что:

$$\vec{R} = \vec{L}n - \vec{L}p \quad (8.3)$$

Подставив полученные ранее выражения и упростив, можно получить формулу отраженного луча:

$$\vec{R} = 2\vec{N}(\vec{N}, \vec{L}) - \vec{L} \quad (8.4)$$

2.4. Диаграмма классов

На рисунке 17 изображена диаграмма классов

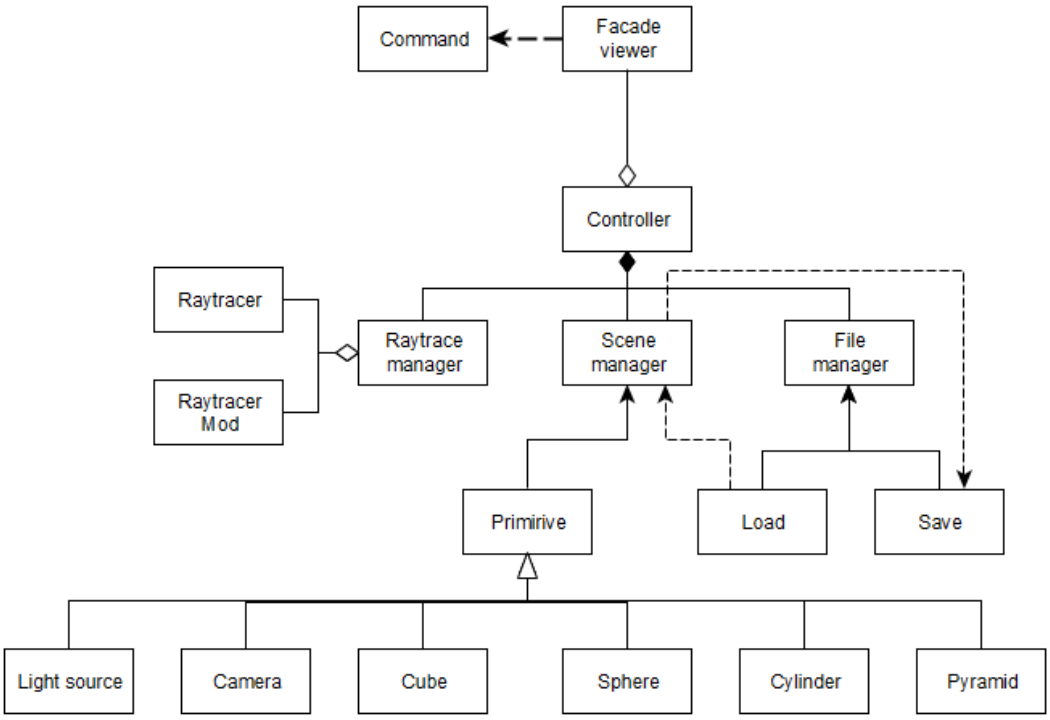

Рисунок 17. *Диаграмма классов*

Разработанная программа состоит из следующих классов.

Классы объектов:

- Primitive – базовый класс объектов;
- Sphere – класс сферы с возможностью задания радиуса и центра сферы.

- Cylinder – класс цилиндра с возможностью задания центра основания, радиуса, направление оси и высоты цилиндра.
- Cube – класс параллелепипеда, который задается координатами двух своих вершин: той, у которой все три координаты минимальны, и той, у которой все три координаты максимальны.
- Pyramid – класс четырехугольной пирамиды с возможностью задания вершины пирамиды и точек основания.

Вспомогательные классы сцены:

- Camera – класс камеры с возможностью перемещения по сцене;
- Light source – класс источника освещения с возможностью перемещения по сцене и изменения интенсивности.

Классы интерфейса:

- Facade – класс, который предоставляет интерфейс работы системы.
- Controller – класс для взаимодействия управляющих классов с классами интерфейса.

Класс визуализации сцены: Raytracer.

2.5. Структуры данных

Сцены созданные программно, сохраняются в специальном формате файла «.scene». Сами файлы представляют собой по структуре json файлы. Все объекты разбиваются на массивы, а затем каждый объект со всеми своими атрибутами записывается в json файл. При загрузке сцены json файл считывается и добавляет на сцену все соответствующие объекты.

3. Технологическая часть

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

3.1. Средства реализации

Для написания курсового проекта в качестве языка программирования был выбран C# т. к.:

- во время занятий по компьютерной графике был изучен данный язык программирования, что сократило время написания программы;
- 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.

В качестве среды разработки была выбрана «Visual Studio 2019» по следующим причинам:

- предоставляет умную проверку кода, быстрое выявление ошибок и оперативное исправление, одновременно с автоматическим рефакторингом кода;
- обеспечивает работу с Windows Forms – интерфейсом, который упрощает доступ к элементам интерфейса Microsoft Windows за счет создания обертки для существующего Win32 API в управляемом коде.

Для организации распараллеливания алгоритма трассировки лучей использовалось пространство имен «System.Threading», которое содержит в себе классы, поддерживающие многопоточное программирование.

3.2. Интерфейс программы

На рисунке 18 и 19 продемонстрирован интерфейс программы.

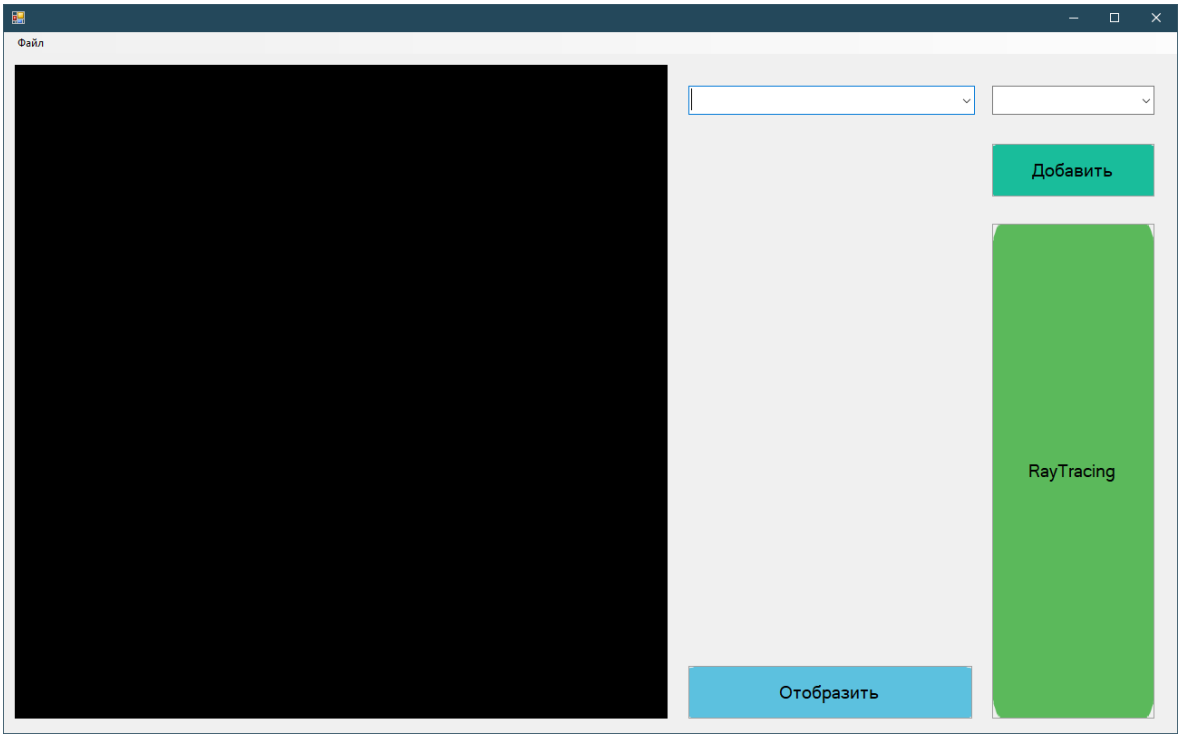

Рисунок 18. *Интерфейс.*

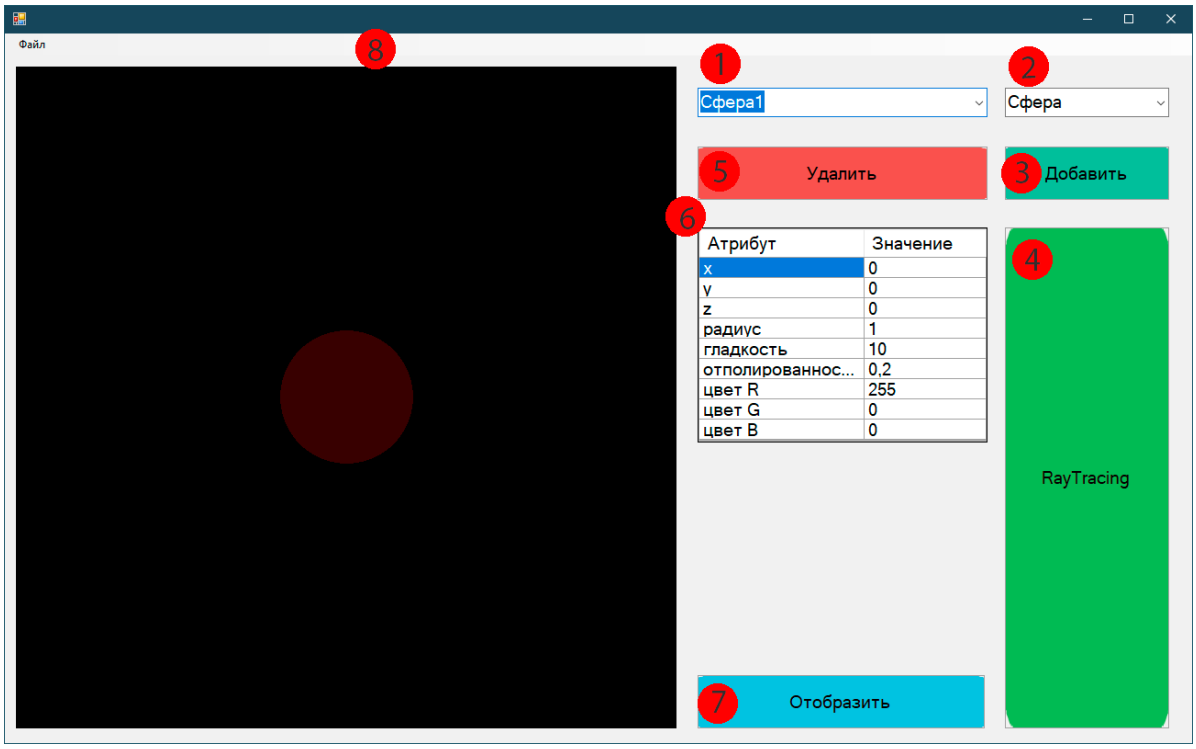

Рисунок 19. *Интерфейс*

На рисунке 19 отмечены основные элементы интерфейса.

1) Выпадающее меню, состоящее из всех элементов сцены.
2) Выпадающее меню, предназначенное для выбора элемента для добавления его на сцену.
3) Кнопка добавления выбранного элемента.
4) Кнопка получения реалистичного изображения с помощью рейтрейсинга.
5) Кнопка удаления выбранного объекта.
6) Таблица параметров выбранного объекта.
7) Кнопка отображения для режима построения сцены.
8) Полотно для отображения.

На рисунке 20 можно увидеть результат работы программы после загрузки сцены из файла.

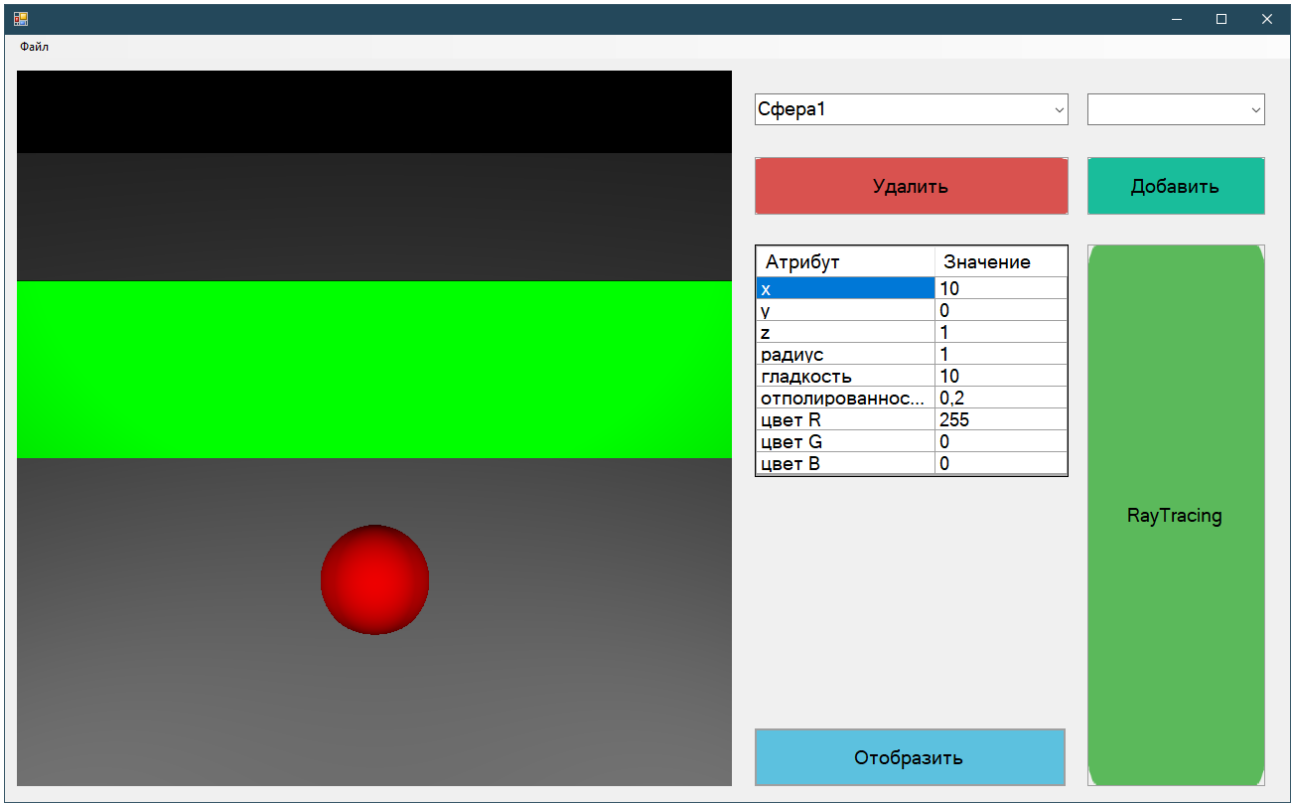

Рисунок 20. *Загрузка сцены*

Появилась таблица с атрибутами объекта, а также кнопка удаления.

На рисунке 21 продемонстрирован результат работы программы при нажатии на кнопку RayTracing.

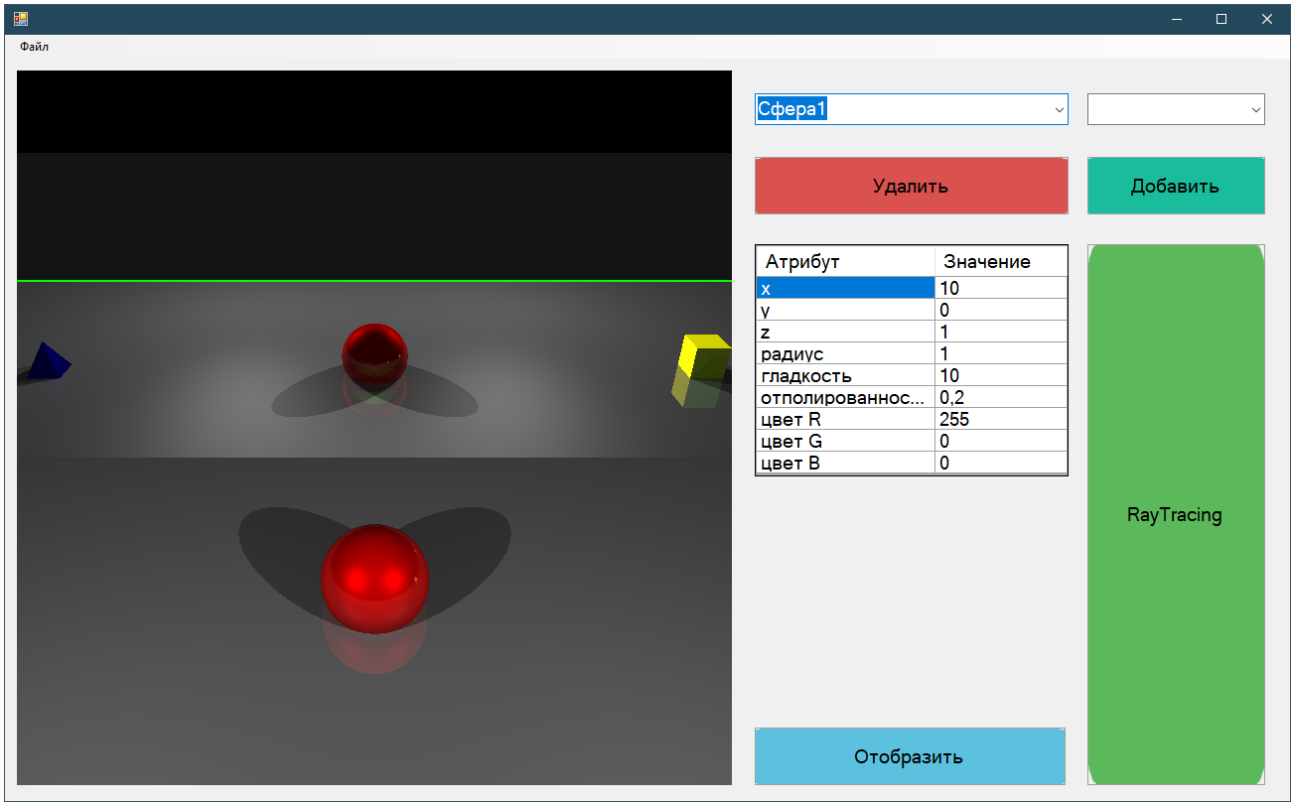

Рисунок 21. *RayTracing*

Задний параллелепипед оказался зеркалом, объект стал отбрасывать тени от источников света, а также стало возможно увидеть объекты находящиеся за пределом поля зрения камеры с помощью зеркала.

4. Исследовательская часть

В данном разделе 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 i3-8100. Для замеров времени использовалось пространство имен «System.Diagnostics».

4.1. Сравнение реализаций трассирования лучей

В ходе выполнения курсового проекта было реализовано два алгоритма трассирования лучей. Алгоритмы отличаются сложностью реализаций. Для дальнейшего различения алгоритмов им даны названия: простой и сложный. Простой отличается от сложного отсутствием теней и таких атрибутов, как зеркальностью объектов и их гладкость.

Необходимо сравнить время работы каждого алгоритма в зависимости от количества объектов на сцене.

Для сравнения данных алгоритмов составлены специальные сцены из разного количества объектов. На рисунке 22, 23 и 24 представлены примеры подобных сцен.

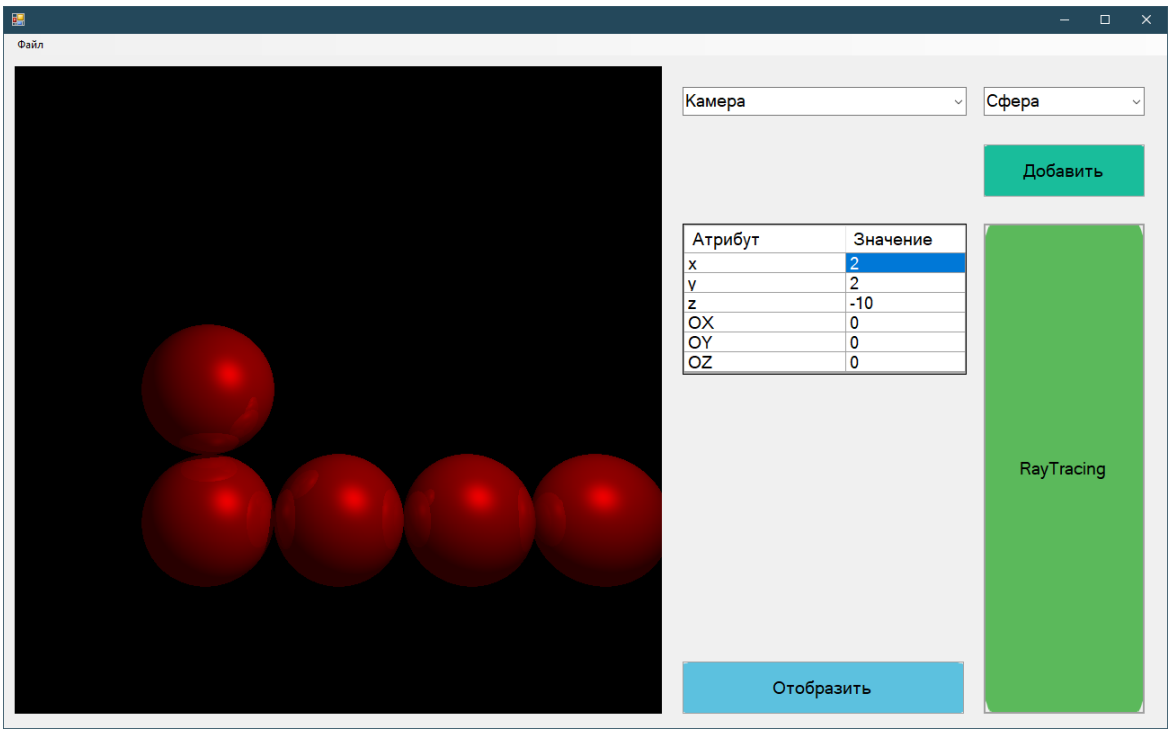

Рисунок 22. *Первая сцена для тестирования*

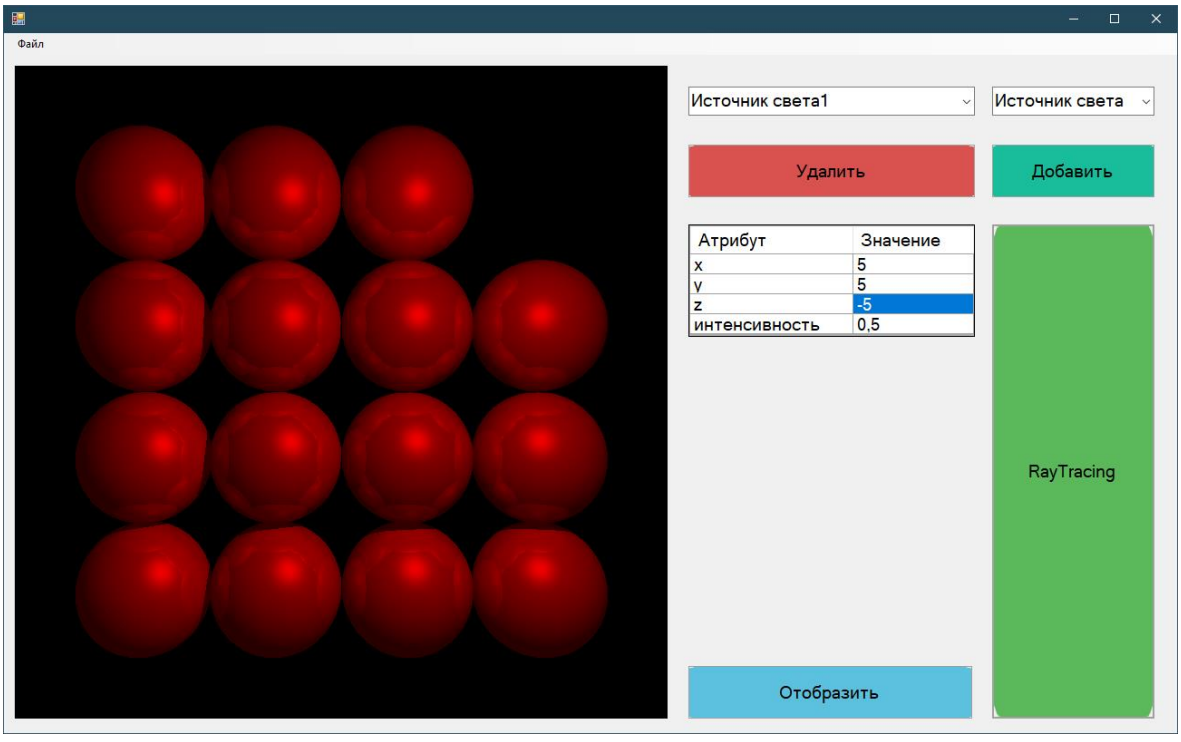

Рисунок 23. *Вторая сцена для тестирования*

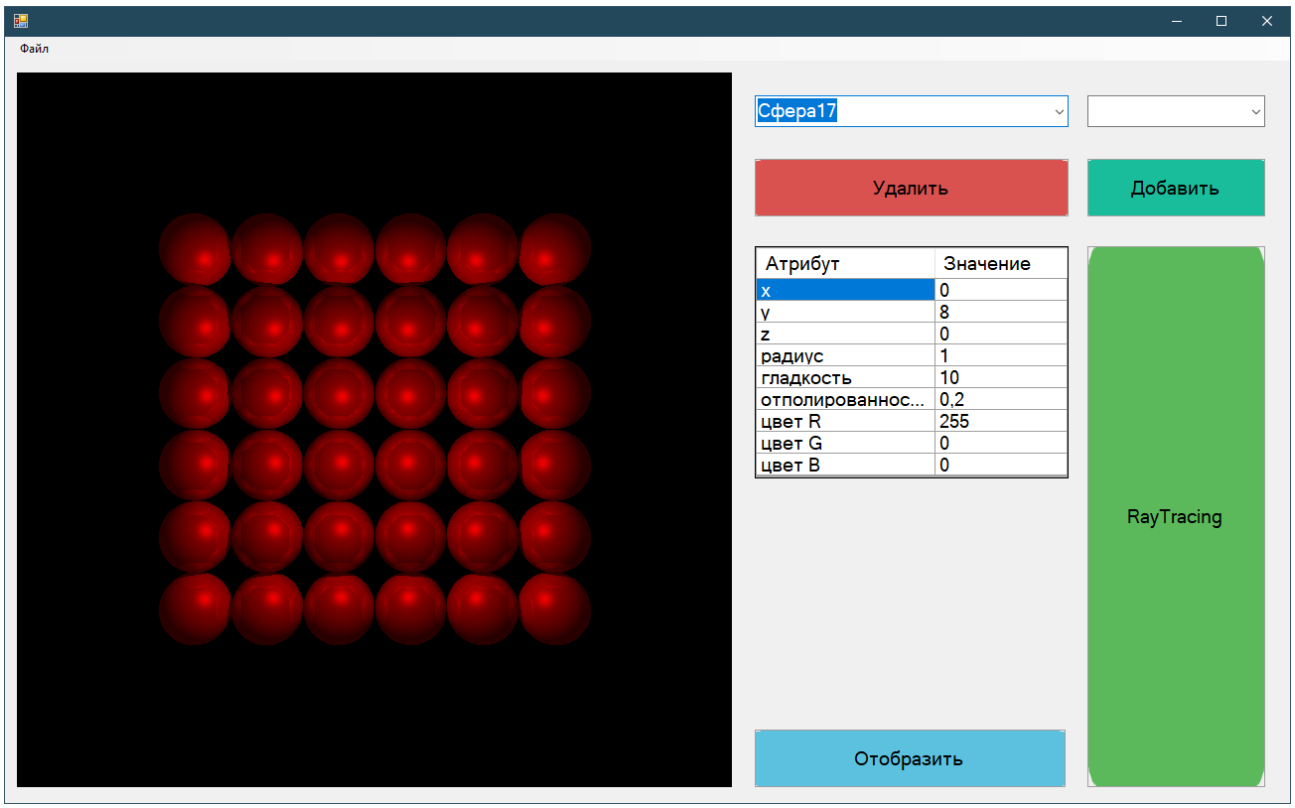

Рисунок 24. *Третья сцена для тестирования*

На рисунке 25 полученный результат представлен в виде графика зависимости времени работы алгоритмов от количества объектов на сцене. Зеленой линией представлен результат простого алгоритма, синий линией – сложного алгоритма. Результаты получены при распараллеливании вычислении на 4 потока. Время построения сцены считалось, как среднее время за 20 измерений.

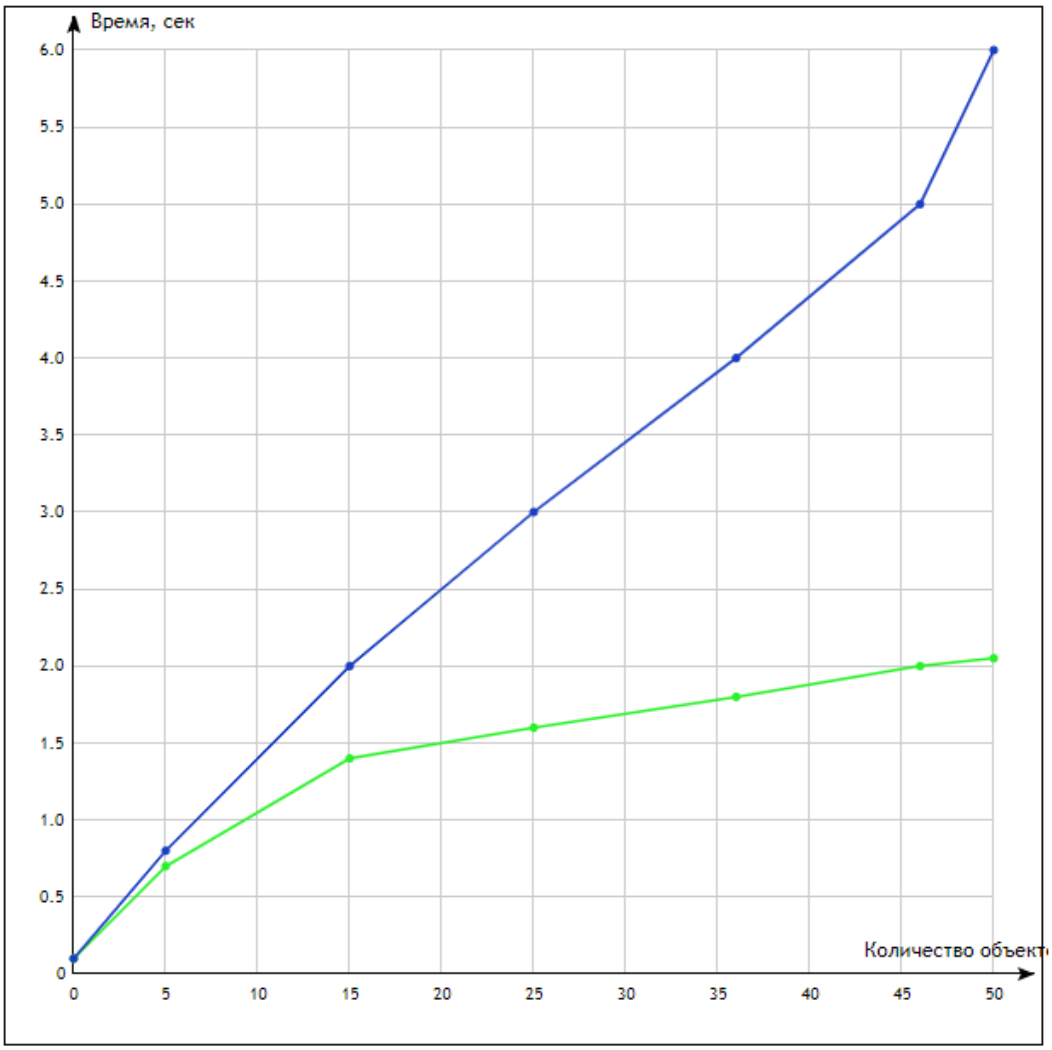

Рисунок 25. *График зависимости времени выполнения алгоритмов от количества объектов на сцене*

Вывод: наглядно видно, что время работы простой реализации гораздо меньше сложной, однако стоит учесть, что простой алгоритм не совсем справляется с задачей получения реалистичного изображения. Простой алгоритм крайне удобен для работы со сценой, для добавления объектов, изменения их свойств и т. д. Когда все объекты расположены и изменены, то можно запускать сложный алгоритм для получения реалистичного изображения.

4.2. Зависимость времени работы алгоритма трассирования лучей от количества потоков программы

В описанной формуле t - среднее время выполнения алгоритма, N – количество проводимых замеров, а T – суммарное время выполнения N замеров. Время построения сцены считалось, как среднее время за 20 измерений. Для измерения использовались сцены, состоящие из сфер (рис. 22 - 24). Время изменяется в секундах.

Формула усреднения значения эксперимента:

$$t = \frac{T}{N}$$

На рисунке 26 изображены графики зависимости времени рендеринга программы от количества объектов на сцене и числа потоков для параллельной версии алгоритма трассировки лучей.

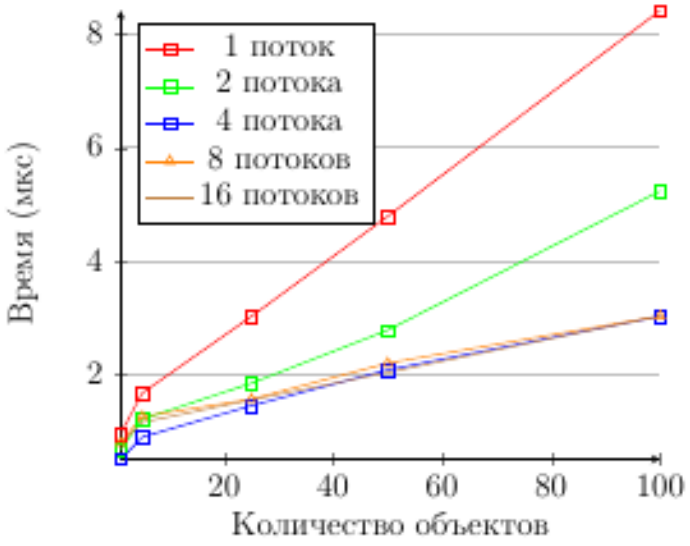

Рисунок 26. *Сравнительный анализ времени выполнения от количества потоков*

Как видно из графика, время рендеринга сцены линейно зависит от количества объектов. Максимальная скорость работы достигается при количестве потоков равным 4. Дальнейшее увеличение их числа не дает прироста в скорости. Это связано с тем, что число потоков, которые работают параллельно, равно числу логических процессоров.

Заключение

В ходе выполнения данной курсовой работы были проанализированы способы представления объектов, существующие алгоритмы удаления невидимых линий и поверхностей, модели освещения, указаны их преимущества и недостатки.

На основе результатов этого анализа, было спроектировано и реализовано программное обеспечение, позволяющее конструировать собственные сцены, состоящие из сфер, цилиндров, параллелепипедов и четырехугольных пирамид.

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

Были проведены эксперименты, показавшие, что алгоритм трассировки лучей можно распараллелить для ускорения его работы, а также, что максимальная скорость достигается при количестве потоков равное количеству логических процессоров экспериментальной машины. Также были проведены эксперименты для сравнения реализаций трассирования лучей, которые показали, что, поскольку для ускорения конструирования сцены в программе нет необходимости в использовании реалистичного изображения, есть возможность ускорить процесс за счет упрощения алгоритма трассировки лучей.

В качестве дальнейшего развития программы можно расширить ее функционал за счет добавления взаимодействия не только с примитивами, но и с более сложными объектами, созданных другими программами. Это даст возможность конструировать более сложные сцены и получать их реалистичные изображения.

Список литературы

1. Роджерс Д. Математические основы машинной графики. / Роджерс Д., Адамс Дж. – М.: Мир, 1989. – 512с.
2. Bruce Baumgart, Winged-Edge Polyhedron Representation for Computer Vision. National Computer Conference, May 1975
3. Е. А. Снижко. Компьютерная геометрия и графика [Текст], 2005. - 17 с.
4. Проблемы трассировки лучей – из будущего в реальное время. [Электронный ресурс]. – Режим доступа: https://nvworld.ru/articles/ray-tracing/3/ (дата обращения 28.06.19)
5. RayTracing – царь света и теней, Лев Дымченко [Электронный ресурс]. – Режим доступа: https://old.computerra.ru/206167/ (дата обращения 28.06.19)
6. Метод трассировки лучей против растеризации: новое поколение качества графики? [Электронный ресурс]. – Режим доступа: http://www.thg.ru/graphic/ray_tracing_rasterization/onepage.html(дата обращения 28.09.20)
7. Статьи по трассировке лучей + рейтрейсер на CUDA [Электронный ресурс]. – Режим доступа: http://www.ray-tracing.ru/(дата обращения 12.12.20)
8. "Ray Tracing Algorithm For Interactive Applications" [Электронный ресурс]. – Режим доступа: https://doi.org/10.1007%2FBF01911003(дата обращения 20.11.20)