



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 построения реалистичного изображения стержня, частично погруженного в тонкостенный прозрачный сосуд, наполненный жидкостью.

Студент ИУ7-54Б
(Группа)

Я. Р. Щховребова
(Подпись, дата)

(И. О. Фамилия)

Руководитель курсовой работы

Ю. И. Терентьев
(Подпись, дата)

(И. О. Фамилия)

2024 г.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ	5
1 Аналитическая часть	6
1.1 Объекты сцены	6
1.2 Способы представления объектов сцены	7
1.2.1 Каркасные модели	7
1.2.2 Полигональные модели	7
1.2.3 Твердотельные модели	8
1.3 Методы рендеринга сцены	8
1.3.1 Метод «бросания лучей»	8
1.3.2 Обратная трассировка лучей	9
1.3.3 Трассировка путей	9
1.4 Модели освещения	11
1.4.1 Простая модель освещений	11
1.4.2 Модель освещения Гуро	11
1.4.3 Модель освещения Фонга	11
2 Конструкторская часть	13
2.1 Требования к программному обеспечению	13
2.2 Разработка алгоритмов	14
2.2.1 Модель освещения Фонга	14
2.2.2 Обратная трассировка лучей	18
2.2.3 Общий алгоритм работы программы	26
2.3 Используемые типы и структуры данных	27
3 Технологическая часть	29
3.1 Средства реализации	29
3.2 Структура программы	29
3.3 Реализации алгоритмов	30
3.4 Графический интерфейс	36

4 Исследовательская часть	39
4.1 Технические характеристики	39
4.2 Время построения изображения	39
4.3 Реалистичность преломления лучей	42
ЗАКЛЮЧЕНИЕ	45
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ	47
ПРИЛОЖЕНИЕ А	48

ВВЕДЕНИЕ

В настоящее время компьютерная графика является одним из наиболее стремительно расширяющихся секторов в сфере информационных технологий. Ее область применения чрезвычайно широка, начиная от инженерии и науки, заканчивая искусством и медициной.

Одним из главных направлений компьютерной графики является создание реалистичных трехмерных изображений, которые могут быть использованы для наглядного представления разнообразной информации [1].

Цель курсовой работы – разработка программного обеспечения для построения реалистичного изображения стержня, частично погруженного в тонкостенный прозрачный сосуд, наполненный жидкостью.

Для достижения цели курсовой работы необходимо выполнить следующие задачи:

- 1) описать моделируемые объекты сцены;
- 2) проанализировать способы представления объектов и обосновать выбор наиболее подходящего;
- 3) проанализировать модели освещения и обосновать выбор наиболее подходящей;
- 4) проанализировать алгоритмы рендеринга сцены и обосновать выбор наиболее подходящего;
- 5) разработать выбранные алгоритмы;
- 6) реализовать программное обеспечение для достижения цели данной работы;
- 7) исследовать время построения изображения в зависимости от глубины рекурсии и от количества используемых дополнительных потоков; реалистичность преломления света для прозрачных объектов в зависимости от входные параметров для материалов объектов для разработанного программного обеспечения.

1 Аналитическая часть

1.1 Объекты сцены

Сцена состоит из нескольких объектов.

Источник света – объект сцены; точка в пространстве, от которой свет излучается равномерно во все направления.

Формула вектора в трехмерном пространстве [2, с. 5–7]:

$$\vec{v} = \begin{pmatrix} x \\ y \\ z \end{pmatrix}. \quad (1)$$

Плоскость – непрозрачный объект сцены; горизонтальная плоскость, которая служит основой для расположения объектов в сцене.

Уравнение плоскости в трехмерном пространстве можно записать в общем виде следующим образом [2, с. 45].

$$Ax + By + Cz + D = 0, \quad (2)$$

где:

- A, B, C – коэффициенты, определяющие ориентацию плоскости; соответствуют нормальному вектору \vec{n} к плоскости;
- D – свободный член, который смещает плоскость относительно начала координат;
- x, y, z – координаты произвольной точки на плоскости.

Сосуд и жидкость – прозрачные объекты сцены, представленные как прямые цилиндры, причем сосуд – тонкостенный.

Уравнение прямого цилиндра [2, с. 45] с радиусом r , высотой h и коэффициентом масштабирования m :

$$x^2 + m^2y^2 = r^2, 0 \leq z \leq h. \quad (3)$$

Стержень – непрозрачный объект сцены, представленный как прямой параллелепипед.

Параллелепипед можно задать шестью уравнениями плоскости [3, с. 143]:

$$x = 0, \quad x = r_x, \quad (4)$$

$$y = 0, \quad y = r_y, \quad (5)$$

$$z = 0, \quad z = r_z, \quad (6)$$

где r_x , r_y и r_z представляют его размеры.

Стержень и жидкость располагаются внутри сосуда, причем стержень частично погружен в жидкость, располагаясь внутри его объема от верхней поверхности до определенной глубины.

1.2 Способы представления объектов сцены

Существует несколько способов представления объектов, которые различаются по степени сложности, возможностям редактирования и применяемости в разных сферах трехмерной графики. Рассмотрим три основных типа представления объектов, описанные в [4, с. 102]

1.2.1 Каркасные модели

Каркасная модель – способ представления объекта, при котором отображаются только его ребра и вершины, без внутренних поверхностей. Является основой для дальнейшего моделирования, текстурирования и визуализации. Каркасная модель не включает в себя полные геометрические данные (поверхности), а только контуры, которые дают общее представление о форме объекта [4, С. 102, 107].

1.2.2 Полигональные модели

Полигональная модель – тип модели, который состоит из набора точек (вершин), ребер и полигонов, которые объединяются в единую структуру. Набор поверхностей можно задать аналитически (уравнением или системой

уравнений) либо в виде полигональной сетки, где каждый полигон представляет собой многоугольник, который обычно состоит из треугольников или четырехугольников [4, с. 108].

1.2.3 Твердотельные модели

Твердотельная модель – это метод представление объекта, который включает полное описание его геометрической формы и внутренней структуры. В отличие от поверхностных моделей, которые описывают только внешнюю оболочку объекта, твердотельная модель полностью описывает объём и физические характеристики объекта [5].

Выбор представления объектов сцены

Так как каркасные и полигональные модели не могут учитывать физические свойства объектов, выбрана твердотельная модель, как способ представления объектов сцены, который обеспечивает полное описание объекта, что позволяет учитывать физические свойства, такие как отражение и преломление света, которые необходимы для построения реалистичного изображения.

1.3 Методы рендеринга сцены

Рендеринг – это процесс преобразования трехмерной модели или сцены в изображение, которое может быть отображено на экране [6, С. 11–12].

При выборе метода рендеринга моделей, необходимо учитывать поддержку эффектов освещения, отражений и преломлений выбранным методом, следовательно, будут рассмотрены методы трассировки лучей.

Трассировка лучей – это метод рендеринга, который моделирует путь света от камеры или источников света в сцене для расчета цветов пикселей. Рассмотрим следующие методы трассировки лучей, описанные в [6, С. 443–446].

1.3.1 Метод «бросания лучей»

Метод «бросания лучей» – это базовый метод трассировки лучей, при котором для каждого пикселя на экране из камеры проходит луч в сцену, и

определяется, с каким объектом он пересекается. Луч проверяет, есть ли на его пути объекты, и возвращает информацию о ближайшем объекте. Этот метод используется в основном для вычисления видимости и освещенности.

Данный метод обладает следующими характеристиками.

- Трудоемкость: $O(N)$, где N – количество пикселей.
- Требуется память для хранения информации о пересечении луча с объектами (объекты сцены, буфер изображения).
- Ограничение основным освещением, то есть без учета отражений, преломлений лучей и сложных эффектов.

1.3.2 Обратная трассировка лучей

Обратная трассировка лучей – это улучшенная версия метода «бросания лучей», которая учитывает не только пересечение луча с объектами, но и дополнительные физические явления, такие как отражения, преломления и тени. Каждый луч может быть использован для вычисления того, как свет распространяется в сцене, создавая более реалистичное изображение.

Данный метод обладает следующими характеристиками.

- Трудоемкость: $O(N \cdot R)$, где N – количество пикселей, R – количество отражений и преломлений лучей.
- Требуется память для хранения информации о пересечении луча с объектами (объекты сцены, буфер изображения), а также дополнительная память для хранения информации о преломлении и отражении каждого луча.
- Учитывает основное освещение, отражения и преломления лучей.

1.3.3 Трассировка путей

Трассировка путей – это улучшение трассировки лучей, где для каждого луча строится случайный путь через сцену. Лучи могут многократно отражаться и преломляться, что позволяет точно моделировать более сложные эффекты, такие как мягкие тени и рассеяние света.

Данный метод обладает следующими характеристиками.

- Трудоемкость: $O(N \cdot M)$, где N – количество пикселей, M – количество дополнительных случайных путей на пиксель.
- Требуется память для хранения информации о пересечении луча с объектами (объекты сцены, буфер изображения), а также дополнительная память хранения путей, случайных точек пересечений и анализа их взаимодействий.
- Поддерживается глобальное освещение, мягкие тени, преломления, отражения и рассеяние.

Выбор метода рендеринга моделей

В таблице 1 приведено сравнение методов трассировки лучей по вычислительной нагрузке, реалистичности и сложности реализации.

В отличие от метода «бросания лучей», который только определяет пересечения лучей с объектами, обратная трассировка лучей способна точно моделировать такие эффекты, как отражения, преломления и тени.

Трассировка путей предоставляет максимальную реалистичность, но требует значительно большего времени на обработку из-за необходимости генерировать множество путей для каждого луча. Этот метод может быть слишком ресурсоемким для большинства приложений, где требуется высокая скорость рендеринга, но при этом важна качественная симуляция отражений и преломлений.

Таблица 1 – Сравнение методов трассировки лучей

Метод	Вычислительная нагрузка	Реалистичность изображения	Сложность реализации
«Бросание лучей»	Низкая	Низкая	Низкая
Обратная трассировка лучей	Средняя	Средняя	Средняя
Трассировка путей	Высокая	Высокая	Высокая

На основе анализа методов рендеринга моделей выбран метод обратной трассировки лучей для построения реалистичного изображения с учетом преломлений и отражений, так как предлагает более сбалансированное решение между качеством изображения и вычислительными затратами.

1.4 Модели освещения

Модели освещения – это математические подходы, используемые в компьютерной графике для расчета взаимодействия света с поверхностями объектов. Они играют ключевую роль в создании реалистичных изображений, моделируя такие эффекты, как отражение, рассеивание и тени. В зависимости от сложности, каждая модель имеет свои характеристики, применимость и вычислительные затраты. Рассмотрим основные модели освещения, описанные в [7].

1.4.1 Простая модель освещений

Простая модель освещения основывается на расчете диффузного освещения, где свет поступает равномерно на поверхность. Это базовая модель, использующая информацию о нормалях поверхностей и источниках света, но не учитывающая эффекты отражений, преломлений или блеска. Вычисления для данной модели быстрые и несложные, но она имеет низкую реалистичность, так как не моделирует более сложные эффекты освещения [7, с. 35].

1.4.2 Модель освещения Гуро

Модель Гуро улучшает простую модель освещения за счет учета нормалей в вершинах объектов. Она использует интерполяцию значений освещенности между вершинами полигона для получения более плавного перехода освещенности на поверхности. Модель Гуро хорошо подходит для рендеринга объектов с матовыми или полугладкими поверхностями, но она ограничена в точности и не может точно моделировать отражения и блики [7, с. 43].

1.4.3 Модель освещения Фонга

Модель Фонга является более сложной, но и наиболее точной в плане реалистичности. Она учитывает три компонента освещения: диффузный, зеркальный и окружающий свет, что позволяет получать точные блики и эффекты отражения. Модель Фонга позволяет моделировать как гладкие, так и зеркальные поверхности. Вычисления для каждого пикселя значительно сложнее, так как включают расчет угла между наблюдателем и источниками

ками света, а также интенсивности отражений. Это требует более высоких вычислительных мощностей, но позволяет получать очень реалистичные результаты [7, с. 33].

Выбор модели освещения

В таблице 2 приведено сравнение моделей освещения по вычислительной нагрузке, реалистичности и сложности реализации.

Таблица 2 – Сравнение моделей освещения

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

На основе анализа моделей освещения выбрана модель освещения Фонга. Простая модель недостаточна из-за низкой реалистичности, а модель Гуро не обеспечивает высокого уровня детализации, а также не подходит к твердотельной модели, геометрическая форма которой задается аналитически. Модель Фонга, благодаря расчету отраженного света с учетом бликов, обеспечивает более реалистичное изображение при умеренной вычислительной нагрузке.

Вывод

В данном разделе были рассмотрены подходы к созданию реалистичных изображений. Описаны характеристики объектов сцены и методы их представления, проанализированы различные алгоритмы рендеринга и модели освещения. На основании анализа сделан выбор в пользу твердотельной модели представления объектов, модели освещения Фонга и алгоритма прямой трассировки лучей. Эти методы наиболее подходящие для разработки алгоритма, способного реалистично отображать сцены с учетом отражений, преломлений и сложного взаимодействия света.

2 Конструкторская часть

В данной разделе описаны требования к программному обеспечению, разработаны схемы алгоритмов, которые будут реализованы для программного обеспечения, выполняющего реалистичное построение изображения, а также будет описаны используемые типы и структуры данных.

2.1 Требования к программному обеспечению

Разрабатываемое программное обеспечение должно предоставлять следующие функциональные возможности.

1. Построение сцены:

- отображение тонкостенного прозрачного цилиндрического сосуда, наполненного жидкостью;
- отображение непрозрачного прямоугольного стержня, частично погруженного в жидкость;
- учет одного источника света для освещения сцены.

2. Характеристики материалов:

- задание характеристик материала сосуда (прозрачность, отражаемость, коэффициент преломления, цвет);
- задание характеристик материала жидкости (прозрачность, отражаемость, коэффициент преломления, цвет);
- задание характеристик материала стержня (отражаемость, цвет).

3. Пользовательский интерфейс: предоставление удобного графического интерфейса для изменения параметров сцены (характеристик материалов, положения источника света и камеры в сцене).

4. Рендеринг изображения:

- построение изображения с реалистичными эффектами освещения;
- отображение результатов с учетом прозрачности, отражения, преломления и теней;

- экспорт финального изображения в стандартные графические форматы (например, PNG, JPEG).

2.2 Разработка алгоритмов

Далее будут описаны схемы алгоритмов, которые будут реализованы при разработке программного обеспечения.

2.2.1 Модель освещения Фонга

Модель освещения Фонга, как показано на рисунке 1, включает три компонента [8]:

- 1) **диффузное освещение** – пропорциональное косинусу угла между нормалью поверхности и направлением света;
- 2) **зеркальное освещение** – имитирующее яркие блики, зависящие от угла между отраженным светом и направлением камеры;
- 3) **фоновое освещение** – добавляет общее освещение.

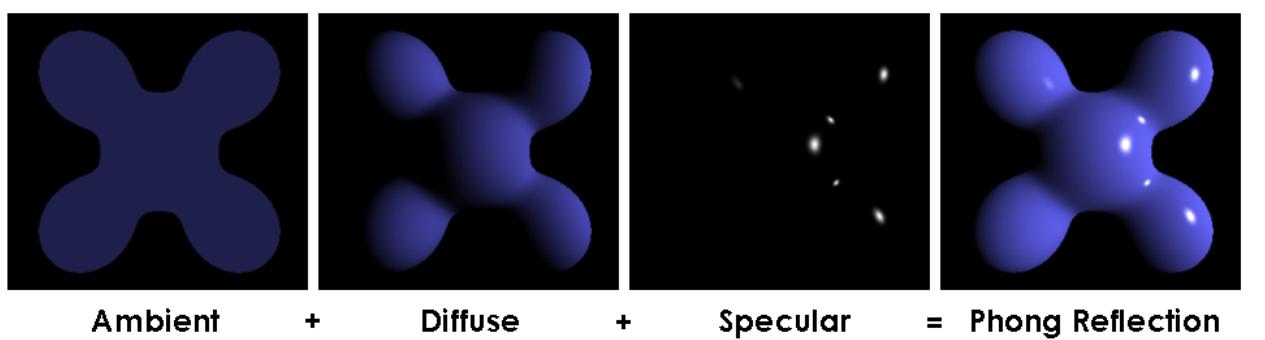


Рисунок 1 – Визуальная иллюстрация уравнения Фонга

Тогда **полное освещение** для пикселя является суммой этих трех компонентов:

$$I = I_{ambient} + I_{diffuse} + I_{specular}, \quad (7)$$

где I – полная интенсивность освещения для пикселя; $I_{ambient}$, $I_{diffuse}$, $I_{specular}$ – компоненты освещения (фоновое, диффузное и зеркальное освещения, соответственно).

На рисунке 2 изображена схема алгоритма вычисления полного освещения для пикселя.

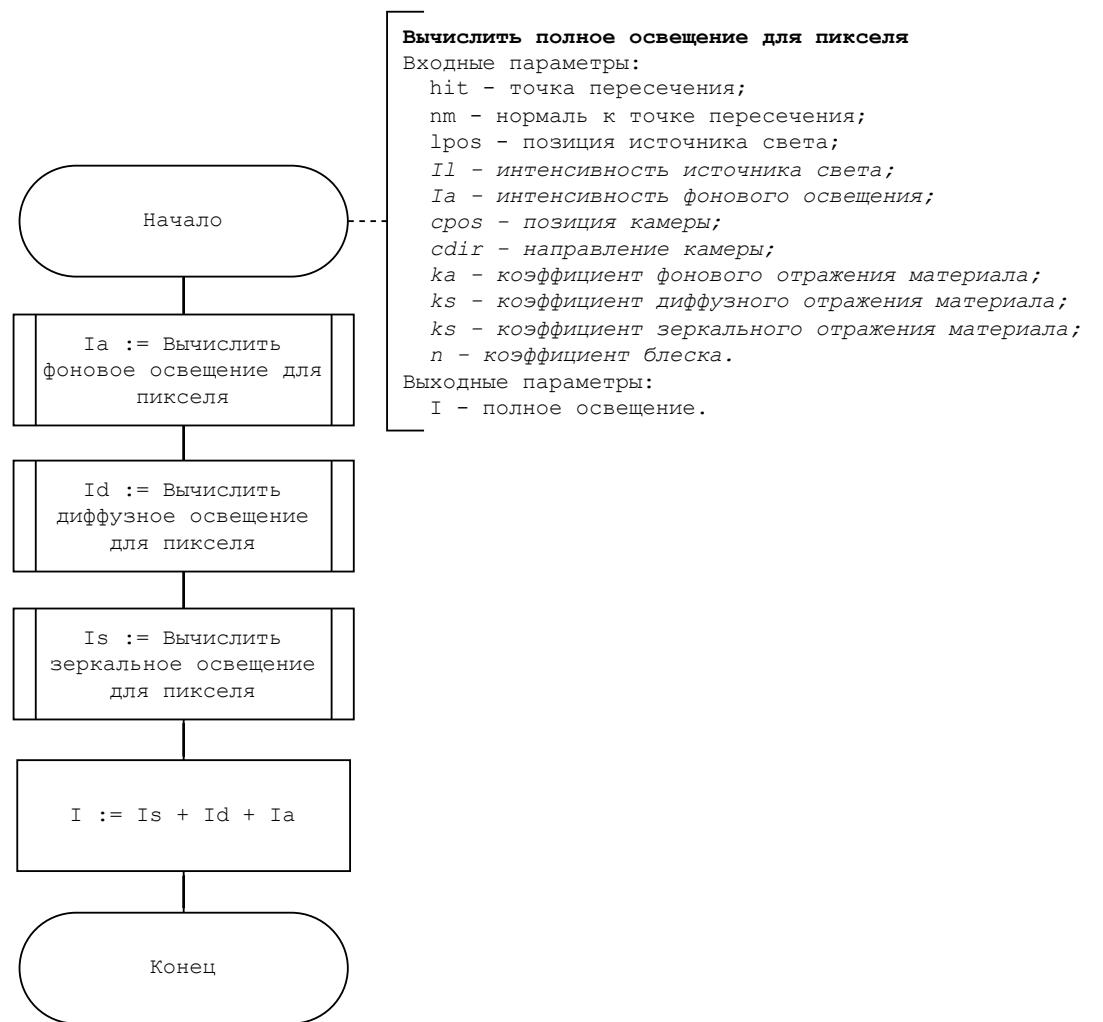


Рисунок 2 – Схема алгоритма вычисления зеркального освещения для пикселя

Фоновое освещение можно вычислить по формуле

$$I_{ambient} = k_a \cdot I_a, \quad (8)$$

где k_a – коэффициент фонового освещения для материала; I_a – интенсивность фонового света.

На рисунке 3 изображена схема алгоритма вычисления фонового освещения для пикселя.

Зеркальное освещение можно вычислить по формуле

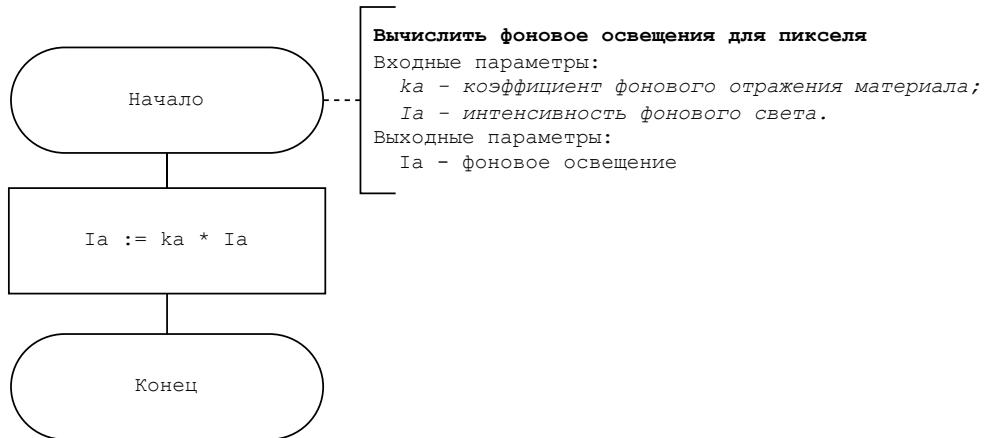


Рисунок 3 – Схема алгоритма вычисления фонового освещения для пикселя

$$I_{specular} = k_s \cdot I_L \cdot (\max(0, \mathbf{R} \cdot \mathbf{V}))^n, \quad (9)$$

где:

- k_s – коэффициент зеркального отражения материала;
- $\mathbf{R} = 2 \cdot (\mathbf{N} \cdot \mathbf{L}) \cdot \mathbf{N} - \mathbf{L}$ – вектор отражения, который вычисляется как отражение вектора \mathbf{L} относительно нормали \mathbf{N} ;
- \mathbf{V} – единичный вектор, направленный от точки пересечения луча к камере;
- n – коэффициент блеска (или экспонента);
- $\max(0, \mathbf{R} \cdot \mathbf{V})$ – эта функция гарантирует, что только те блики, которые направлены в сторону камеры, будут учтены.

На рисунке 4 изображена схема алгоритма вычисления зеркального освещения для пикселя.

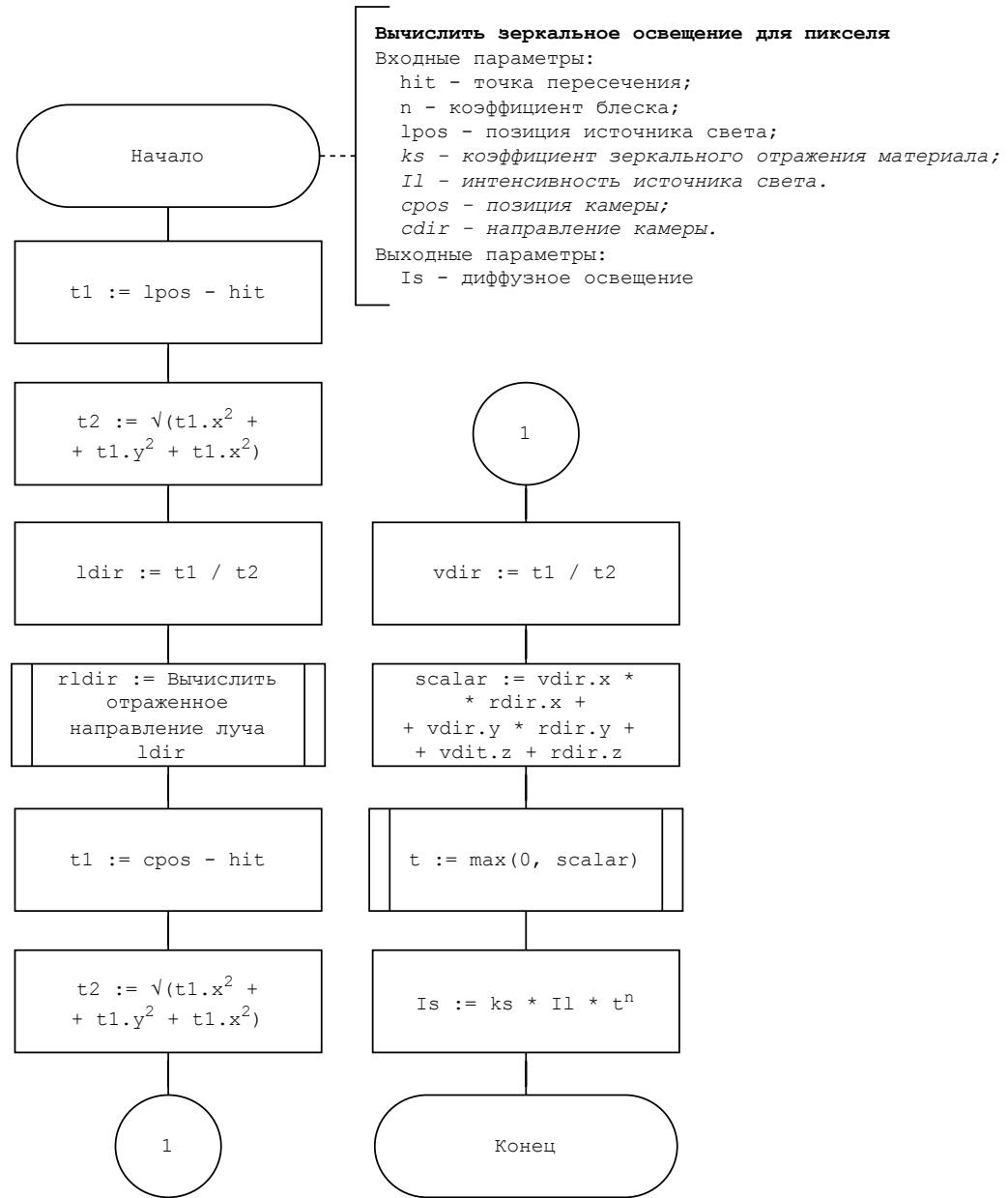


Рисунок 4 – Схема алгоритма вычисления зеркального освещения для пикселя

Диффузное освещение можно вычислить по формуле

$$I_{diffuse} = k_d \cdot I_L \cdot \max(0, \mathbf{N} \cdot \mathbf{L}), \quad (10)$$

где:

- k_d — коэффициент диффузного отражения материала;
- I_L — интенсивность источника света;

- \mathbf{N} — нормаль поверхности в точке пересечения луча;
- \mathbf{L} — единичный вектор, направленный от точки пересечения луча к источнику света.

На рисунке 5 изображена схема алгоритма вычисления диффузного освещения для пикселя.

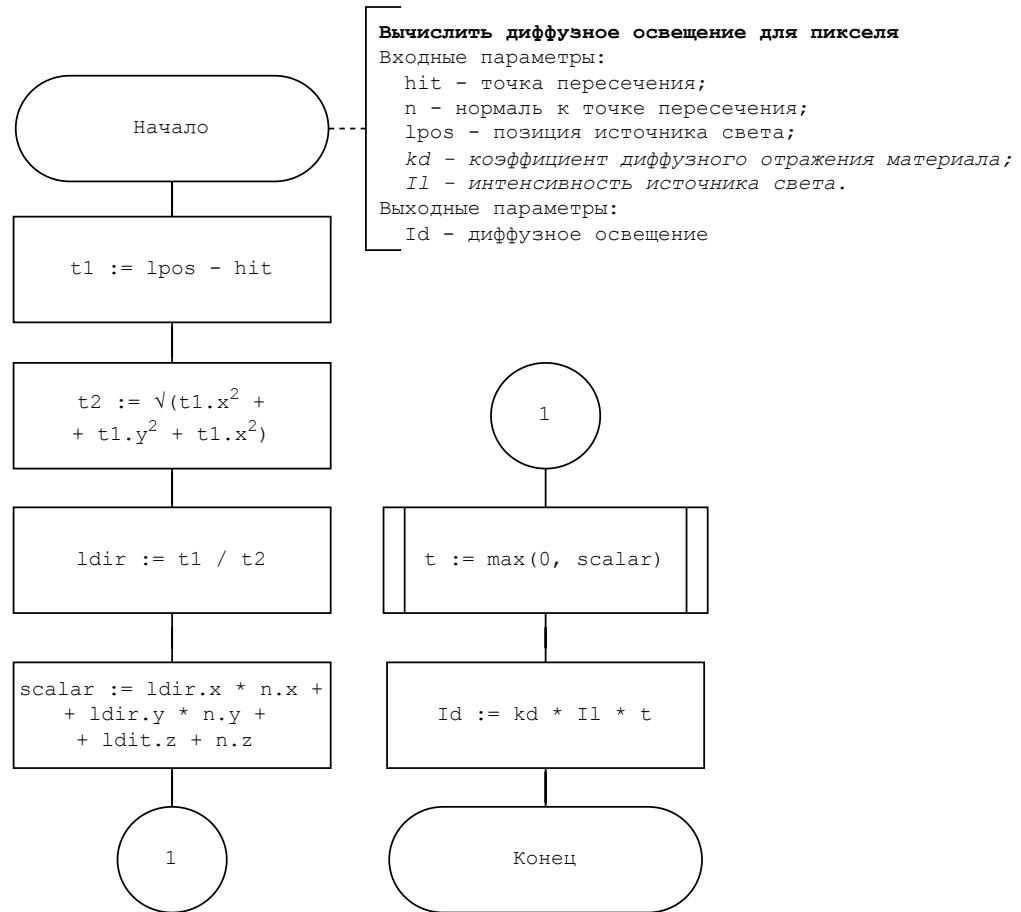


Рисунок 5 – Схема алгоритма вычисления диффузного освещения для пикселя

2.2.2 Обратная трассировка лучей

Для разработки алгоритма обратной трассировки лучей необходимо описать, как вычисляются пересечение луча с объектами сцены, преломление и отражение луча.

Пересечение луча с объектом

Уравнение прямой описывается следующей формулой [3, с. 140].

$$P(t) = S + t \cdot V, \quad (11)$$

где $P(t)$ – точка на прямой, которая выражается с помощью параметра t (меняется в диапазоне от минимального до максимального значения для задания всей прямой); V – вектор направления прямой, который определяет, в каком направлении прямая распространяется от начальной точки S .

Тогда значение параметра t , соответствующее точке пересечения, например, с плоскостью **параллелепипеда**, заданным формулой (1.1), $x = r_x$, можно вычислить следующим образом [3, с. 143].

$$t = \frac{r_x - S_x}{V_x}, \quad (12)$$

при этом координаты y и z точки пересечения внутри граней параллелограмма должны удовлетворять условиям:

$$0 \leq [P(t)]_y \leq r_y; 0 \leq [P(t)]_z \leq r_z. \quad (13)$$

Значение параметра t соответствующее точке пересечения с плоскостями параллелепипеда $z = r_z$, $y = r_y$ вычисляется аналогично.

Уравнение (1.1), задающее **цилиндр**, можно привести к следующему виду [3, С. 145–146]:

$$(V_x^2 + m^2 V_y^2)t^2 + 2(S_x V_x + m^2 S_y V_y)t + S_x^2 + m^2 S_y^2 - r^2 = 0. \quad (14)$$

Решение этого уравнение дает значение параметра t , где луч пересекает бесконечный цилиндр. Полученные точки пересечения должны быть проверены по z -координатам, чтобы удовлетворять условию:

$$0 \leq z \leq h, \quad (15)$$

где h – высота цилиндра.

На рисунке 6 изображена схема алгоритма нахождения точки пересечения луча с цилиндром.

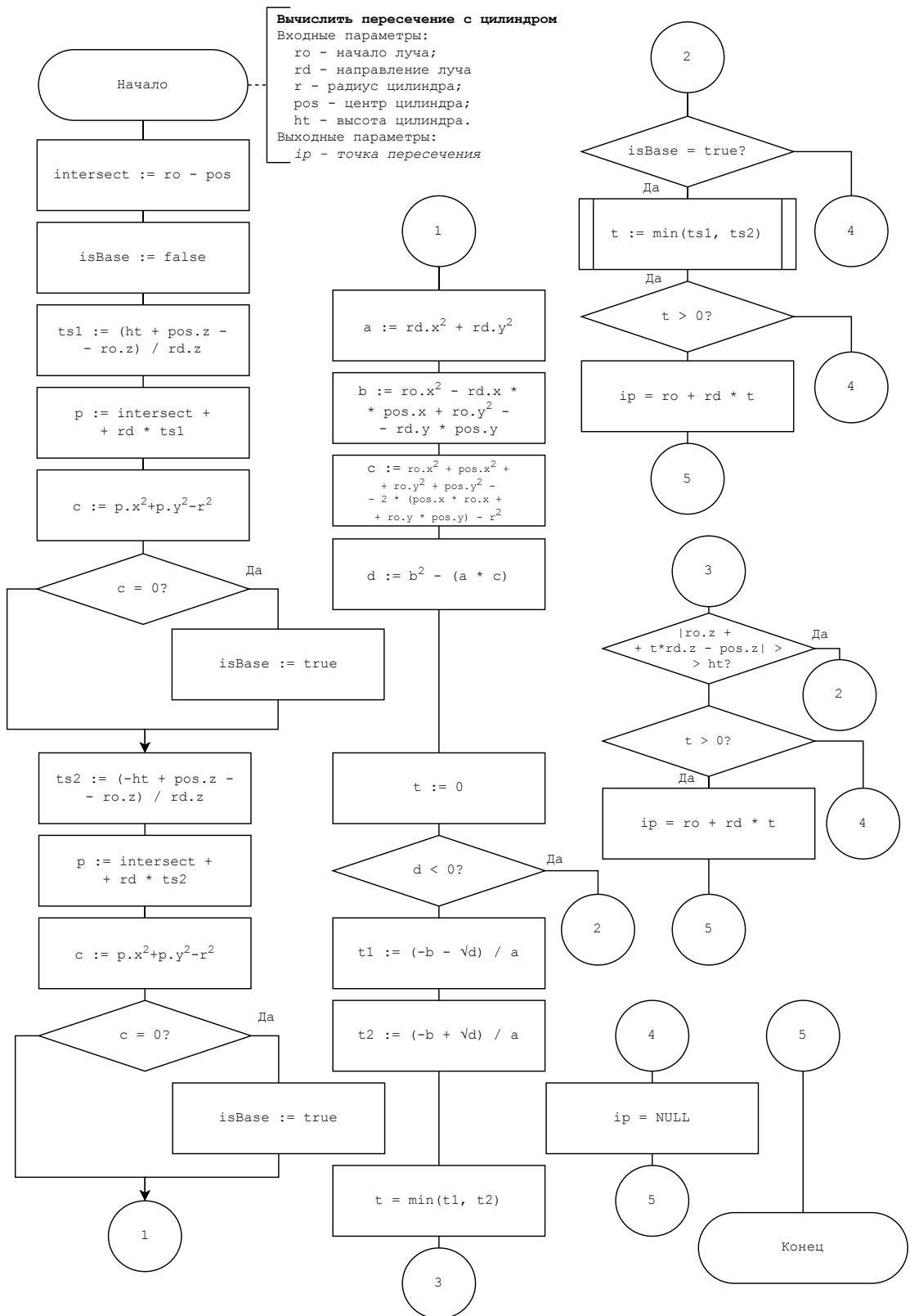


Рисунок 6 – Схема алгоритма нахождения точки пересечения луча с цилиндром

На рисунке 7 изображена схема алгоритма нахождения точки пересечения луча с параллелепипедом.

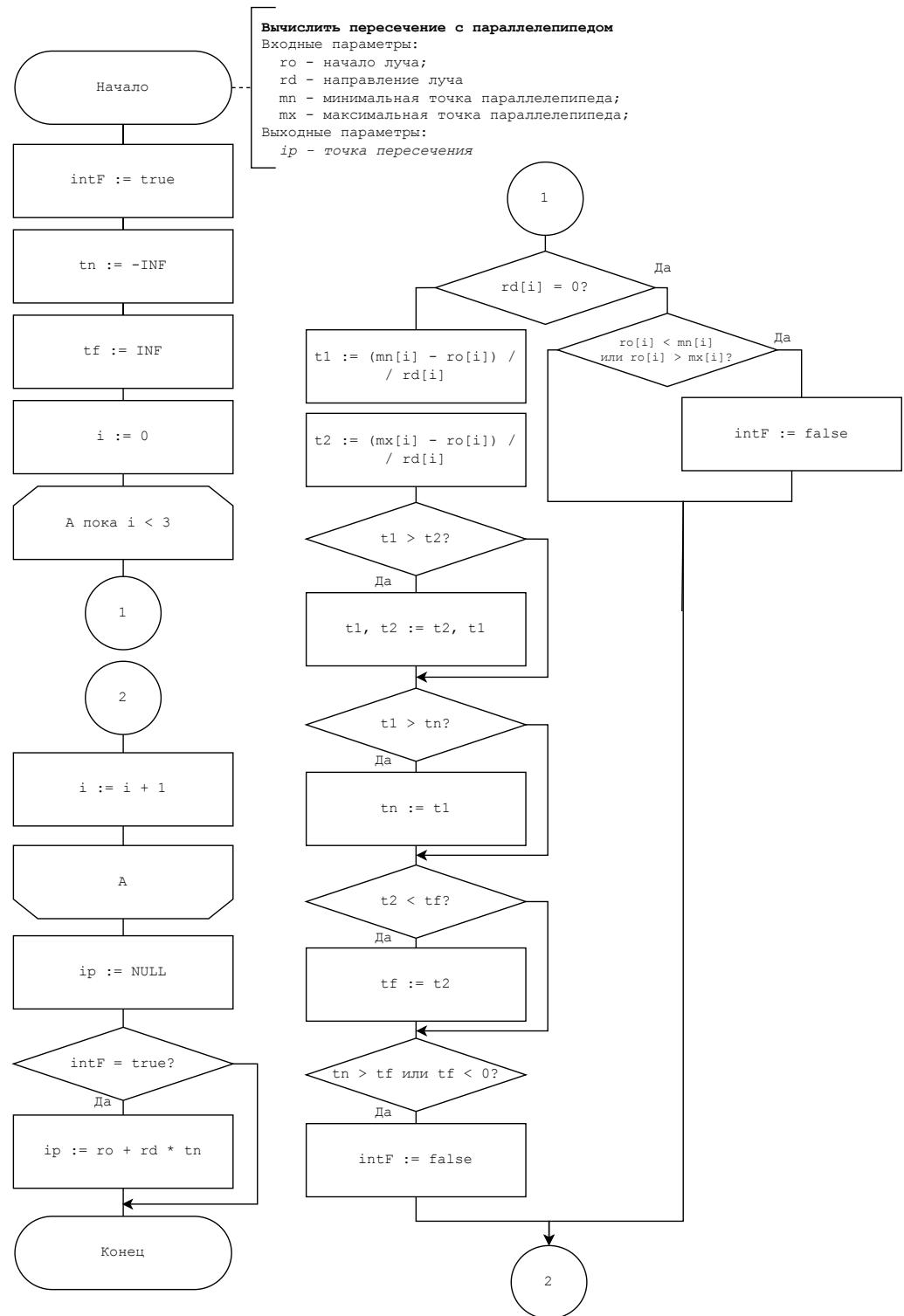


Рисунок 7 – Схема алгоритма нахождения точки пересечения луча с параллелепипедом

Отражение луча

Направление отражения света от блестящей поверхности подчиняется правилу: угол падения равен углу отражения, как показано на рисунке 8 [3, с. 150].

Пусть векторы N и L нормализованы – единичной длины. Тогда вектор R – направление отраженного луча, можно вычислить следующий образом:

$$R = 2(N \cdot L)N - L. \quad (16)$$

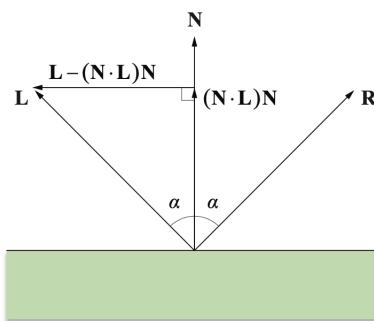


Рисунок 8 – Отражение луча от поверхности [3, с. 150]

На рисунке 9 изображена схема алгоритма вычисления отраженного луча.

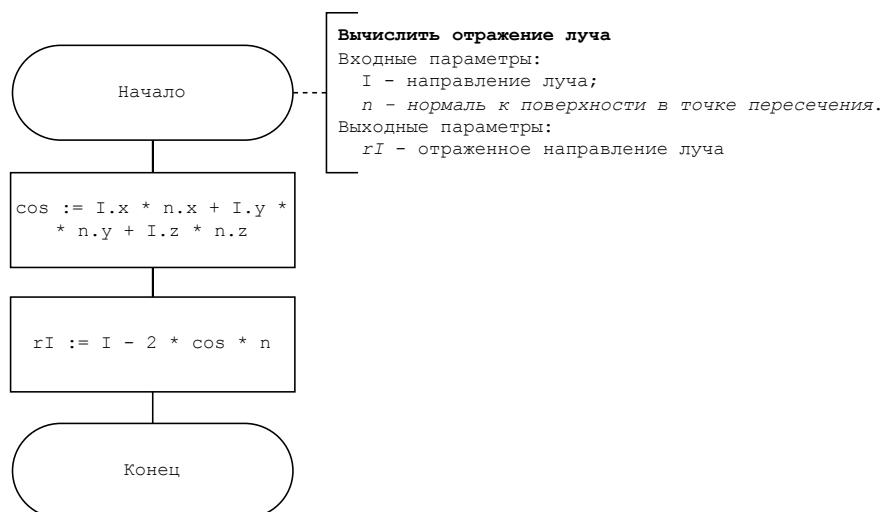


Рисунок 9 – Схема алгоритма вычисления отражения луча от поверхности

Преломление луча

Согласно закону Снеллиуса, как показано на рисунке 10, угол падения θ_L и угол преломления θ_T для двух сред с показателями преломления η_L и η_T связаны уравнением [3, с. 151]:

$$\eta_L \sin(\theta_L) = \eta_T \sin(\theta_T). \quad (17)$$

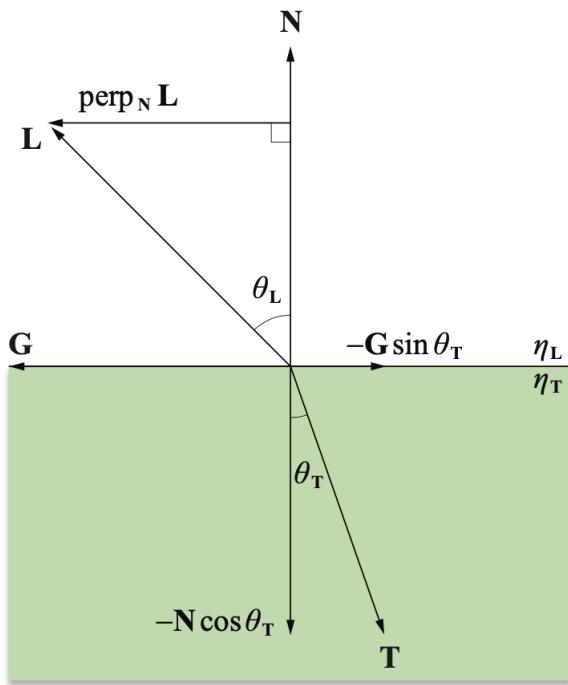


Рисунок 10 – Преломление луча, на границе двух сред [3, с. 152]

Пусть векторы N и L нормализованы – единичной длины. Тогда вектор T – направление преломления, можно вычислить следующий образом:

$$T = \left(\frac{\eta_L}{\eta_T} N \cdot L - \sqrt{1 - \frac{\eta_L^2}{\eta_T^2} [1 - (N \cdot L)^2]} \right) N - \frac{\eta_L}{\eta_T} L. \quad (18)$$

На рисунке 11 изображена схема алгоритма вычисления преломления луча.

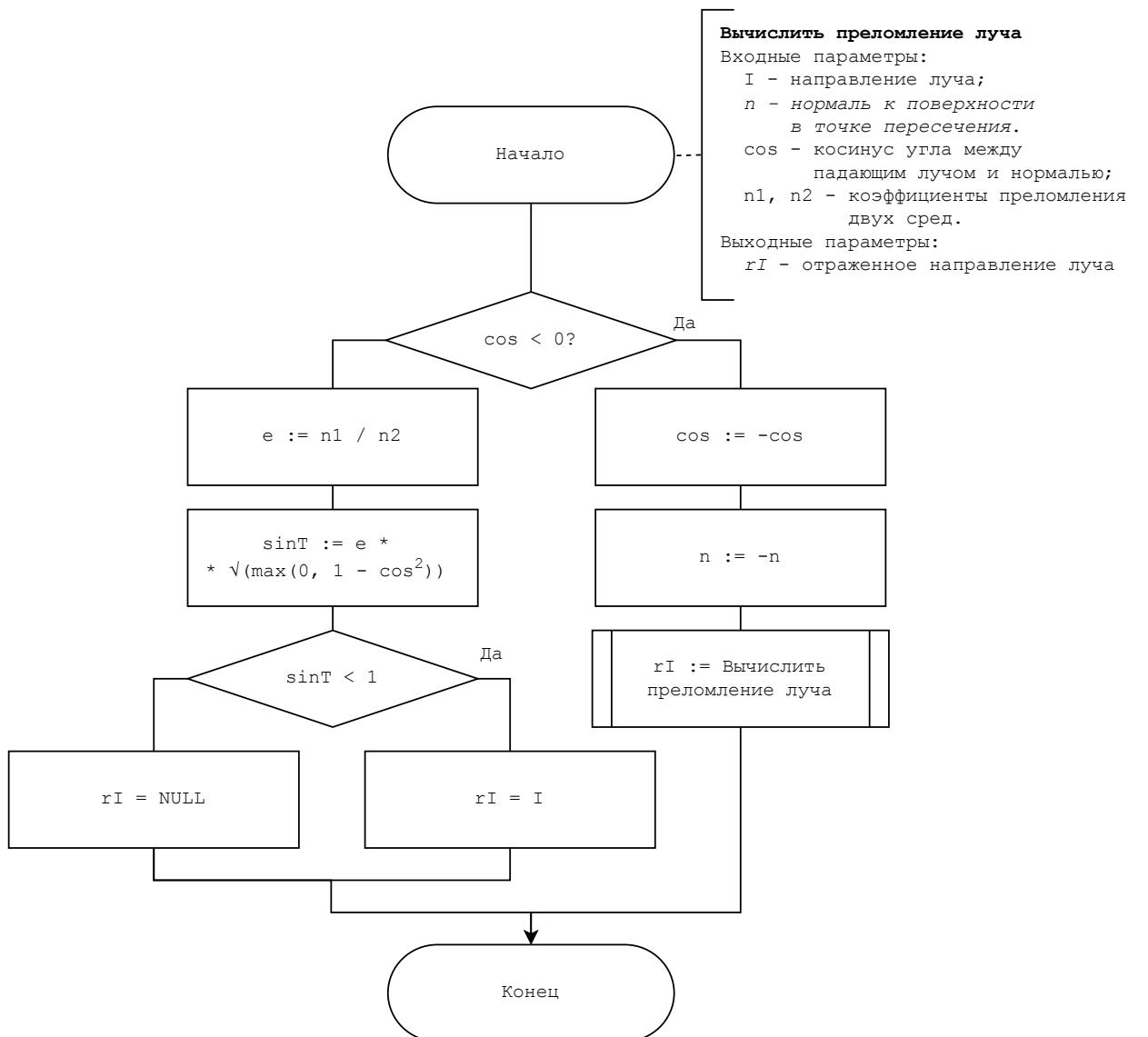


Рисунок 11 – Схема алгоритма вычисления преломления луча

Алгоритм прямой трассировки лучей

На рисунке 12 изображена схема обобщенного алгоритма обратной трассировки луча для вычисления цвета одного пикселя изображения.

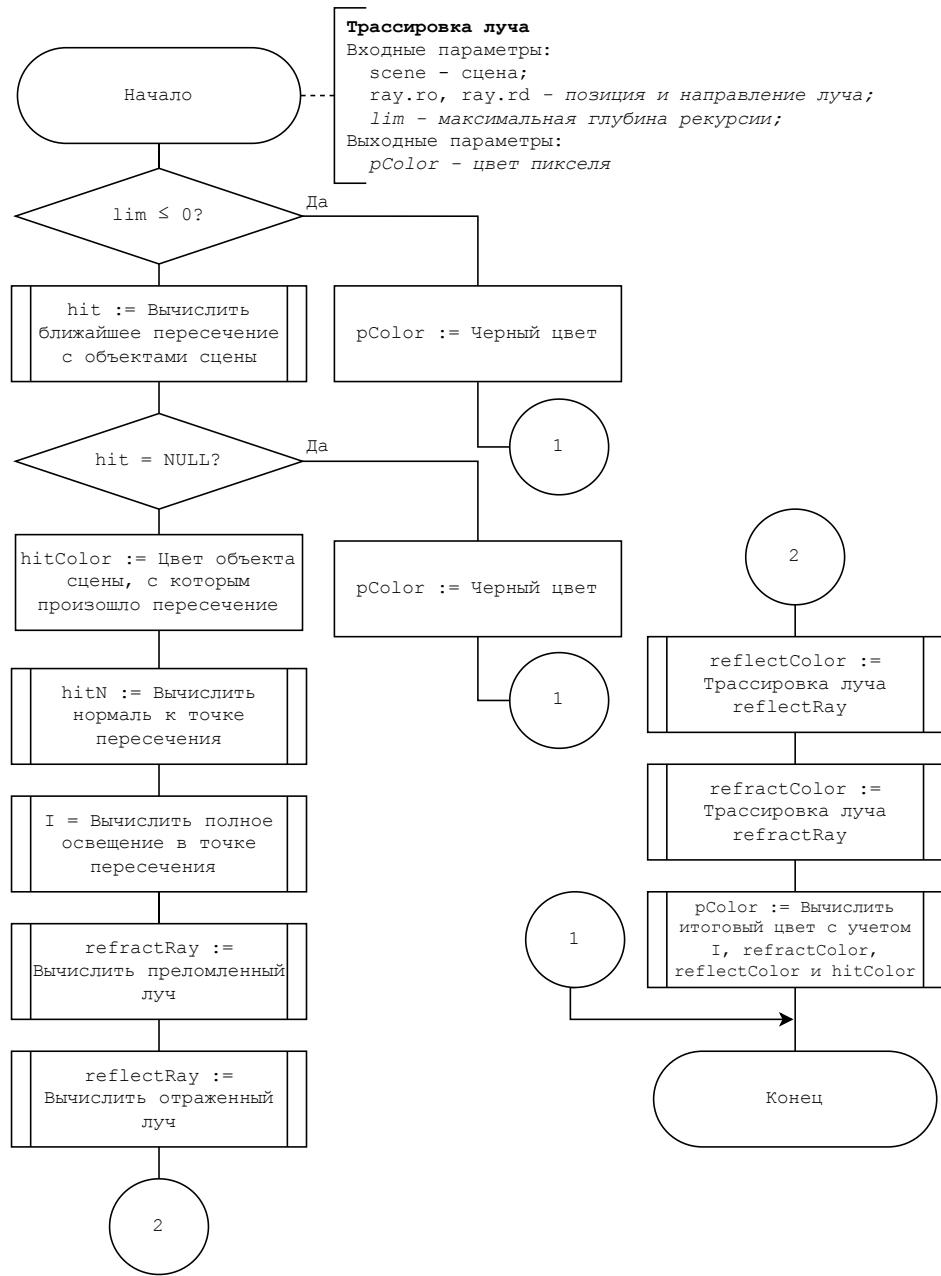


Рисунок 12 – Схема алгоритма трассировки луча

На рисунке 13 изображена схема алгоритма обратной трассировки лучей для вычисления цвета всех пикселей изображения.

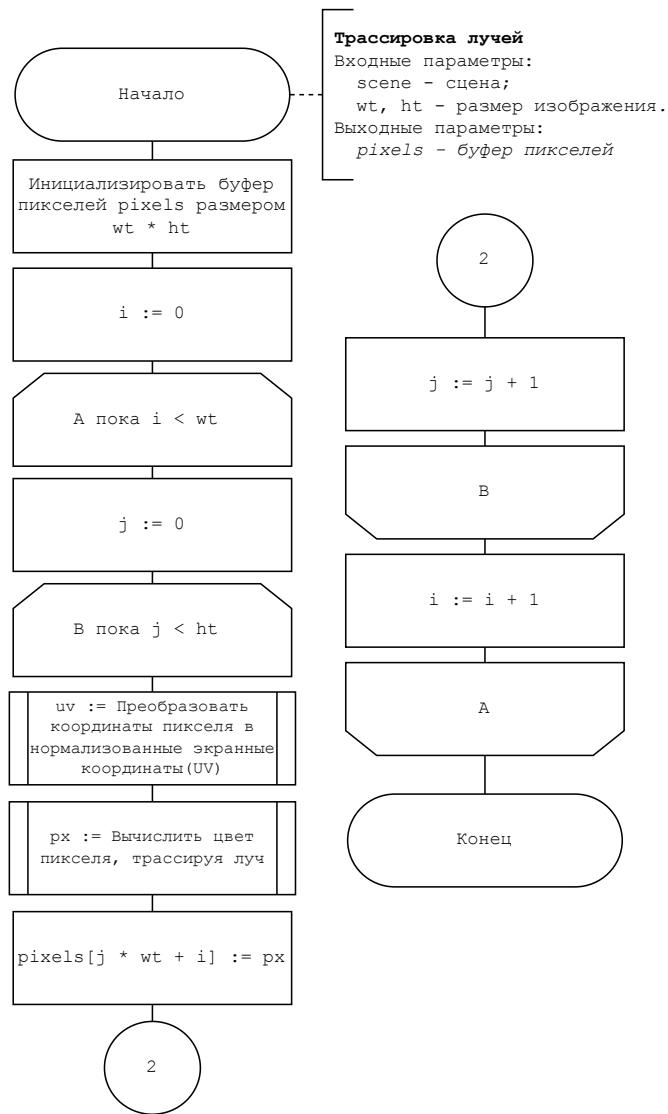


Рисунок 13 – Схема алгоритма трассировки лучей

2.2.3 Общий алгоритм работы программы

На рисунке 14 представлена схема общего алгоритма работы программы.

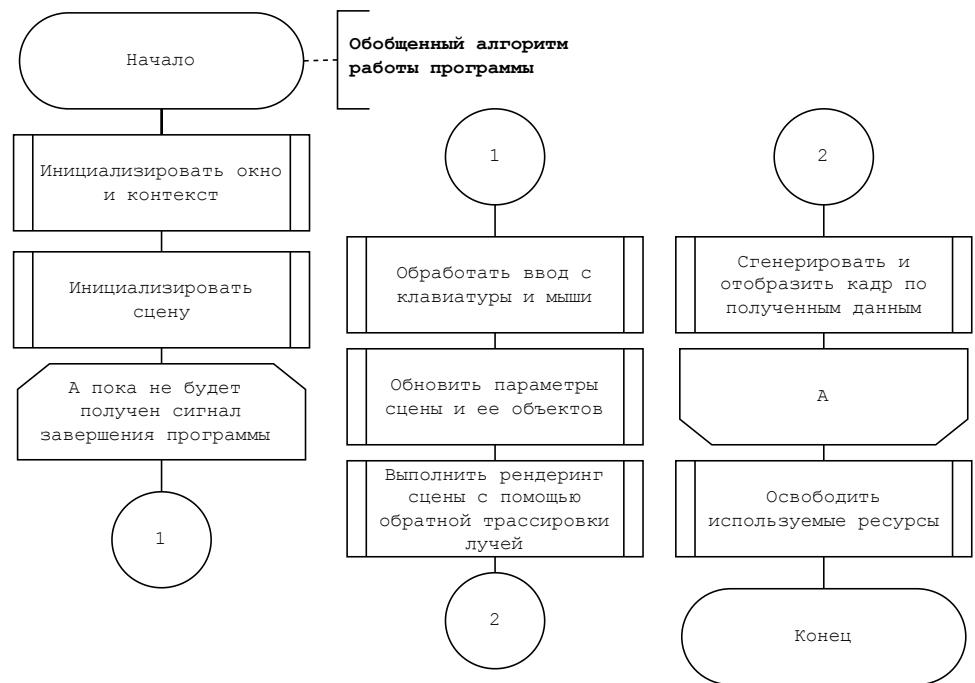


Рисунок 14 – Схема общего алгоритма работы программы

2.3 Используемые типы и структуры данных

В таблице 3 описаны используемые типы и структуры данных.

Таблица 3 – Используемые типы и структуры данных (начало)

Объект	Представление / тип данных
Трехмерный вектор	Структура vec3 с полями: x, y, z – числа с плавающей точкой.
Цвет	Структура ColorRGB с полями: r, g, b – числа с плавающей точкой.
Луч	Структура Ray с полями orig и dir – трехмерные векторы.
Источник света	Структура Light с полем pos – трехмерный вектор.

Таблица 3 – Используемые типы и структуры данных (окончание)

Объект	Представление / тип данных
Камера	Структура Camera с полями: pos – трехмерный вектор; yaw, pitch, fov – числа с плавающей точкой.
Тело	Структура Shape с полями: pos (трехмерный вектор); color (цвет); reflect (коэффициент отражения света), refract (коэффициент пропускания света), n (коэффициент преломления света) – числа с плавающей точкой.
Плоскость	Структура Plane с полями структуры Shape и полем height – число с плавающей точкой.
Цилиндр	Структура Cylinder с полями структуры Shape и полями height, radius – числа с плавающей точкой.
Параллелепипед	Структура Box с полями структуры Shape и полями min, max – трехмерные векторы.
Пересечение с объектом	Структура RayHit с полями: pos, nrm – трехмерные векторы; n – число с плавающей точкой.
Сцена	Структура Scene с полями: camera (камера), light (источник света), shapes (тела) – массив структур Shape.

Вывод

В данном разделе были приведены функциональные требования к программному обеспечению, разработаны схемы алгоритмов, описаны типы и структуры данных, которые будут использованы при разработке.

3 Технологическая часть

Данный раздел обосновывает выбор средств реализации, которые будут использоваться при разработке программного обеспечения. Будет продемонстрирован пользовательский интерфейс, как демонстрация работы программы, а также описана структура разрабатываемого программного обеспечения с помощью UML–диаграммы классов.

3.1 Средства реализации

Язык программирования *C++* [9] выбран для разработки программного обеспечения, так как средствами языка можно реализовать разработанные алгоритмы и все необходимые типы и структуры данных, описанные в таблице 3, а так же из-за моего опыта работы с данным языком.

В качестве среды разработки выбрана среда *Visual Studio Code* [10]. Эта среда предоставляет набор инструментов для написания и отладки кода на различных языках программирования, включая *C++*.

3.2 Структура программы

На рисунке 15 изображена структура программы в виде UML – диаграммы.

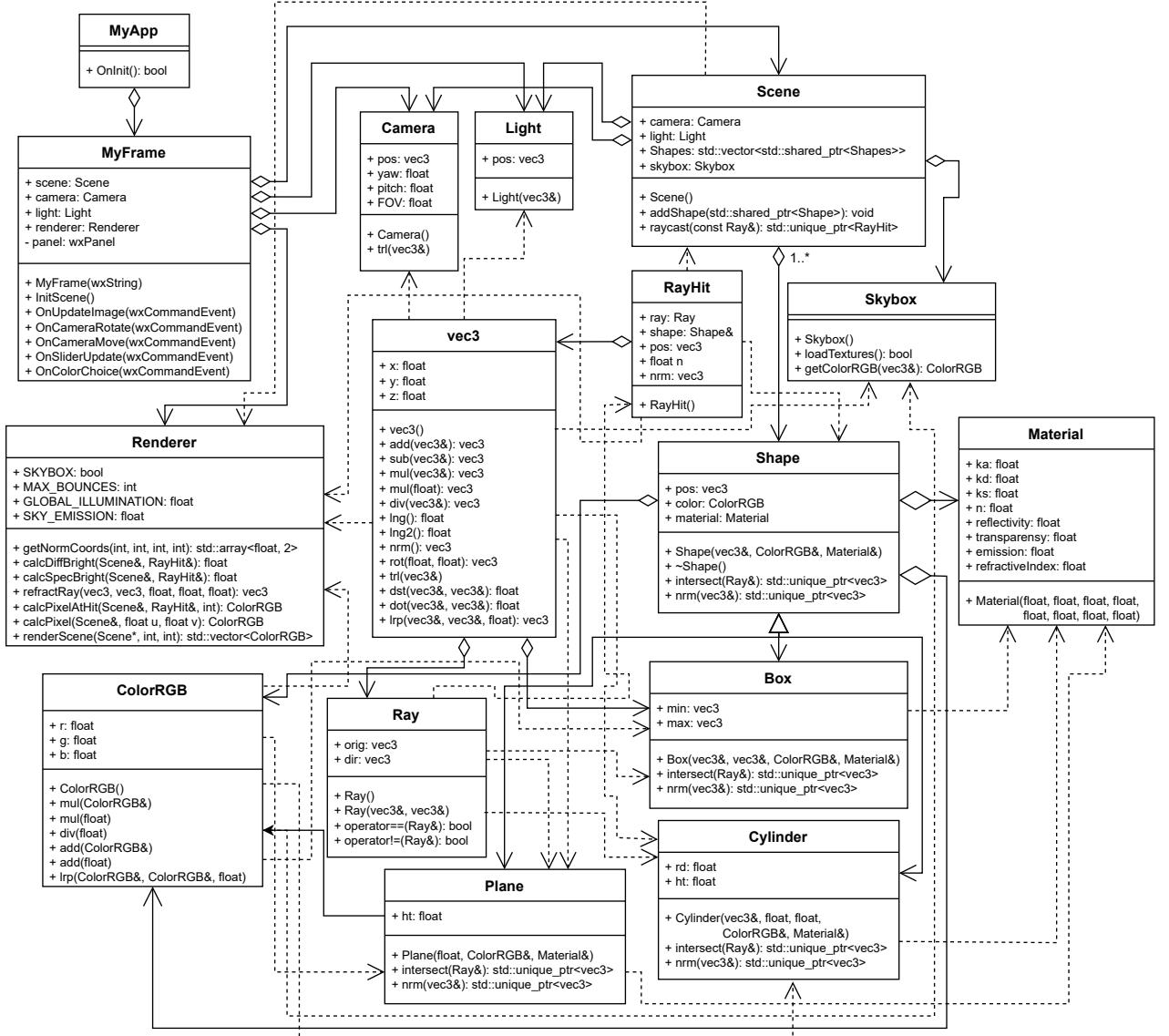


Рисунок 15 – Структура программы

3.3 Реализации алгоритмов

На листингах 1 и 2 представлены реализации алгоритмов вычисления пересечения луча с цилиндром и параллелепипедом, соответственно.

Листинг 1 – Реализация алгоритма поиска пересечения луча с цилиндром

```

1 std :: unique_ptr<vec3> Cylinder :: intersect( const Ray& r ) const {
2     vec3 intp = r.orig - pos;  bool isBase = false;
3     float ts1 = (ht - r.orig.y + pos.y) / (r.dir.y + EPS);
4     vec3 point = intp + r.dir.mul(ts1);
5     if ((point.x*point.x+point.z*point.z - rd*rd<EPS)) isBase=true;

```

```

6   float ts2 = (-ht - r.orig.y + pos.y) / (r.dir.y + EPS);
7   point = intp + r.dir.mul(ts2);
8   if (point.x*point.x+point.z*point.z-rd*rd<EPS) isBase=true;
9   float a = r.dir.x * r.dir.x + r.dir.z * r.dir.z;
10  float b = r.orig.x * r.dir.x - r.dir.x * pos.x + r.orig.z *
11      r.dir.z - r.dir.z * pos.z;
12  float c = r.orig.x * r.orig.x + pos.x * pos.x + r.orig.z *
13      r.orig.z + pos.z * pos.z - 2 * (pos.x * r.orig.x +
14      r.orig.z * pos.z) - rd * rd;
15  float delta = b * b - (a * c); float t = 0.0f;
16  if (delta < EPS) { if (isBase) {
17      t = std::min(ts1, ts2);
18      if ((!(abs(r.orig.y+t*r.dir.y-pos.y)>ht))&&(t>0.0))
19          return std::make_unique<vec3>(r.orig+(r.dir.mul(t)));
20  } else return nullptr; }
21  float t1 = (-b-sqrt(delta))/a; float t2 = (-b+sqrt(delta))/a;
22  t = (t1 < 0.0) ? t2 : t1;
23  if ((!(abs(r.orig.y + t * r.dir.y - pos.y) > ht))&&(t > 0.0))
24      return std::make_unique<vec3>(r.orig+(r.dir.mul(t)));
25  if (!isBase) return nullptr;
26  t = std::min(ts1, ts2);
27  if ((!(abs(r.orig.y + t * r.dir.y - pos.y) > ht))&&(t > 0.0))
28      return std::make_unique<vec3>(r.orig+(r.dir.mul(t)));
29  return nullptr;
30 }

```

Листинг 2 – Реализация алгоритма поиска пересечения луча с параллелепипедом

```

1 std::unique_ptr<vec3> Box::intersect (const Ray& r) const {
2     float t1, t2; float tn = -INFINITY; float tf = INFINITY;
3     bool intersection = true;
4     std::array<float, 3> rDirArr = r.dir.tArr();
5     const float* rDir = rDirArr.data();
6     std::array<float, 3> rayOriginArr = r.orig.tArr();
7     const float* rayOrigin = rayOriginArr.data();
8     std::array<float, 3> b1Arr = min.tArr();
9     const float* b1 = b1Arr.data();
10    std::array<float, 3> b2Arr = max.tArr();
11    const float* b2 = b2Arr.data();
12    for (int i = 0; i < 3; ++i) {
13        if (rDir[i]==0.0f) {

```

```

14         if (rayOrigin[i] < b1[i] || rayOrigin[i] > b2[i])
15             intersection = false;
16     } else {
17         t1=(b1[i] - rayOrigin[i]) / rDir[i];
18         t2=(b2[i] - rayOrigin[i]) / rDir[i];
19         if (t1 > t2) std::swap(t1, t2);
20         if (t1 > tn) tn = t1; if (t2 < tf) tf = t2;
21         if (tn > tf) intersection = false;
22         if (tf < 0) intersection = false;
23     }
24 }
25 return intersection ? std::make_unique<vec3>(r.orig +
26 r.dir.mul(tn)) : nullptr;
}

```

На листинге 3 представлена реализация класса *Renderer*, которая включает в себя реализации следующих алгоритмов: вычисление полного освещения для пикселя, трассировка лучей и вычисление преломления луча.

Листинг 3 – Реализация класса *Renderer*

```

1 std::array<float, 2> Renderer::getNormCoords(int x, int y, int
wt, int ht) {
2     std::array<float, 2> uv;
3     float u, v;
4     if (wt > ht) {
5         u = (x - wt / 2 + ht / 2) / static_cast<float>(ht)*2 - 1;
6         v = -(y / static_cast<float>(ht) * 2 - 1);
7     } else {
8         u = x / static_cast<float>(wt) * 2 - 1;
9         v = -((y - ht / 2 + wt / 2) / static_cast<float>(wt)*2 - 1);
10    }
11    uv[0] = u;   uv[1] = v;
12    return uv;
13 }
14 vec3 Renderer::refractRay(vec3 I, vec3 n, float cos, float n1,
15 float n2) {
16     if (cos < 0.0f) return refractRay(I, n, -cos, n2, n1);
17     float eta = n1/n2;
18     float sin = eta * sqrtf(std::max(0.0f, 1.0f - cos * cos));
19     if (sin >= 1.0f) return vec3(-1.0f, 0.0f, 0.0f);
20     return I*eta+n*(eta*cos-sqrtf(std::max(0.0f,1.0f-sin*sin)));
21 }

```

```

21 float Renderer::calcDiffBright(Scene& scene, RayHit& hit) {
22     vec3 lightDir = (scene.light.pos - hit.pos).nrm();
23     Ray shadowRay(hit.pos + lightDir * 0.001f, lightDir);
24     auto hs = &(hit.shape); float accumulatedLight = 1.0f;
25     int bounceCount = 0; float n = 1.0;
26     while (bounceCount < MAX_BOUNCES) {
27         ++bounceCount;
28         auto lightBlocker = scene.raycast(shadowRay);
29         if (lightBlocker && (lightBlocker->pos -
30             scene.light.pos).lsg2() < EPS) {
31             return std::max(GLOBAL_ILLUMINATION,
32                             std::min(1.0f, vec3::dot(hit.nrm, lightDir)) *
33                             accumulatedLight);
34         }
35         if (!lightBlocker) {
36             return std::max(GLOBAL_ILLUMINATION,
37                             std::min(1.0f, vec3::dot(hit.nrm, lightDir)) *
38                             accumulatedLight);
39         }
40         if (&(lightBlocker->shape) == hs) {
41             if (vec3::dot(lightBlocker->nrm, lightDir) > 0.0f) {
42                 shadowRay.orig = lightBlocker->pos + lightDir *
43                     0.001f;
44                 continue;
45             } else return GLOBAL_ILLUMINATION;
46         }
47         float transparency =
48             (lightBlocker->shape).material.transparency * 0.9;
49         if (transparency > 0.0f) {
50             accumulatedLight *= transparency;
51             if (accumulatedLight < 0.01f) return
52                 GLOBAL_ILLUMINATION;
53             vec3 refrRay = refractRay(lightDir,
54                                         lightBlocker->nrm, vec3::dot(lightDir,
55                                         lightBlocker->nrm),
56                                         lightBlocker->shape.material.refractiveIndex, n);
57             shadowRay.orig = lightBlocker->pos + lightDir *
58                 0.001f;
59             shadowRay.dir = refrRay.nrm();
60         } else return GLOBAL_ILLUMINATION;
61         hs = &(lightBlocker->shape);

```

```

52    }
53    return std::max(GLOBAL_ILLUMINATION,
54        std::min(1.0f, vec3::dot(hit.nrm, lightDir)) *
55        accumulatedLight);
56}
57float Renderer::calcSpecBright(Scene& scene, RayHit& hit) {
58    vec3 lightDir = (hit.pos - (scene.light.pos).nrm()).nrm();
59    float specularFactor = std::max(0.0f, vec3::dot(lightDir -
60        hit.nrm * (vec3::dot(lightDir, hit.nrm) * 2),
61        (scene.camera.pos - hit.pos).nrm()));
62    return std::pow(specularFactor, 2.0f) *
63        hit.shape.material.reflectivity;
64}
65ColorRGB Renderer::calcPixelAtHit(Scene& scene, RayHit& hit, int
66    limit) {
67    const ColorRGB hitColorRGB = hit.shape.color;
68    const float dBright = calcDiffBright(scene, hit);
69    const float sBright = calcSpecBright(scene, hit);
70    const float reflectivity = hit.shape.material.reflectivity;
71    const float emission = hit.shape.material.emission;
72    const float transparency = hit.shape.material.transparency;
73    const float n1 = hit.n; const float n2 =
74        hit.shape.material.refractiveIndex;
75    ColorRGB reflectionColor; ColorRGB refractionColor;
76    float dirCos = vec3::dot(hit.ray.dir, hit.nrm);
77    vec3 reflectRayDir = hit.ray.dir - (hit.nrm * (2.0f *
78        dirCos));
79    vec3 reflectRayOrig = hit.pos + (reflectRayDir * 0.001f);
80    vec3 refractRayDir = refractRay(hit.ray.dir, hit.nrm, dirCos,
81        n2, n1);
82    vec3 refractRayOrig = hit.pos + (refractRayDir * 0.001f);
83    std::unique_ptr<RayHit> reflectHit = (limit > 0) ?
84        scene.raycast(Ray(reflectRayOrig, reflectRayDir)) :
85        nullptr;
86    if (reflectHit) reflectionColor = calcPixelAtHit(scene,
87        *reflectHit, limit - 1);
88    else if (SKYBOX) reflectionColor =
89        scene.skybox.getColorRGB(reflectRayDir).mul(SKY_EMISSION);
90    else reflectionColor = ColorRGB::SKYBLUE;
91    std::unique_ptr<RayHit> refractHit = (limit > 0) ?
92        scene.raycast(Ray(refractRayOrig, refractRayDir)) :

```

```

    nullptr;

81 if (refractHit) refractionColor = calcPixelAtHit(scene ,
82     *refractHit , limit - 1);
83 else if (SKYBOX) refractionColor =
84     scene.skybox.getColorRGB(refractRayDir).mul(SKY_EMISSION);
85 else refractionColor = ColorRGB::SKYBLUE;
86 ColorRGB pixelColorRGB = ColorRGB::lrb(hitColorRGB ,
87     reflectionColor ,
88     reflectivity).mul(dBright).add(sBright).add(reflectionColor
89     .mul(reflectivity *emission))
90     .add(refractionColor.mul(transparency));
91 return pixelColorRGB;
92 }

93 ColorRGB Renderer::calcPixel(Scene& scene , float u, float v) {
94     vec3 eyePos(0.0f , 0.0f , (-1.0f / std::tan(scene.camera.fOV *
95         M_PI / 360.0f)));
96     Camera& cam = scene.camera;
97     vec3 rayDir = (vec3(u, v, 0.0f) - eyePos).nrm().rot(cam.yaw,
98         cam.pitch);
99     auto hitPtr = scene.raycast(Ray(eyePos + cam.pos , rayDir));
100    if (hitPtr) return calcPixelAtHit(scene , *hitPtr ,
101        MAX_BOUNCES);
102    else if (SKYBOX) return
103        scene.skybox.getColorRGB(rayDir).mul(SKY_EMISSION);
104    else return ColorRGB::SKYBLUE;
105 }

106 std::vector<ColorRGB> Renderer::renderScene(Scene* scene , int wt ,
107     int ht) {
108     std::vector<ColorRGB> pixels(wt * ht);
109     auto start = std::chrono::high_resolution_clock::now();
110     int numThreads = std::thread::hardware_concurrency();
111     if (numThreads == 0) numThreads = 8;
112     auto renderBlock = [&scene , &pixels , wt , ht](int startX , int
113         startY , int blockWidth , int blockHeight) {
114         for (int x = startX; x < std::min(startX + blockWidth ,
115             wt); ++x) {
116             for (int y = startY; y < std::min(startY +
117                 blockHeight , ht); ++y) {
118                 std::array<float , 2> uv = getNormCoords(x , y , wt ,
119                     ht);
120                 ColorRGB pixel = calcPixel(*scene , uv[0] , uv[1]);
121             }
122         }
123     };
124 }
```

```

106         pixels[(y * wt + x)] = pixel;
107     }
108 }
109 };
110 std::vector<std::thread> threads;
111 int blockWidth = (wt + numThreads - 1) / numThreads;
112 for (int t = 0; t < numThreads; ++t) {
113     int startX = (t % numThreads) * blockWidth; int startY =
114         0;
115     threads.emplace_back(renderBlock, startX, startY,
116                           blockWidth, ht);
117 }
118 for (auto& t : threads) { t.join(); }
119 auto end = std::chrono::high_resolution_clock::now();
120 std::chrono::duration<double> elapsed_seconds = end - start;
121 auto elapsed_ms =
122     std::chrono::duration_cast<std::chrono::milliseconds>(end -
123     - start);
124 std::cout << "Время рендеринга: " << elapsed_seconds.count()
125     << " секунд (" << elapsed_ms.count() << " мс)" <<
126     std::endl;
127 return pixels;
128 }
```

3.4 Графический интерфейс

На рисунках 16 и 17 изображены окна, представляющие графический интерфейс для пользователя.

Программа предоставляет графический интерфейс: окно для настроек рендеринга изображения; окно для отображения сгенерированного изображения. При запуске все виджеты имеют значения по умолчанию. Для того, чтобы открыть окно для отображения изображения, необходимо нажать на кнопку «Обновить изображение».

С помощью окна настроек можно выполнять следующие действия:

- изменять позицию и поворот камеры в сцене с помощью соответствующих кнопок на панелях «Позиция камеры» и «Поворот камеры», соответственно;

- изменять позицию источника света в сцене с помощью соответствующих полей на панели «Параметры источника света»;
- изменять цвет, отражаемость, прозрачность и коэффициент преломления для материалов таких объектов, как сосуд и жидкость, с помощью соответствующих ползунков и выпадающих списков на панелях «Параметры жидкости» и «Параметры сосуда»;
- изменять отражаемость и цвет материала для объекта – стержня с помощью соответствующих ползунка и выпадающего списка на панели «Параметры стержня»;
- изменять дополнительные параметры для рендеринга изображения на панели «Дополнительные параметры»;
- сгенерировать и обновить изображение для отображения с помощью кнопки «Обновить изображение».

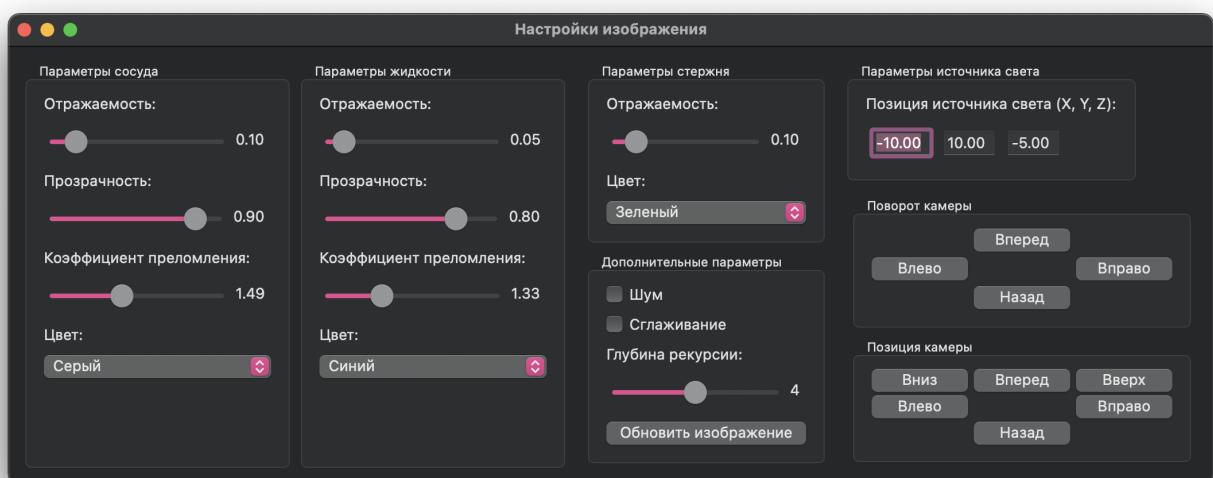


Рисунок 16 – Пользовательский интерфейс: окно для настроек рендеринга

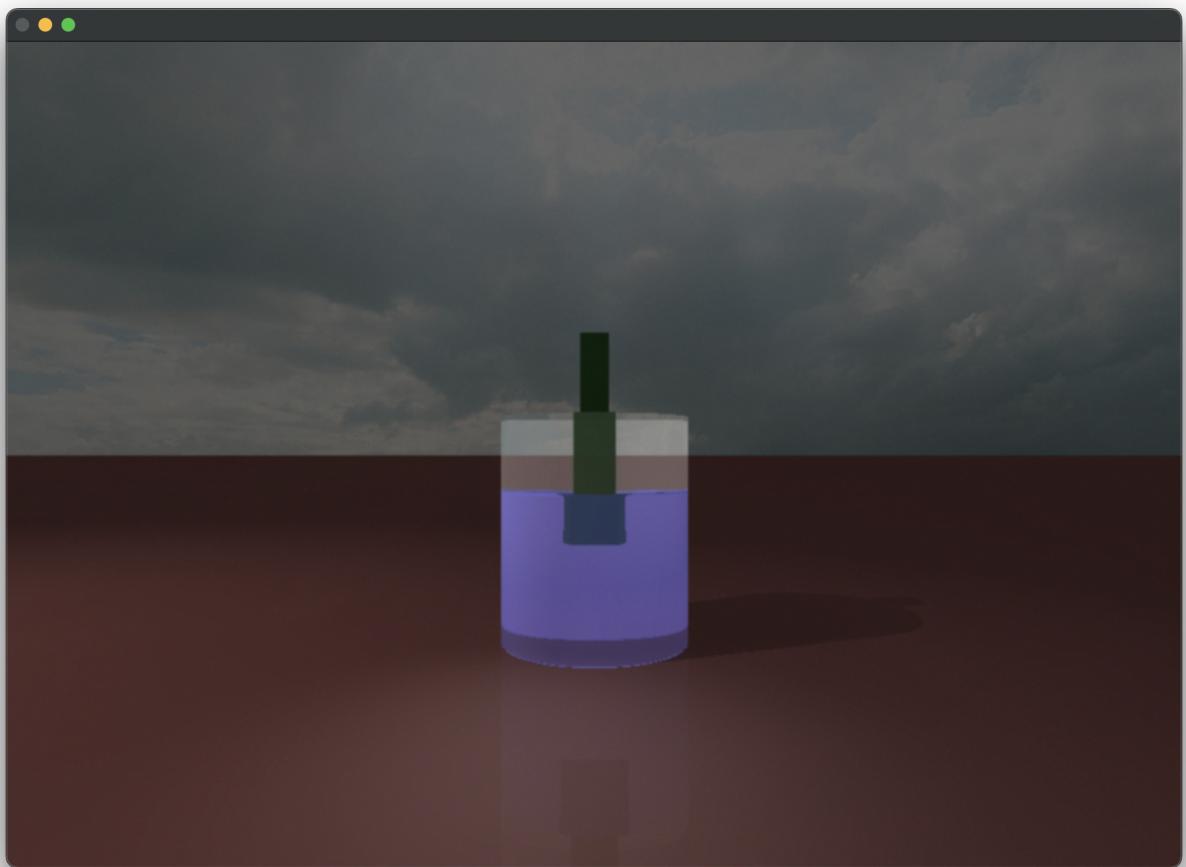


Рисунок 17 – Пользовательский интерфейс: окно для отображения изображения

Вывод

В данном разделе были выбраны средства реализации для программного обеспечения, приведена структура программы в виде UML – диаграммы классов, и предоставлены реализации разработанных алгоритмов. Также описан графический интерфейс, и приведена демонстрация работы программы.

4 Исследовательская часть

В данном разделе представлены описания исследований, проведенных для: оценки производительности программного обеспечения в зависимости от глубины рекурсии и от количества используемых дополнительных потоков при построении изображения, оценки реалистичности преломления света для прозрачных объектов в зависимости от входные параметров для материалов объектов; и сравнительный анализ результатов, полученных в ходе исследований.

4.1 Технические характеристики

Исследования проводилось на устройстве со следующими техническими характеристиками:

- операционная система macOS 14.6.1 [11];
- оперативная память (RAM) 16 ГБ;
- процессор (CPU) 2 ГГц 4-ядерный Intel Core i5 (8 логических ядер) [12].

Во время проведения исследования, устройство было подключено к сети электропитания и не было нагружено сторонними приложениями, за исключением встроенных приложений окружения.

4.2 Время построения изображения

Далее будет приведен сравнительный анализ для исследований производительности программного обеспечения в зависимости от глубины рекурсии и от количества используемых дополнительных потоков при построении изображения.

Для замеров процессорного и реального времени выполнения построения изображения используются функция `std::clock_t clock()` из библиотеки `<ctime>` и класс `std::chrono::high_resolution_clock` из библиотеки `<chrono>`, соответственно [9]. Разница между начальным и конечным временем использовалась для получения точных результатов в секундах. Для входных данных измерения времени выполнения проводились по 10 раз.

Влияние глубины рекурсии

В реализации алгоритма обратной трассировки лучей задается глубина рекурсии, которая влияет на время построения изображения, а также на реалистичность изображения прозрачных объектов. В исследование производились замеры процессорного времени.

В таблице 4 приведены результаты исследования времени построения изображения от глубины рекурсии. На рисунке 18 изображено графическое представление данных из таблицы 4.

Таблица 4 – Время построения изображения

Глубина рекурсии	Время выполнения (с)
0	2.709822
1	3.369566
2	3.967754
3	4.955640
4	7.129620
5	8.539870
6	11.750470
7	16.894780
8	24.755420

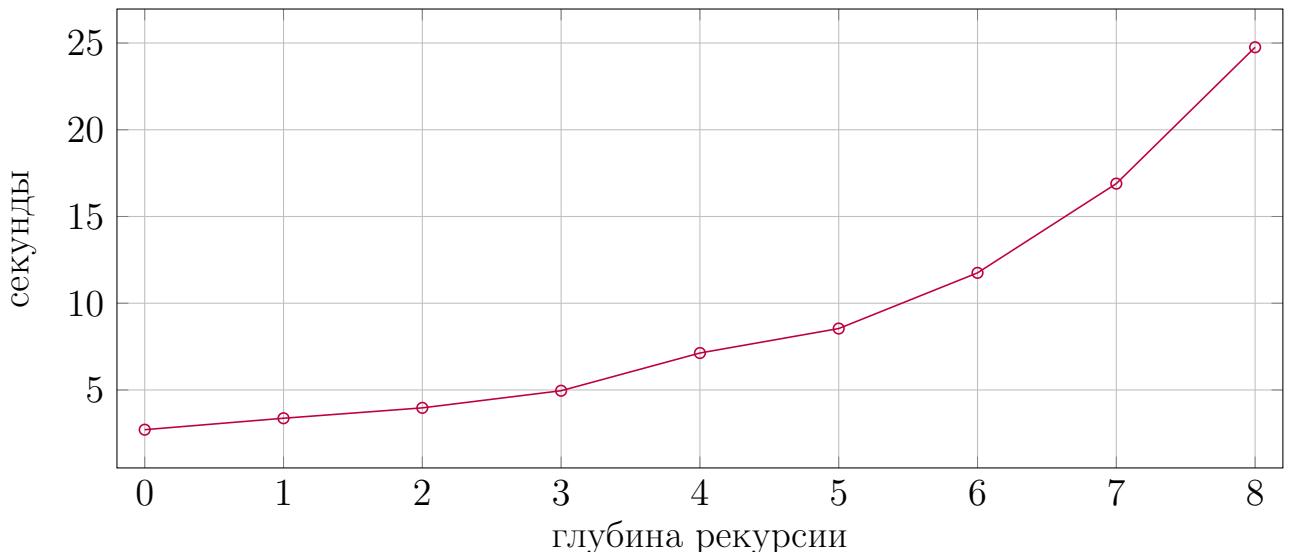


Рисунок 18 – Зависимость времени построения изображения от глубины рекурсии

Влияние использования дополнительных потоков

В реализации алгоритма построения изображения используются дополнительные потоки, чтобы ускорить процесс вычисления цвета всех пикселей изображения, следовательно, использование дополнительных потоков влияет на реальное время построения изображения.

В таблице 5 приведены результаты исследования времени построения изображения от количества используемых дополнительных потоков. На рисунке 19 изображено графическое представление данных из таблицы 5. Замеры времени проводились при глубине рекурсии равной 4.

Таблица 5 – Время построения изображения

Количество потоков	Время выполнения (с)
1	5.10565
4	2.27339
8	1.98203
16	1.73251
32	1.19489
64	1.08422
128	1.08445
256	1.09310
512	1.19786
1024	1.21875

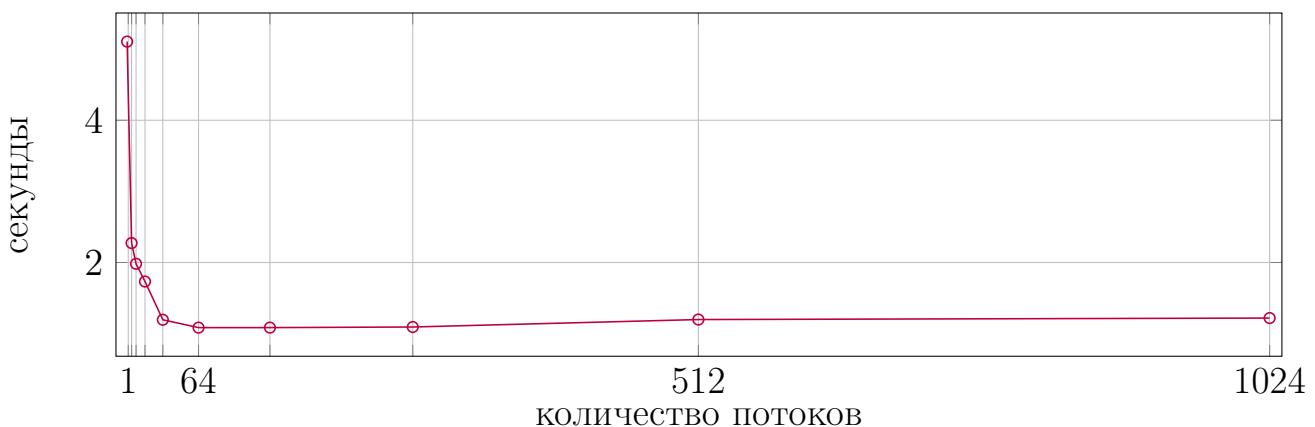


Рисунок 19 – Зависимость времени построения изображения от количества используемых потоков

4.3 Реалистичность преломления лучей

Целью данного исследования является оценка влияния изменения коэффициента преломления на визуализацию прозрачных объектов в сцене.

Изменяя коэффициент преломления для жидкости или стекла, можно наблюдать, как меняется угол преломления лучей и, соответственно, видимость объектов через эти материалы. Следовательно, было исследовано, как различные значения коэффициентов преломления для стекла и жидкости влияют на искажения изображения объектов в сцене.

Были рассмотрены несколько комбинаций коэффициентов преломления для стекла и жидкости, которые описаны в таблице 6.

Таблица 6 – Комбинации коэффициентов преломления для стекла и жидкости

Номер комбинации	Коэффициент преломления	
	Стекло	Жидкость
1	1.0	1.0
2	1.0	1.5
3	1.3	1.0
4	1.3	1.5

На рисунке 20 представлено изображение, полученное для комбинации №1 – входных данных, где коэффициенты преломления для стекла и для жидкости равны 1. Изображение имеет наименьшее искажение света, что соответствует общепринятому поведению для одинаковых материалов.

На рисунке 21 представлено изображение, полученное для комбинации №2 – входных данных, где коэффициенты преломления для стекла равен 1, а для жидкости – 1.5. Можно наблюдать более выраженное преломление света, что характерно для перехода из менее плотной среды в более плотную, как это происходит при переходе из воздуха в воду.

На рисунке 22 представлено изображение, полученное для комбинации №3 – входных данных, где коэффициенты преломления для стекла равен 1.3, а для жидкости – 1. Изображение демонстрирует несколько иное поведение света, с уменьшением преломления по сравнению с предыдущим случаем, что обусловлено меньшим коэффициентом преломления для стекла.

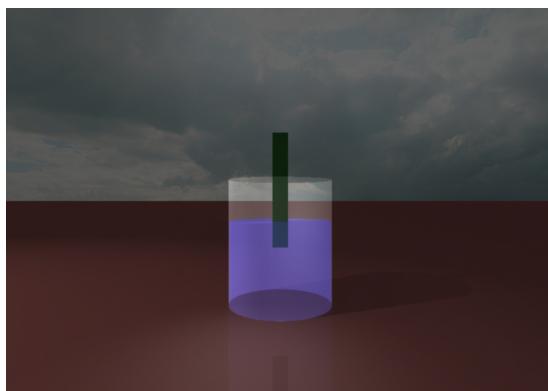


Рисунок 20 – Изображение, полученное для комбинации №1



Рисунок 21 – Изображение, полученное для комбинации №2

На рисунке 23 представлено изображение, полученное для комбинации №4 – входных данных, где коэффициенты преломления для стекла равен 1.3, а для жидкости – 1.5. Можно отметить комбинированный эффект увеличенного преломления как для стекла, так и для жидкости, что ведет к значительным изменениям в преломлении света при переходе через границу между двумя различными средами.



Рисунок 22 – Изображение, полученное для комбинации №3



Рисунок 23 – Изображение, полученное для комбинации №4

Вывод

Время построения изображения увеличивается с увеличением глубины рекурсии, что связано с увеличением сложности вычислений для каждого пикселя. Например, при глубине рекурсии 0 время выполнения составляет 2.7 секунды, а при глубине 8 – 24.8 секунды. Эти данные подтверждают, что более высокая глубина рекурсии увеличивает время построения изображения,

но также способствует более точной симуляции преломления и отражения света, улучшая реалистичность сцены.

Использование многозадачности с дополнительными потоками позволяет заметно сократить время построения изображения. Например, при использовании одного потока время выполнения составляет 5.1 секунды, а при использовании 128 потоков оно снижается до 1.08 секунды. Однако, после достижения определённого количества потоков (128 потоков) время выполнения начинает стабилизироваться, что свидетельствует о насыщении процессора и ограничениях на многозадачность. Это подтверждает, что для оптимизации времени рендеринга важна не только добавление потоков, но и балансировка их числа с учётом архитектуры процессора.

Изменение коэффициента преломления для материалов (стекла и жидкости) оказывает значительное влияние на визуализацию прозрачных объектов. В комбинации №1 (стекло и жидкость с коэффициентом преломления 1.0) изображение имеет наименьшие искажения света. В комбинациях с более высокими коэффициентами преломления, например, в комбинации №2 (где коэффициенты для стекла и жидкости составляют 1.0 и 1.5 соответственно), наблюдается более выраженное преломление света, что создаёт более реалистичное изображение при переходе через границу между двумя средами с различной плотностью. Такие изменения в коэффициенте преломления подтверждают важность точных расчётов для достижения визуальной правдоподобности при рендеринге прозрачных материалов.

ЗАКЛЮЧЕНИЕ

В ходе выполнения курсовой работы было разработано программное обеспечение для создания реалистичных изображений стержня, частично погруженного в прозрачный сосуд, наполненный жидкостью, с применением алгоритма обратной трассировки лучей. Описание моделируемых объектов, выбор методов представления объектов и моделей освещения позволили обеспечить точное и реалистичное отображение сцены. Было обосновано использование алгоритма обратной трассировки лучей как основного инструмента для вычисления взаимодействия света с материалами, что позволило добиться высокого уровня детализации и достоверности изображения.

Реализация программы позволила провести ряд исследований для оценки производительности в зависимости от глубины рекурсии и использования дополнительных потоков. Результаты показали, что увеличение глубины рекурсии улучшает реалистичность изображения, но сопровождается увеличением времени построения кадра. Использование многозадачности с дополнительными потоками позволяет существенно сократить время рендеринга, что подтверждает важность эффективного распределения вычислительных ресурсов.

Кроме того, было проведено исследование влияния коэффициента преломления на визуализацию прозрачных объектов, что продемонстрировало, как изменение оптических свойств материалов влияет на визуальное восприятие сцены.

В результате работы выполнены все поставленные задачи, и достигнута цель данной курсовой работы.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Peddie J. The History of Visual Magic in Computers: How Beautiful Images are Made in CAD, 3D, VR and AR. – New York: Springer, 2013. p. 101.
2. Беклемишев Д.В. Курс аналитической геометрии и линейной алгебры: учебное пособие для физико–математических и инженерно–физических специальностей вузов. – М.: Наука, 1984.
3. Lengyel E. Mathematics for 3D Game Programming and Computer Graphics. 3rd edition. – Boston, USA: Course Technology, Cengage Learning, 2011.
4. Vaughan W. Digital Modeling. – Berkeley, CA: New Riders, 2012.
5. Shapiro V. Solid Modeling. – Madison, USA: Elsevier, 2001.
6. Akenine-Möller T. Real-Time Rendering. Fourth edition edition. – Boca Raton, USA: Taylor & Francis, CRC Press, 2018.
7. Задорожный А.Г. Модели освещения и алгоритмы затенения в компьютерной графике: учебное пособие. – Новосибирск: НГТУ, 2020.
8. Phong B. T. Illumination for Computer Generated Pictures // Communications of the ACM. 1975. June. Vol. 18, no. 6. P. 311–317.
9. Документация ISO/IEC 14882:2020 [Электронный ресурс]. Режим доступа: <https://www.iso.org/standard/79358.html> (Дата обращения: 20.10.2024).
10. Документация среды разработки Visual Studio Code [Электронный ресурс]. Режим доступа: <https://code.visualstudio.com/docs> (Дата обращения: 20.10.2024).
11. Руководство пользователя macOS 14.6 [Электронный ресурс]. Режим доступа: https://developer.apple.com/documentation/macos-release-notes/macos-14_6-release-notes (Дата обращения: 15.11.2024).

12. Документация процессора Intel Core i5-10210U [Электронный ресурс].
Режим доступа: <https://ark.intel.com/content/www/us/en/ark/products/195436/intel-core-i510210u-processor-6m-cache-up-to-4-20-ghz.html> (Дата обращения: 17.10.2024).

ПРИЛОЖЕНИЕ А

Далее представлены слайды презентации.



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)

Построение реалистичного изображения стержня, частично погруженного в тонкостенный прозрачный сосуд, наполненный жидкостью

Дисциплина: «Компьютерная графика»

Студент: Цховребова Яна Роландовна, ИУ7–54Б

Научный руководитель: Терентьев Юрий Иванович

Москва, 2024 г.

Цель и задачи работы

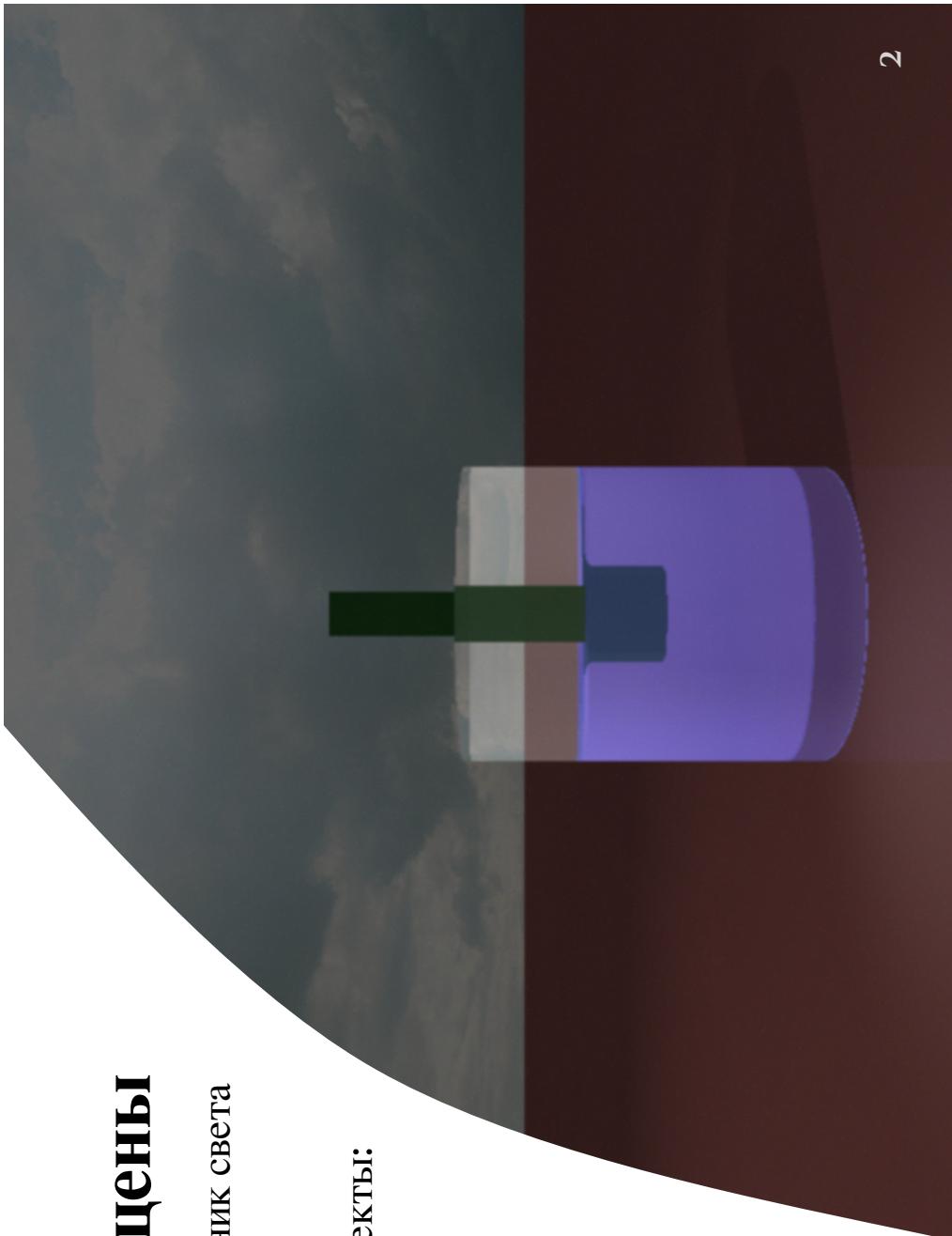
Цель работы – разработка программного обеспечения для построения реалистичного изображения стержня, частично погруженного в тонкостенный прозрачный сосуд, наполненный жидкостью.

Задачи

- Описать моделируемые объекты сцены.
- Проанализировать способы представления объектов и обосновать выбор наиболее подходящего.
- Проанализировать модели освещения и обосновать выбор наиболее подходящей.
- Проанализировать алгоритмы рендеринга и обосновать выбор наиболее подходящего.
- Разработать выбранные алгоритмы.
- Реализовать программное обеспечение для достижения цели данной работы.
- Исследовать время построения изображения в зависимости от глубины рекурсии и от количества используемых дополнительных потоков; реалистичность преломления света для прозрачных объектов для разработанного программного обеспечения.

Объекты сцены

- Точечный источник света
- Камера
- Трехмерные объекты:
 - плоскость
 - сосуд
 - жидкость
 - стержень



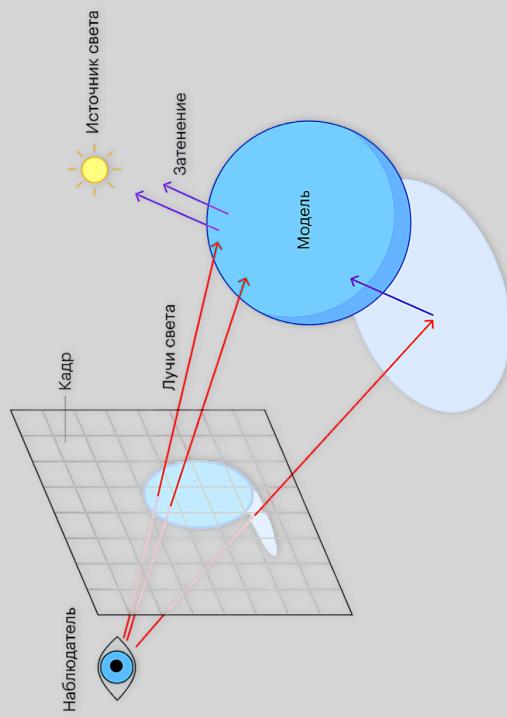
Твердотельная модель

- Геометрическая форма (задается аналитически)
- Физические характеристики:
 - прозрачность
 - отражаемость
 - коэффициент преломления

Выбор метода рендеринга сцены

Метод	Реалистичность изображения	Сложность реализации	Вычислительная нагрузка
«Бросание лучей»	Низкая	Низкая	Низкая
Обратная трассировка лучей	Средняя	Средняя	Средняя
Трассировка путей	Высокая	Высокая	Высокая

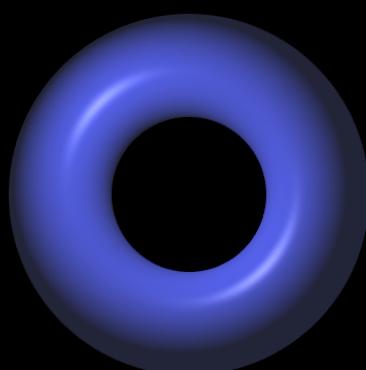
Выбор: метод обратной трассировки лучей



Простая модель освещения



Модель освещения Фонга

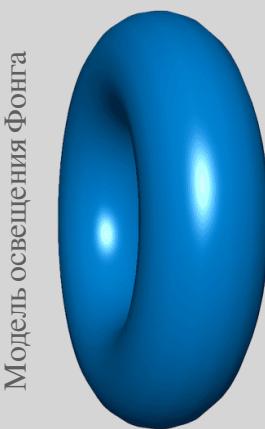


Выбор: модель освещения Фонга

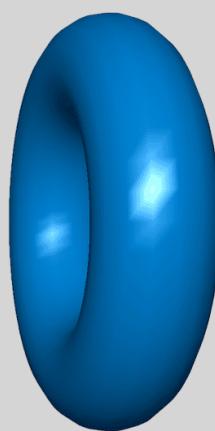
Выбор модели освещения

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

Модель освещения Фонга

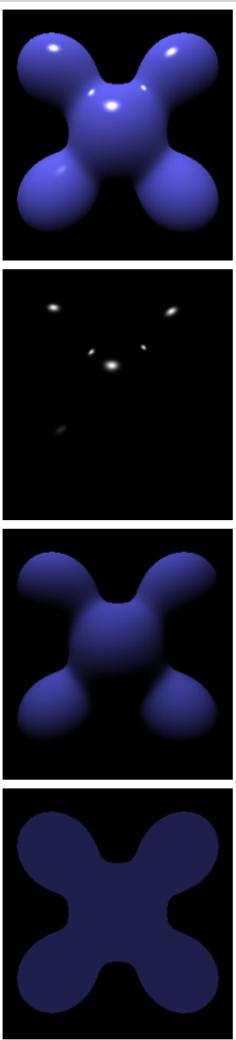


Модель освещения Гуро



Модель освещения Фонга

Включает



I — полная интенсивность освещения; I_L — интенсивность источника света

N — нормаль к поверхности в точек пересечения; L — направление от точки пересечения к источнику света

$R = 2(N \cdot L) \cdot N - L$ — вектор отражения; V — направление от точки пересечения к камере

k_a, k_d, k_s — коэффициента фонового, диффузного и зеркального освещения для материала

$$I_{amb} = k_a \cdot I_a \quad \text{интенсивность фонового освещения}$$

$$I_{dif} = k_d \cdot I_L \cdot \max(0, \mathbf{N} \cdot \mathbf{L}) \quad \text{интенсивность диффузного освещения}$$

$$I_{spc} = k_s \cdot I_L \cdot (\max(0, \mathbf{R} \cdot \mathbf{V}))^n \quad \text{интенсивность зеркального освещения}$$

$$I = I_{amb} + I_{dif} + I_{spc}$$

Пересечение луча с объектами

$$P(t) = S + t \cdot V - \text{уравнение прямой}$$

$P(t)$ – точка на прямой, t – параметр, V – вектор направления, S – начальная точка

Уравнение цилиндра

$$x^2 + m^2y^2 = r^2, 0 \leq z \leq h$$

r – радиус, h – высота, m – коэффициент масштабирования

$$(V_x^2 + m^2V_y^2)t^2 + 2(S_xV_x + m^2S_yV_y)t + S_x^2 + m^2S_y^2 - r^2 = 0$$

Решение этого уравнения дает значение параметра t , где луч пересекает бесконечный цилиндр. Полученные точки пересечения должны быть проверены по z-координатам, чтобы удовлетворять условию: $0 \leq z \leq h$

Уравнение параллелепипеда

$$x = 0, x = r_x,$$

$$y = 0, y = r_y,$$

$$z = 0, z = r_y$$

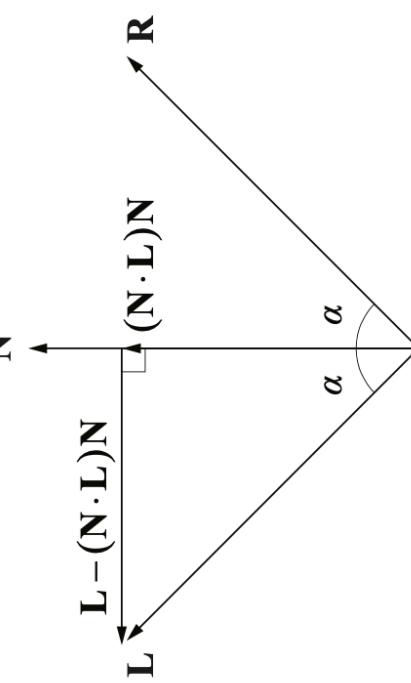
r_x, r_y, r_z – представляют размеры параллелограмма

$$t = \frac{r_x - S_x}{V_x} - \text{для точки пересечения } P(t) \text{ с}$$

плоскостью $x = r_x$ (аналогично для остальных)

Отражение луча

Угол падения равен углу отражения



$$R = 2(N \cdot L)N - L$$

- R — направление отраженного луча
- L — направление от точки пересечения к источнику света
- N — нормаль к поверхности

Закон Снеллиуса

$$\eta_L \sin(\theta_L) = \eta_T \sin(\theta_T)$$

- η_L, η_T — показатели преломления двух сред

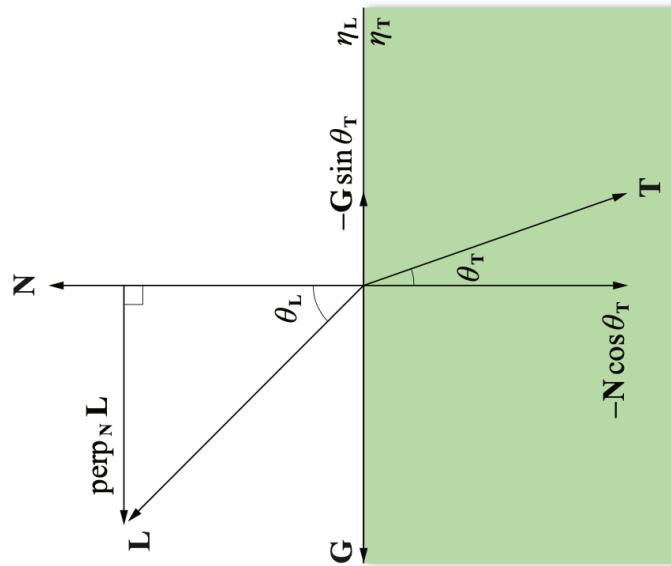
- θ_L — угол падения; θ_T — угол преломления

$$T = \left(\frac{\eta_L}{\eta_T} N \cdot L - \sqrt{1 - \frac{\eta_L^2}{\eta_T^2} [1 - (N \cdot L)^2]} \right) N - \frac{\eta_L}{\eta_T} L$$

- T — направление преломленного луча

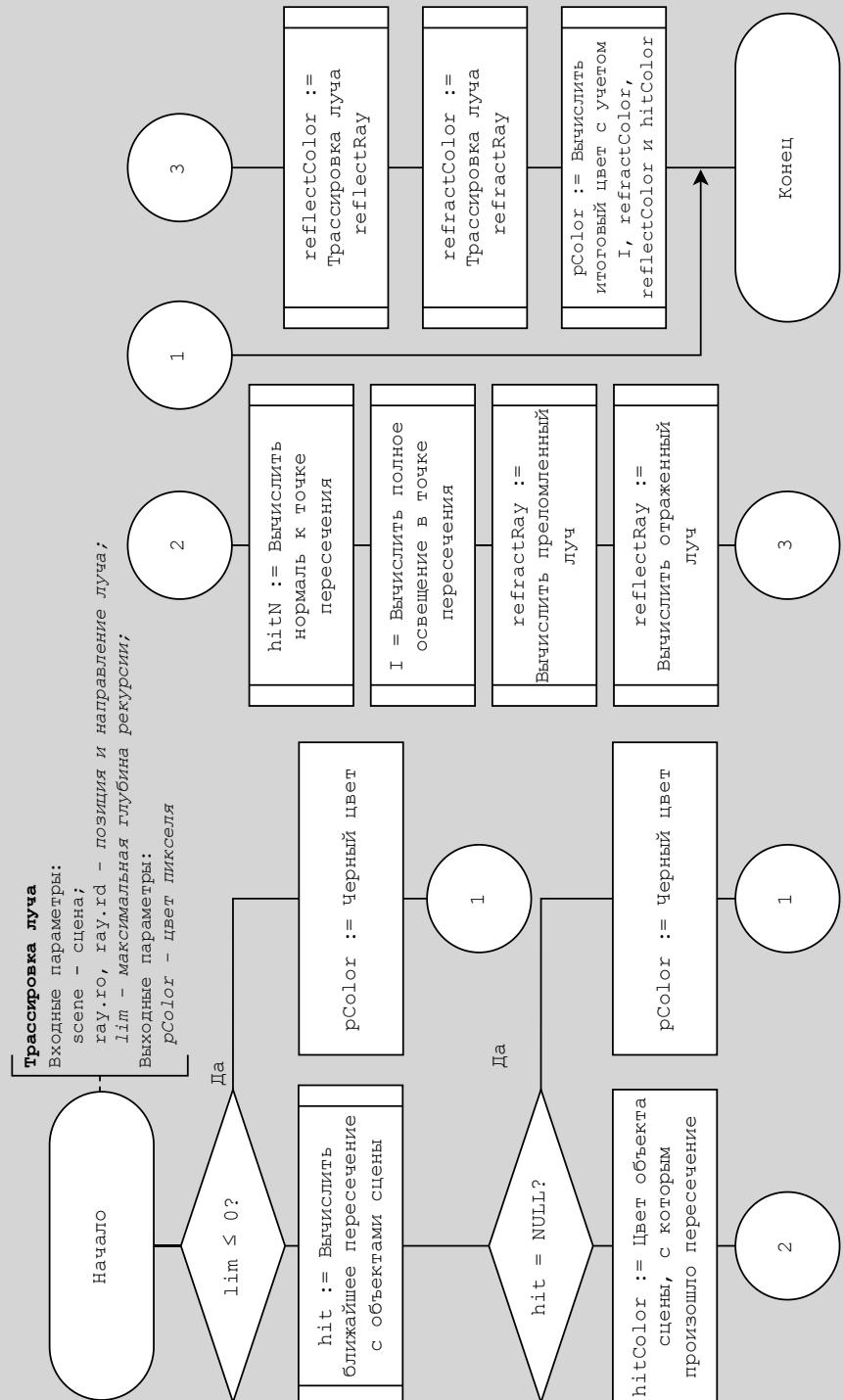
- L — направление от точки пересечения к источнику света
- N — нормаль к поверхности

Преломление луча

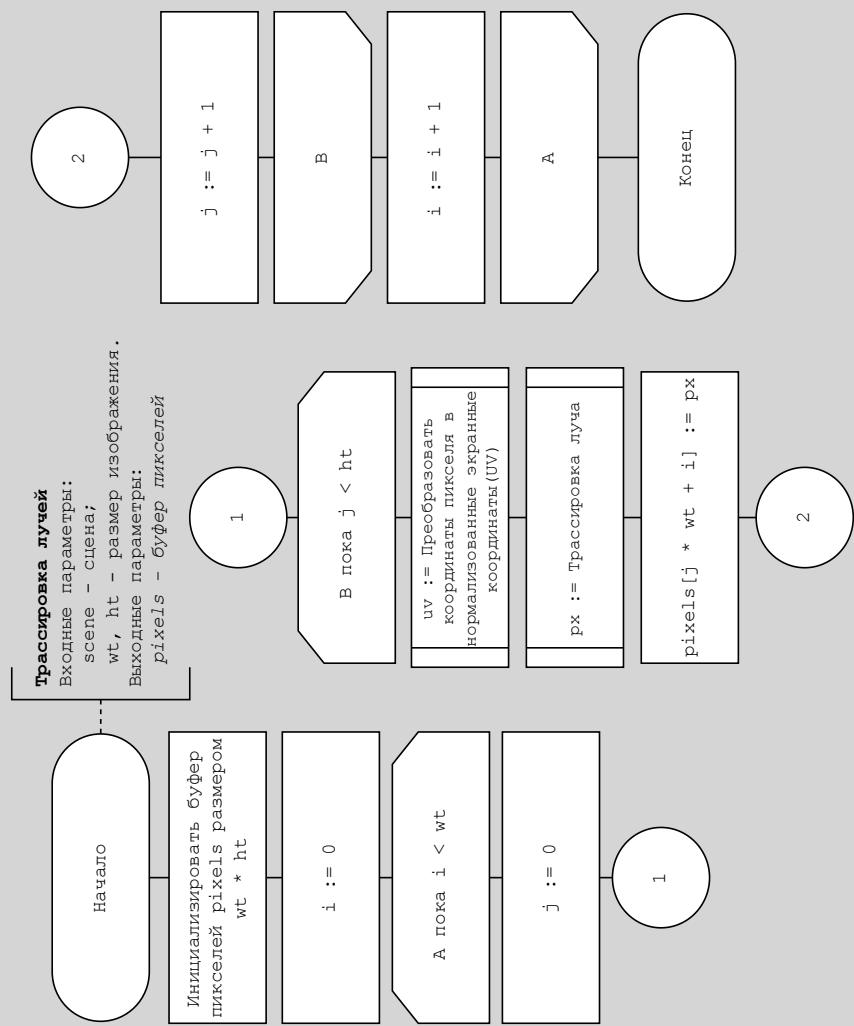


9

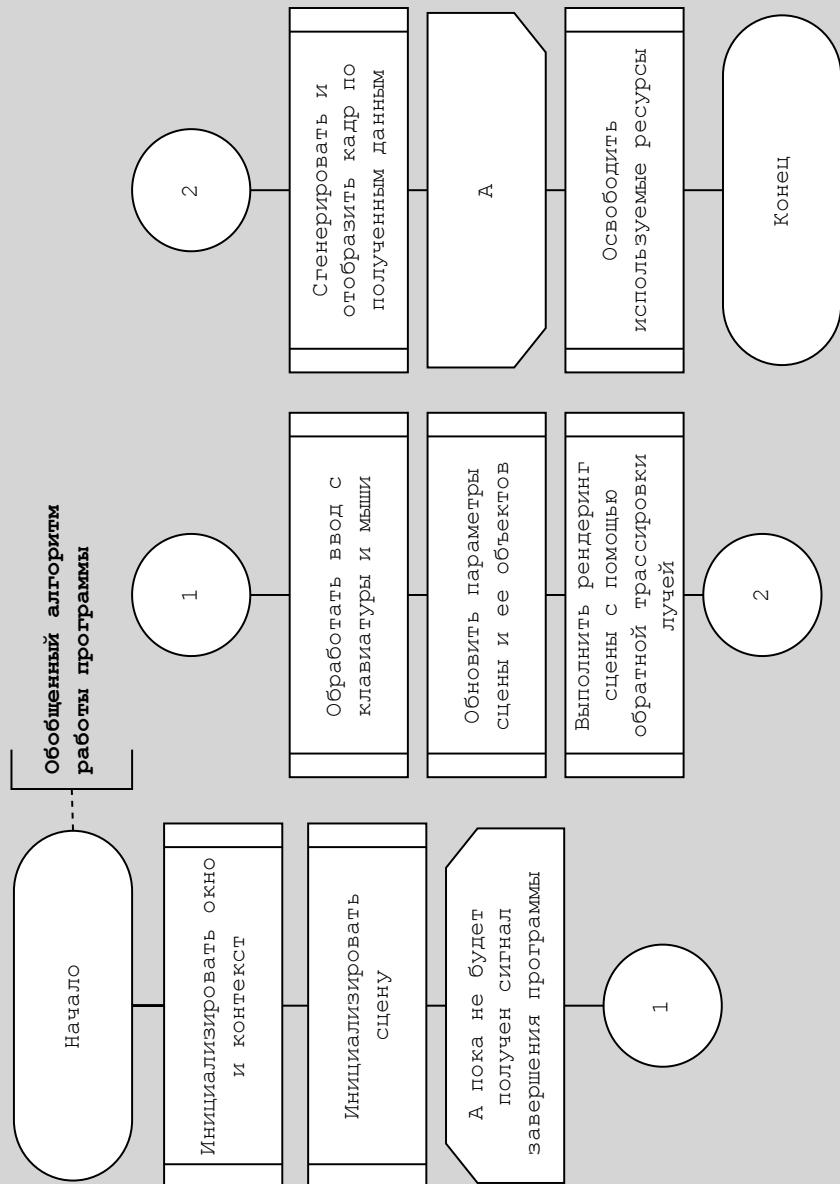
Алгоритм обратной трассировки лучей для одного пикселя



Алгоритм обратной трассировки лучей

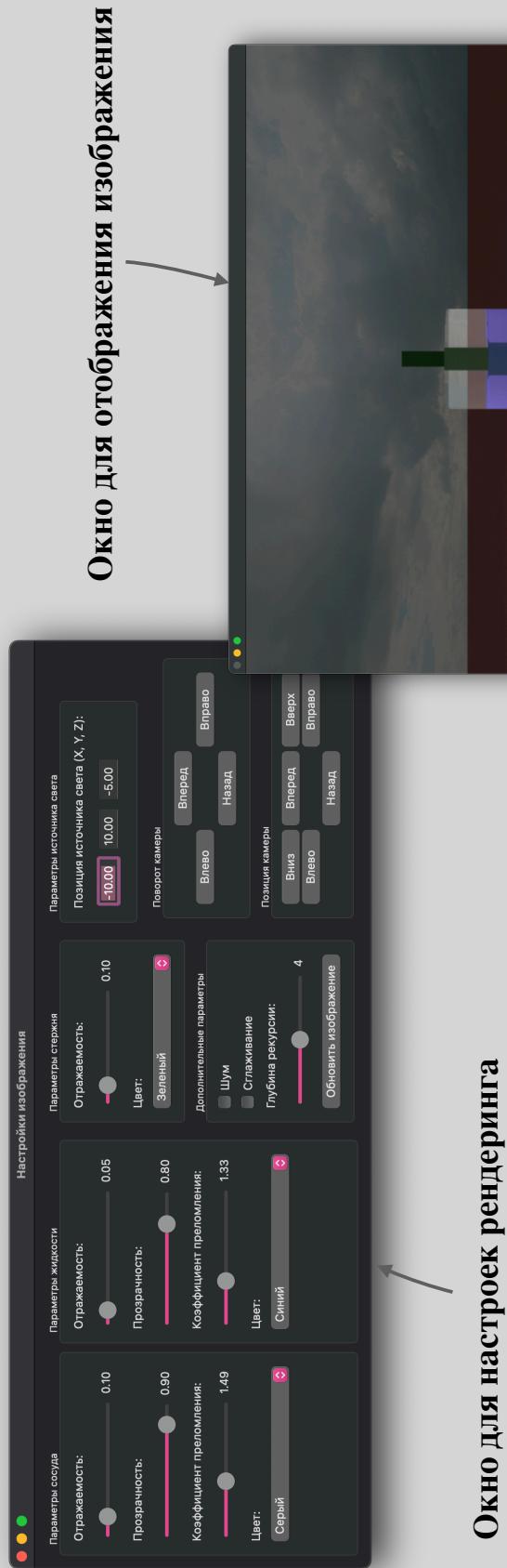


Алгоритм работы программы



12

Пользовательский интерфейс



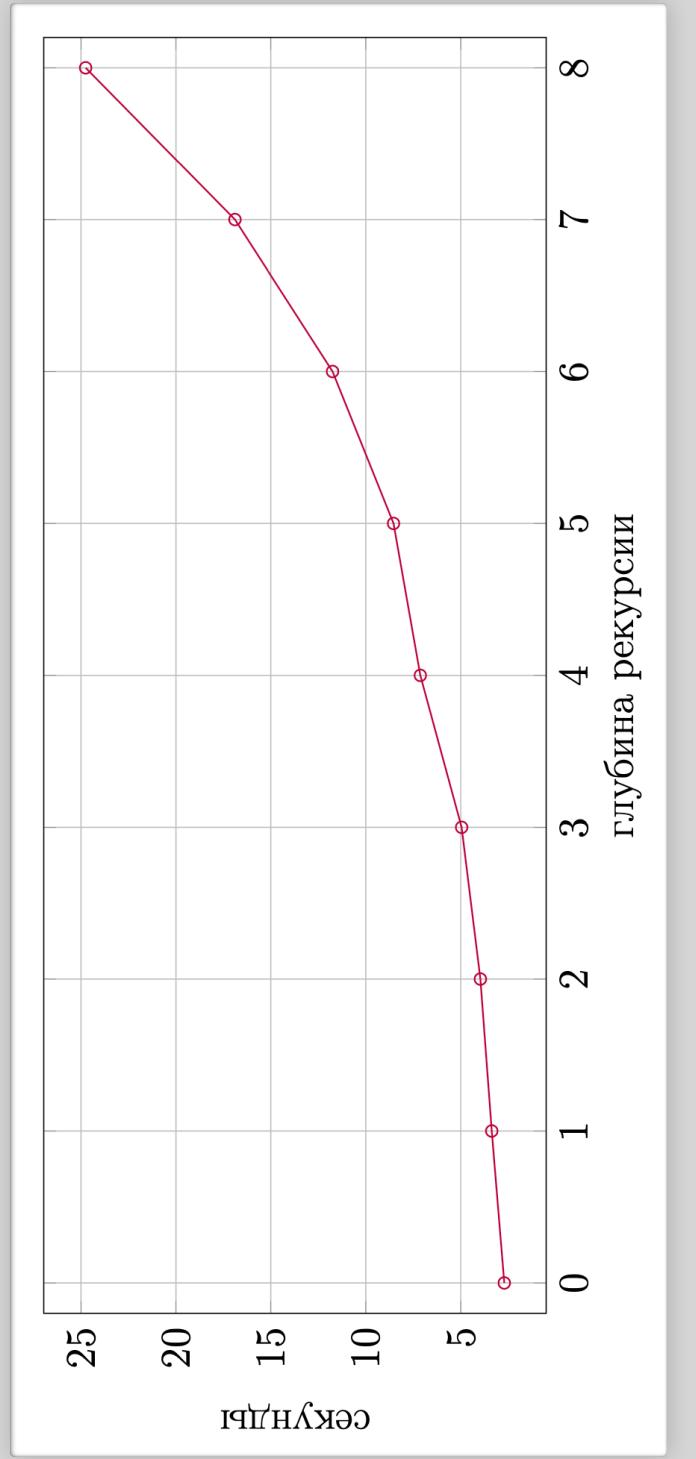
Окно для настроек рендеринга

13

Технические характеристики

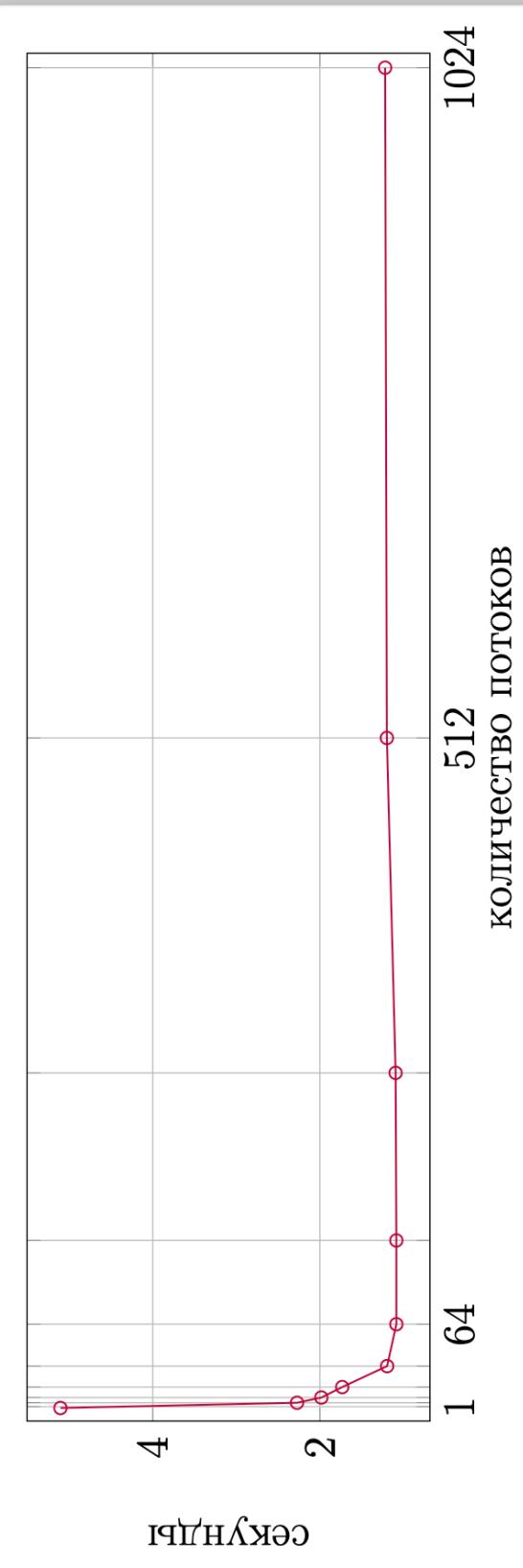
- Операционная система macOS 14.6.1
- Оперативная память 16 ГБ
- Процессор 2 ГГц 4-ядерный Intel Core i5 (8 логических ядер)

Зависимость времени построения изображения от глубины рекурсии



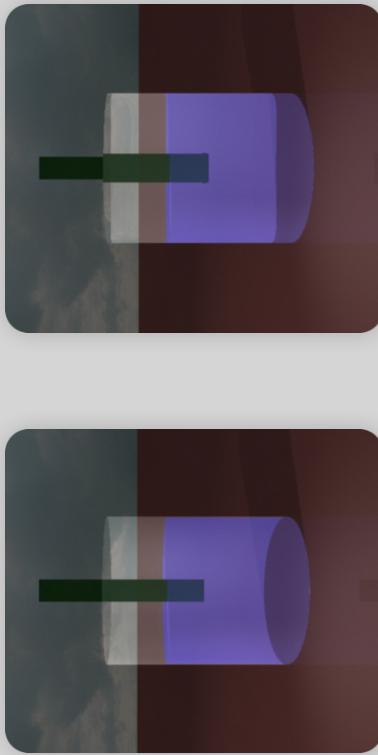
15

Зависимость времени построения изображения от количества используемых потоков



Реалистичность преломления лучей для прозрачных материалов объектов сцены

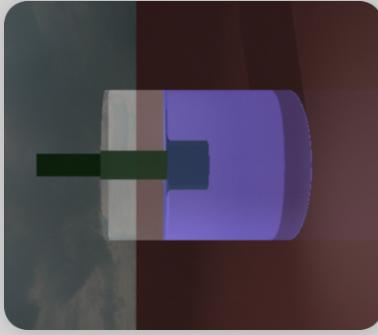
$$k_v = 1.0; k_l = 1.0$$



$$k_v = 1.0; k_l = 1.5$$



$$k_v = 1.3; k_l = 1.5$$



k_v — коэффициент преломления сосуда

k_l — коэффициент преломления жидкости

Заключение

В ходе выполнения работы:

- описаны моделируемые объекты сцены;
- проанализированы способы представления объектов и выбран наиболее подходящий;
- проанализированы алгоритмы рендеринга и выбран наиболее подходящий;
- проанализированы модели освещения и выбрана наиболее подходящая;
- разработаны выбранные алгоритмы;
- реализовано программное обеспечение для достижения цели данной работы;
- исследованы время построения изображения в зависимости от глубины рекурсии; время построения изображения от количества используемых дополнительных потоков; реалистичность преломления света для прозрачных объектов для разработанного программного обеспечения.