Содержание

Введение

Компьютерная графика — область деятельности, в которой компьютеры используются с целью создания (синтеза) и изменения графических изображений. В наши дни компьютерная графика применяется во всех областях жизни человека, что связано с широким распространением и стремительным ростом производительности вычислительных систем.

Особенный интерес представляют алгоритмы построения реалистичных изображений. Они способны учитывать множество физических явлений, таких как отражение, преломление, прозрачность, блеск и тень, но являются крайне требовательными к ресурсам компьютера. Их скорость работы напрямую зависит от требований к качеству и реалистичности синтезируемого изображения. Необходимо найти баланс между производительностью и реалистичностью получаемого изображения. Одним из самых распространённых алгоритмов построения реалистического изображения является алгоритм трассировки лучей.

Целью работы является анализ и реализация алгоритма построения реалистичного изображения с применением трассировки лучей и глобальной моделью освещения.

Для достижения поставленной цели необходимо решить следующие задачи:

— описать визуализируемую сцену;

— описать существующие алгоритмы построения реалистичных изображений, текстурирования, моделей освещения и способы представления объектов сцены и выбрать подходящие для построения реалистичного изображения;

— разработать выбранные алгоритмы и структуры данных;

— реализовать все алгоритмы в виде программы с графическим интерфейсом;

— провести исследование быстродействия разработанного ПО в зависимости от количества параллельно работающих потоков, выполняющих рендер изображения сцены.

1 Аналитический раздел

В этом разделе будет представлен анализ существующих способов представления объектов, алгоритмов построения реалистического изображения, текстурирования и моделей освещения.

1.1 Описание трёхмерного объекта

1.1.1 Геометрические примитивы

Примитив может быть описан некоторой функцией, принимающей параметры, например, центр и радиус для сферы или ширину, высоту и глубину для параллелепипеда. В качестве таких примитивов могут выступать любые геометрические объекты: куб, конус, пирамида, сфера или цилиндр и т. д. Достоинствами данного метода являются простота нахождения пресечения луча и модели и малое количество информации для хранения представления объекта. К недостаткам можно отнести сложность создания реалистичных моделей из геометрических примитивов и трудность наложения текстур.

1.1.2 Воксельная модель

Двумерные модели можно описать с помощью пикселей. Аналогично можно и описывать трёхмерные модели, используя воксели, представляющие собой маленькие кубики, из которых строится модель. Достоинством данного метода являются простота реализации алгоритма трассировки лучей с воксельными моделями. К недостаткам можно отнести большой расход памяти для хранения представления объекта и низкое разрешение моделей.

1.1.3 Полигональная модель

Полигональная модель использует полигональную сетку, которая является совокупностью связанных между собой выпуклых многоугольников

(полигонов), аппроксимирующих поверхность модели. Представление модели в виде многоугольников упрощает их отрисовку, или рендер. Чаще всего как полигон используются треугольник, так как он является простейшим многоугольником и все остальные многоугольники могут быть разбиты на треугольники. Данным способом можно описать объекты любой формы с хорошим разрешением и детализацией. Благодаря тому, что полигоны — это плоские многоугольники, их легко использовать для имитации неровностей путём изменения их нормали, а также для текстурирования посредством указания текстурных координат. Достоинствами данного метода являются необходимость вычислять только координаты вершин при преобразованиях и небольшой объём данных при некоторой аппроксимации поверхности. К недостаткам можно отнести сложность алгоритмов визуализации и погрешность при аппроксимации [1].

Вывод

После анализа вышеописанных вариантов в качестве основного способа представления объектов сцены была выбрана полигональная модель. Так как в отличии от других вариантов с помощью полигональной сетки можно представить объекты любой формы с хорошим разрешением и детализацией, используя относительно небольшой объём памяти для хранения. Сложность же алгоритмов отрисовки не играет большой роли в программе, поскольку в данной работе предполагается получение реалистичного изображения, и критерий сложности алгоритма отрисовки не является главным.

1.2 Алгоритм построения трёхмерного изображения

1.2.1 Алгоритм Z-буфера

Алгоритм работает в пространстве изображения и в нём используется 2 буфера: Z-буфер (буфер глубины) и буфер кадра, размеры которых соответ-

ствуют количеству пикселей на экране. В Z-буфере находится информация о координате z для каждого пикселя, а в буфере кадра — его интенсивность. В начале работы буферы заполняются минимальным значением координаты z и интенсивностью фона соответственно. Затем каждый многоугольник преобразуется в растровую форму и записывается в буфер кадра, при этом не производится никакого начального упорядочения.

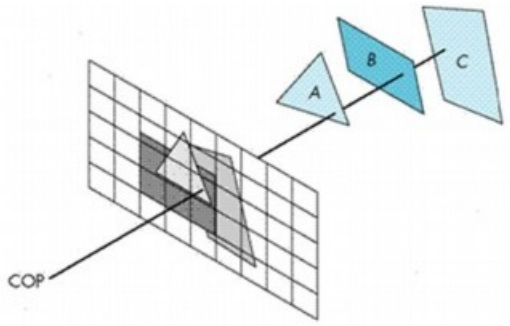

Рисунок 1.1 – Демонстрация пересечения лучом объектов сцены по алгоритму Z-буфера

В процессе работы глубина каждого нового пикселя сравнивается с глубиной, занесённой в буфер глубины. Если глубина нового пикселя меньше, то в буфер кадра заносится данные интенсивности нового пикселя, а в z-буфер — новую координату z. Иначе данные в буферах не меняются.

Достоинствами данного алгоритма является простота его реализации и отсутствие сортировки элементов сцены. К недостаткам же можно отнести большой объём используемой памяти и трудоемкость реализации эффектов прозрачности и преломления, а также устранения ступенчатости [2].

1.2.2 Алгоритм обратной трассировки лучей

Идея алгоритма обратной трассировки лучей основана на отслеживании взаимодействия отдельных лучей с объектами. Алгоритм используется для создания реалистичного освещения, отражений и теней, обеспечивающее более высокий уровень реализма по сравнению с традиционными способами рендеринга.

Из камеры выпускается луч в каждый пиксель экрана и отслеживается, куда он попадёт. Если луч пересекает объект, то рассчитывается освещение в

этой точке в соответствии с моделью освещения. Интенсивность света в точке состоит из фоновой, диффузной и зеркальной составляющей. Последние два зависят от интенсивности и положения источника света. Итоговая интенсивность получается из суммы интенсивностей от всех видимых из точки пересечения источников света.

Также учитывается тень. Для этого из точки пресечения испускается теневой луч в сторону источника. Если находится пресечение с объектом между началом луча и источником света, то переходим к следующему источнику, иначе вычисляем интенсивность для этого источника в точке пересечения.

Если объект обладает отражающими свойствами, то вычисляется и испускается отражённый луч. Аналогично алгоритм работает с преломлённым лучом.

Достоинствами данного алгоритма являются высокая реалистичность получаемого изображения и учёт таких физических явлений, как тень, преломление, отражение. Также алгоритм легко поддаётся распараллеливанию. Основным недостатком является производительность. Каждый раз необходимо просчитывать множество новых лучей, что создаёт немалую нагрузку на вычислительные устройства [3].

Вывод

Для создания реалистичного изображения лучше всего подходит алгоритм трассировки лучей, так как он даёт наиболее приближенный к реальности результат и учитывает отражения, прозрачность и тени.

1.3 Модель освещения трёхмерных объектов

1.3.1 Модель Ламберта

Простейшая модель освещения: она учитывает только диффузное освещение. Считается, что свет падающий в точку, одинакового рассеивается по

всем направлением полупространства. Таким образом, освещенность в точке определяется только плотностью света в точке поверхности, а она линейно зависит от косинуса угла падения и считается по формуле

$$I = k_d \cdot (\vec{n}, \vec{l}), \tag{1.1}$$

где приняты следующие обозначения:

— k_d — коэффициент диффузного отражения;

— $\vec{n}$ — нормаль в точке пресечения;

— $\vec{l}$ — единичный вектор, направленный к источнику света.

1.3.2 Модель Фонга

Модель расчёта освещения трёхмерных объектов, в том числе полигональных моделей и примитивов, а также метод интерполяции освещения по всему объекту. Это локальная модель освещения, то есть она учитывает только свойства заданной точки и источников освещения, игнорируя эффекты рассеивания, линзирования, отражения от соседних тел. Эта модель состоит из диффузной составляющей и зеркальной и рассчитывается по формуле (1.2). Благодаря зеркальной составляющей на объектах появляются блики. Интенсивность в точке зависит от того, насколько близок отражённый вектор к вектору, направленному из точки падения в сторону наблюдателя. В модели учитывается интенсивность фонового, диффузного и зеркального освещения. Для расчёта диффузной составляющей используется модель Ламберта [4]. Освещение рассчитывается по формуле

$$I = k_a \cdot I_a + k_d \cdot (\vec{n}, \vec{l}) + k_s \cdot (\vec{v}, \vec{r})^s, \tag{1.2}$$

где приняты следующие обозначения:

— k_d — коэффициент диффузного отражения;

— k_s — коэффициент зеркального отражения;

— k_a — коэффициент рассеянного отражения;

— I_a — интенсивность фонового освещения;

— $\vec{n}$ — нормаль в точке пересечения;

— $\vec{l}$ — единичные вектор, направленный к источнику света;

— $\vec{v}$ — единичные вектор, направленный к наблюдателю;

— $\vec{r}$ — единичные вектор, отражение $\vec{l}$;

— s — степень, аппроксимирующая пространственное распределение зеркально отраженного света.

1.3.3 Модель Уиттеда

Модель освещенности Уиттеда является одной из самых распространенных и наиболее часто используемой моделью в методе трассировки лучей. Использует для расчёта интенсивности глобальную модель освещения, учитывающую свет провзаимодействовавший с другими объектами. Помимо учёта фоновой, диффузной и зеркальной компонент в этой модели ещё учитывается интенсивность отражённого и преломлённого света от других тел [5]. Освещение рассчитывается по формуле

$$I = k_a \cdot I_a \cdot C + k_d \cdot I_d \cdot C + k_s \cdot I_s + k_r \cdot I_r + k_t \cdot I_t, \tag{1.3}$$

где приняты следующие обозначения:

— k_a — коэффициент фоновой подсветки;

— k_d — коэффициент диффузного рассеивания;

— k_s — коэффициент зеркальности;

— k_r — коэффициент отражения;

— k_t — коэффициент прозрачности;

— I_a — интенсивность фоновой подсветки;

— I_d — интенсивность, учитываемая для диффузного рассеивания;

— I_s — интенсивность, учитываемая для зеркальности;

— I_r — интенсивность излучения, приходящего по отраженному лучу;

— I_t — интенсивность излучения, приходящего по преломленному лучу;

— C — цвет исходного объекта.

Для расчёта локальной интенсивности т. е. фоновой, диффузной и зеркальной составляющей используется модель Фонга.

Вывод

Для получения наиболее реалистических изображений была выбрана модель Уиттеда как самая подходящая и реалистичная, так как она учитывает такие эффекты, как отражение, прозрачность, преломление, тень.

2 Конструкторская часть

В этом разделе представлено описание используемых типов данных, подробный разбор алгоритмов и их схемы.

2.1 Описание используемых типов данных

Используются следующие типы данных:

— Point — точка;

— Vector — вектор;

— Color — цвет;

— Polygon — полигон, в нём хранятся вершины;

— SceneObject — объект сцены;

— Sphere — шар;

— Light — точечный источник света (цвет и координаты источника света);

— Camera — камера (координаты, направление и вектор, указывающий наверх);

— Ray — луч, в нём хранятся его начало и направление;

— Scene — сцена, хранящая все объекты.

2.2 Разработка алгоритма обратной трассировки лучей

2.2.1 Алгоритм обратной трассировки лучей

На рисунке 2.1 представлена схема алгоритма обратной трассировки лучей.

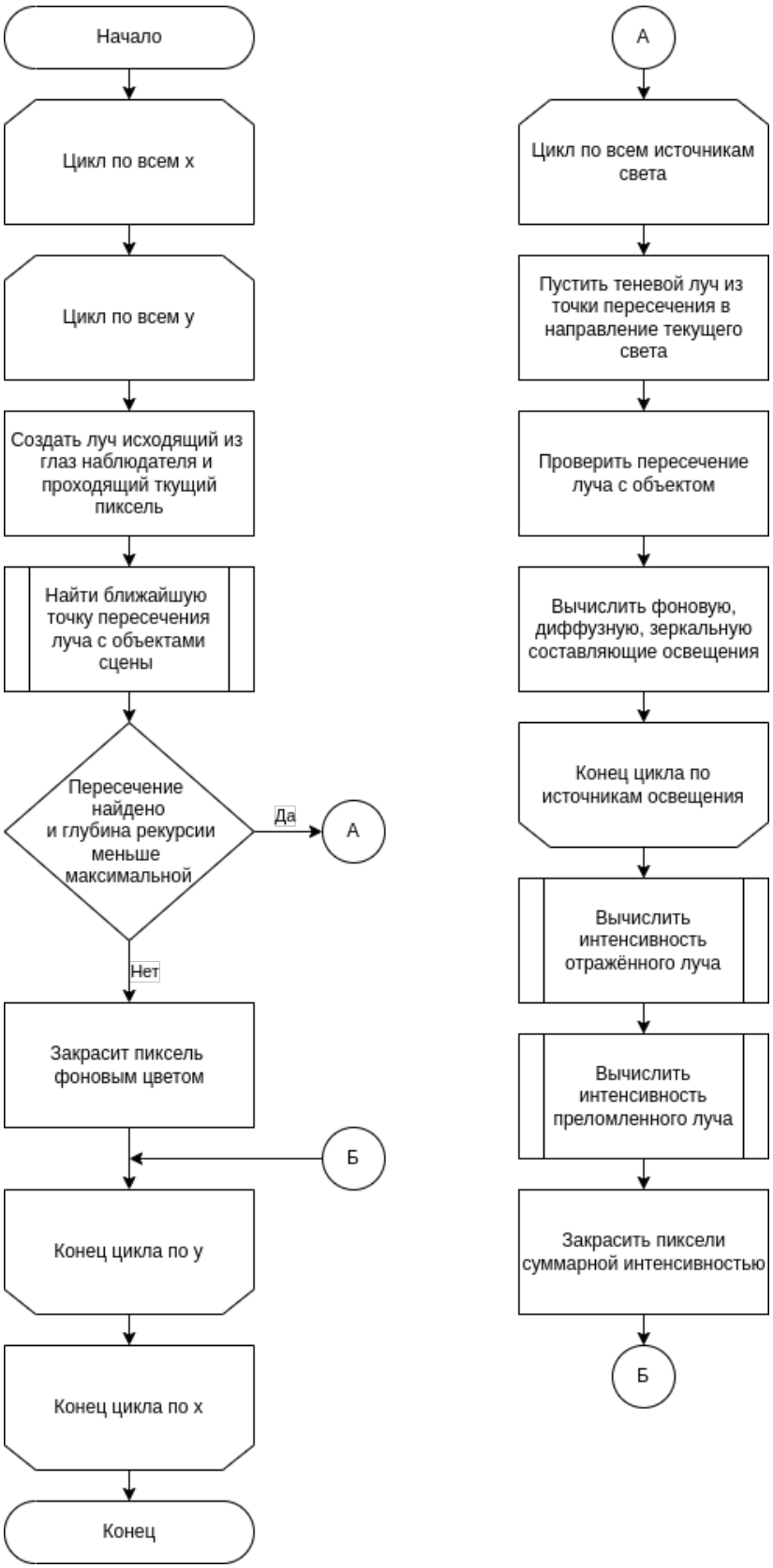

Рисунок 2.1 – Схема алгоритма обратной трассировки лучей

2.2.2 Нахождение пересечения луча с полигоном

Для поиска пересечения луча с треугольником используется алгоритм Моллера — Трумбора. Пример треугольника и луча приведён на рис. 2.2.

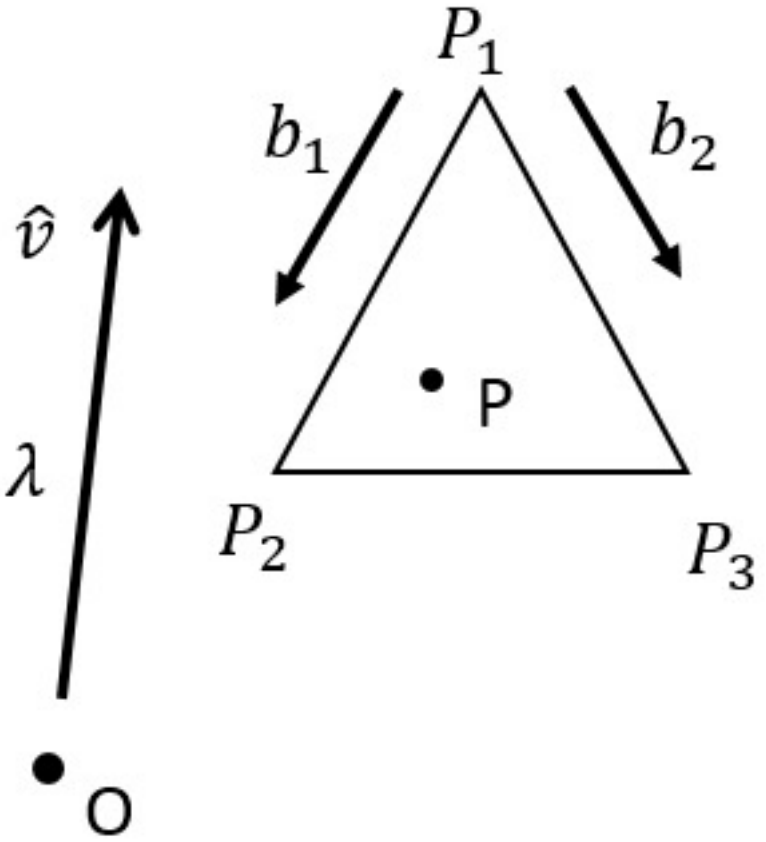

Рисунок 2.2 – Пример треугольника и луча

Введём следующие обозначения, соответствующие рисунку 2.2

— P — точка пересечения;

— O — начало луча;

— λ — расстояние от O до P;

— $\vec{v}$ — направление луча;

— P_0, P_1, P_2 — вершины треугольника;

— u, v — барицентрические координаты.

— $b_1 = P_2 - P_2$ — барицентрические координаты.

Барицентрические координаты представляют собой отношения площадей маленьких треугольников к большому треугольнику. Имея 3 точки на плоскости, можно выразить любую другую точку через её барицентрических координаты по формуле (2.1) из определения барицентрических координат. Если каждая из этих координат будет больше или равна нулю и сумма $u + v$ будет меньше 1, то искомая точка принадлежит треугольнику.

$$P(u, v) = (1 - u - v) \cdot P_1 + u \cdot P_2 + v \cdot P_3. \tag{2.1}$$

Выразим точку пересечения через параметрическое уравнения луча

$$P(t) = O + \lambda \cdot \vec{v}. \tag{2.2}$$

Приравняв правую часть уравнений (2.1) и (2.2), получим:

$$P(t) = O + \lambda \cdot \vec{v} = (1 - u - v) \cdot P_1 + u \cdot P_2 + v \cdot P_3. \tag{2.3}$$

Уравнений 2.4 по сути является системой из 3 уравнений с тремя неизвестными u, v, λ.

Проведя алгебраические преобразования, получим

$$\begin{bmatrix} \lambda \\ u \\ v \end{bmatrix} = \frac{1}{(\vec{D}, \vec{b_1})} \cdot \begin{bmatrix} (\vec{Q}, \vec{b_2}) \\ (\vec{D}, \vec{T}) \\ (\vec{Q}, \vec{v}) \end{bmatrix}. \tag{2.4}$$

где приняты следующие обозначения:

– $\vec{b_1} = P_2 - P_1$;

– $\vec{b_2} = P_3 - P_1$;

– $\vec{T} = P - P_1$;

– $\vec{D} = (\vec{v} \times \vec{d_2})$;

– $\vec{Q} = (\vec{T} \times \vec{d_1})$.

2.2.3 Нахождение пересечения с объемлющей оболочкой

При трассировке лучей крайне неэффективно искать пересечения с каждым полигоном объектов каждый раз. Лучше поместить объект в объемлющую оболочку и сначала проверять пересечение с ней. Если луч не пересекает оболочку, то он не пересекает объект сцены, находящийся в оболочке, и его можно сразу отбросить. В качестве такой оболочки будет использоваться сфера в связи с простотой поиска пересечения луча и сферы.

Из параметрического уравнение луча имеем

$$X = A + t \cdot \vec{d}. \tag{2.5}$$

Уравнение для точки на поверхности сферы выглядит следующим образом:

$$|X - C|^2 = r^2 \tag{2.6}$$

Подставив X во второе уравнение получаем:

$$|A + t \cdot \vec{d} - C|^2 = r^2 \tag{2.7}$$

Обозначим $\vec{s} = A - C$, тогда

$$|A + t \cdot \vec{d} - C|^2 = r^2 \tag{2.8}$$

Получилось квадратное уравнение относительно t. Дискриминант считается считается следующим образом:

$$D = 4 \cdot ((\vec{s}, \vec{d})^2 - d^2 \cdot (s^2 - r^2)) \tag{2.9}$$

Если D < 0, то объект, находящийся в объемлющей сфере, сразу можно исключать из рассмотрения, так как луч его точно не пересекает.

2.2.4 Оптимизация алгоритма трассировки лучей

Распараллеливание алгоритмов часто используют для ускорения работы их реализаций. Алгоритм трассировки лучей отлично поддаётся распараллеливанию, поскольку каждый пиксель экрана обрабатывается независимо. Можно разбить экран на сегменты, или сектора, в виде прямоугольников, которые будут обрабатываться параллельно, независимо друг от друга.

В программе будут использованы вспомогательные потоки, выполняющие рендер изображения сцены, и запускающий их главный поток. Последний после запуска вспомогательных потоков должен будет дождаться их завершения.

Также можно строить иерахическую структуру оболочек, что позволит отбрасывать сразу целые группы объектов, которые не пересекает данный луч. Это позволит снизить трудоёмкость алгоритма.

2.3 Реализация модели освещения Уиттеда

2.3.1 Нахождение отражённого луча

Для нахождения направление отражённого луча $\vec{R}$ необходимо знать только направления нормали $\vec{N}$ и падающего луча $\vec{L}$. Падающий вектор $\vec{L}$ можно разложить на два проекции $\vec{L_N}$ и $\vec{L_P}$, как на рисунке 2.3.

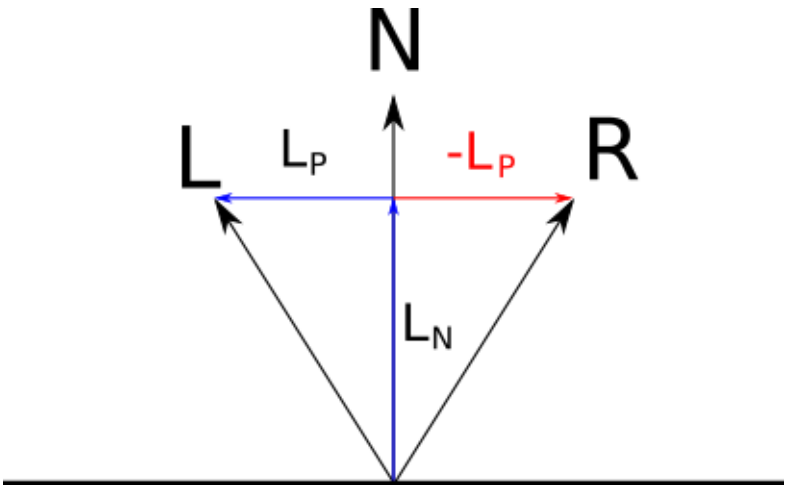

Рисунок 2.3 – Разложение падающего луча

Тогда

$$\vec{L} = \vec{L_N} + \vec{L_P}. \tag{2.10}$$

Так как $\vec{N}$ — единичный вектор, то длинна проекции будет равна $(\vec{L}, \vec{N})$ и

$$\vec{L_N} = (\vec{L}, \vec{N}) \cdot \vec{N}. \tag{2.11}$$

С учётом того что

$$L_P = \vec{L} - \vec{L_N}, \tag{2.12}$$

отражённый луч можно выразить как

$$\vec{R} = \vec{L_N} - \vec{L_P} = 2 \cdot (\vec{L}, \vec{N}) \cdot \vec{N} - \vec{L}. \tag{2.13}$$

2.3.2 Нахождение преломлённого луча

Преломлённый луч $\vec{P}$ можно найти исходя из того факта, что падающий и преломлённый лучи лежат в одной плоскости и из закона Снелиуса, который записывается так:

$$sin(\alpha) \cdot n_1 = sin(\gamma) \cdot n_2. \tag{2.14}$$

где приняты следующие обозначения:

— α — угол между падающим лучом и нормалью в точке пересечения лучом объекта (точке падения);

— γ — угол между преломлённым лучом и нормалью;

— n_1 — показатель преломления среды, из которой свет попадает;

— n_2 — показатель преломления среды, в которую свет попадает.

Введём дополнительные обозначения $n = \frac{n_1}{n_2}$, $\vec{L}$ — падающий луч, $\vec{N}$ — нормаль. Можно получить уравнение для вектора преломлённого луча:

$$\vec{P} = n \cdot (\vec{L} + cos(\alpha) \cdot \vec{N}) - \vec{N} \cdot \sqrt{1 - sin^2(\gamma)} \tag{2.15}$$

Если подкоренное выражение отрицательно, то этот случай соответствует полному отражению.

2.3.3 Общая реализация модели освещения Уиттеда

Раскрыв зеркальную и диффузную составляющие в формуле (1.3) согласно модели Фонга и просуммировав их по всем источникам света, получим окончательную формулу для расчёта интенсивности:

$$\begin{aligned} I = k_a \cdot I_a \cdot C + k_d \cdot \sum I_i \cdot (\vec{n}, \vec{l_i}) \cdot C + \\ + k_s \cdot \sum I_i \cdot (\vec{v}, \vec{r_i})^s + k_r \cdot I_r + k_t \cdot I_t. \end{aligned} \tag{2.16}$$

Вывод

В данном разделе были рассмотрены используемые типы данных, алгоритмы обратной трассировки лучей, пересечения с объектами сцены, поиск отражённого и преломлённого луча и расчёт интенсивности согласно модели Уиттеда.

3 Технологическая часть

В данном разделе рассмотрены средства реализации, описана структура программы, представлены листинги реализации алгоритма трассировки лучей и продемонстрирован интерфейс программы.

3.1 Средства реализации

В данной работе для реализации был выбран язык программирования $C++$ ввиду следующих причин:

— наличие большого количества библиотек для разработки приложений;

— возможность создания многопоточных приложений;

— наличие опыта программирования на данном языке.

В качестве среды разработки (IDE) была выбрана среда Clion, поскольку она содержит всю функциональность для удобной разработки ПО и в ней есть поддержка фреймворка Qt.

3.2 Сведения о модулях программы

Программа состоит из следующих модулей:

— Command — модуль, реализующий паттерн «команда» для работы со сценой;

— Drawer — модуль, хранящий «холст» и предоставляющий методы для рисования на нём;

— Error — модуль с классами для обработки исключительных ситуаций;

— Manager — модуль, хранящий сцену и предоставляет методы для работы с ней;

— Vec — модуль, реализующий математические вектора;

— Matrix — модуль, реализующий матрицы;

— Transformation — модуль предоставляющий функции для получения матриц преобразования;

— GeometryPrimitives — модуль, реализующий лучи, вершины, полигоны;

— Textures — модуль, реализующий текстуры;

— Properies — модуль с классами, хранящими дополнительную информацию для рендера;

— Loader — модуль, предоставляющий загрузку моделей из файлов;

— Model — модуль, реализующий объекты сцены;

— Light — модуль, реализующий источник освещения;

— Camera — модуль, реализующий камеру;

— Render — модуль, реализующий алгоритм генерации изображения;

— Scene — модуль, хранящий объекты сцены и предоставляющий методы для работы с ней;

— MainWindow — модуль окна приложения;

— main.cpp — файл, содержащий весь служебный код;

3.3 Реализация алгоритмов

В листингах 3.1–3.7 представлена реализация алгоритма трассировки лучей. Для функции трассировки лучей указаны аргументы min_x и max_x, которые обозначают границу блока, который обрабатывает и отрисовывает данная функция. При многопоточном исполнении эти аргументы будут обозначать область ответственности каждого вспомогательного потока.

Листинг 3.1 – Трассировка лучей

```
1  void RayTracingRendered::render(shared_ptr<Scene> scene, int min_x,
       int max_x)
2  {
3      int height = props.s_height, width = props.s_width;
4      double aspect_ratio = double(width) / height;
5      double viewport_height = props.v_height;
6      double viewport_width = aspect_ratio * viewport_height;
7      double focal_length = props.f_len;
8      Point3d origin = scene->cam()->origin;
9      Vec3d right = scene->cam()->dir ^ scene->cam()->up;
10     Vec3d dir_f = scene->cam()->dir * focal_length,
11     v_2 = scene->cam()->up * (viewport_height / 2.0),
12     h_2 = right * (viewport_width / 2.0);
13     Point3d lb_corner = origin + dir_f - v_2 - h_2,
14     rb_corner = origin + dir_f - v_2 + h_2,
15     lt_corner = origin + dir_f + v_2 - h_2;
16     Vec3d horizontal = rb_corner - lb_corner, vertical = lt_corner
         - lb_corner;
17     Color color{0, 0, 0};
18     double du = 1.0 / (width - 1), dv = 1.0 / (height - 1);
19     double min_u = double(min_x) / (width - 1), cu = min_u, cv = dv;
20     for (int j = 1; j < height; j++){
21         for (int i = min_x; i < max_x; i++){
22             color = {0, 0, 0};
23             Vec3d dir = lb_corner + cu * horizontal + cv * vertical
                 - origin;
24             dir.norm();
25             Ray r(origin, dir);
26
27             if (emitRay(scene, r, color, props.max_depth))
28                 drawer->draw_pixel(i, height - j, color);
29
30             cu += du;
31         }
32         cu = min_u;
33         cv += dv;
34     }
35 }
```

Листинг 3.2 – Расчёт освещения для луча (начало)

```
1  bool RayTracingRendered::emitRay(const shared_ptr<Scene> &scene,
       const Ray &r, Color &color, int depth)
2  {
3      IntersectionData data;
4
5      if (!closest_intersection(scene, r, data))
6      return false;
7      Color ambient, diffuse, specular, reflect, refract;
8      color = {0, 0, 0};
9
10     ambient = props.ambient * data.color;
11
12
13     for (auto it = scene->LightsBegin(); it != scene->LightsEnd();
           it++)
14     {
15         Vec3d l = (*it)->origin - data.p;
16
17         if (closest_intersection(scene, {data.p, l}, data, EPS, 1))
18         continue;
19         l.norm();
20
21         double n_dot_l = data.n & l;
22         if (n_dot_l >= 0)
23         {
24             Vec3d refl = reflectedRay(-l, data.n);
25             refl.norm();
26
27             if ((*data.iter)->props().diffuse > 0)
28             diffuse += (*it)->comp_color * n_dot_l *
                   (*data.iter)->props().diffuse;
29             if ((*data.iter)->props().specular > 0)
30             specular += (*it)->comp_color * mirror_reflection(refl,
                   -r.direction, (*data.iter)->props().shine) *
31             (*data.iter)->props().specular;
32         }
33     }
34
35     diffuse.member_mult(data.color);
```

Листинг 3.3 – Расчёт освещения для луча (окончание)

```
1
2       if (depth <= 0)
3       return true;
4
5       if ((*data.iter)->props().reflective > 0)
6       {
7           Ray refl_ray(data.p, reflectedRay(r.direction, data.n));
8           Color refl_color;
9           if (emitRay(scene, refl_ray, refl_color, depth - 1))
10          reflect = (*data.iter)->props().reflective * refl_color;
11      }
12
13      if ((*data.iter)->props().refraction > 0)
14      {
15          Ray refr_ray(data.p, refractedRay(r.direction, data.n,
                props.mi_world / data.iter->get()->props().mi));
16          Color refr_color;
17          if (emitRay(scene, refr_ray, refr_color, depth - 1))
18          refract = (*data.iter)->props().refraction * refr_color;
19      }
20
21      color = ambient + diffuse + specular + reflect + refract;
22
23      return true;
24  }
```

Листинг 3.4 – Распараллеливание трассировки лучей

```
1
2   void RenderCommand::execute(shared_ptr<Scene> scene)
3   {
4       std::vector<std::thread> threads;
5       int w_step = renderer->width() / thread_number, cur_x = 0;
6       for (int i = 0; i < thread_number; i++){
7           threads.emplace_back(run_thread, renderer, scene, cur_x,
                cur_x + w_step);
8           cur_x += w_step;
9       }
10      for (auto &th: threads)th.join();
11  }
```

Листинг 3.5 – Поиск ближайшего пересечения луча с объктом сцены

```
1  bool
2  RayTracingRendered::closest_intersection(const shared_ptr<Scene>
       &scene, const Ray &r, IntersectionData &data,
3  double t_min,
4  double t_max)
5  {
6      IntersectionData cur_data;
7      data.t = t_max;
8      bool flag = false;
9
10     for (auto it = scene->ModelsBegin(); it != scene->ModelsEnd();
           it++)
11     if ((*it)->intersect(r, cur_data) && cur_data.t < data.t &&
           cur_data.t > t_min)
12     {
13         flag = true;
14         data = std::move(cur_data);
15         data.iter = it;
16     }
17
18
19     return flag;
20 }
```

Листинг 3.6 – Расчёт отражённого луча

```
1  Vec3d RayTracingRendered::reflectedRay(const Vec3d &in, const Vec3d
       &n)
2  {
3      return in - 2.0 * (in & n) * n;
4  }
```

Листинг 3.7 – Расчёт преломлённого луча

```
1 Vec3d RayTracingRendered::refractedRay(const Vec3d &v, const Vec3d
    &n, double mi)
2 {
3     double cos_teta = (-v) & n, n_coef = (mi * cos_teta -
        std::sqrt(1 - (mi * mi * (1 - cos_teta * cos_teta))));
4     return mi * v + n_coef * n;
5 }
```

3.4 Описание интерфейса

На рисунке 3.1 представлен оконный интерфейс программы. Он предоставляет возможность добавить, удалить и изменить объекты сцены. Также можно изменять настройки окружающей среды и рендера и просматривать изображение, полученное в результате работы.

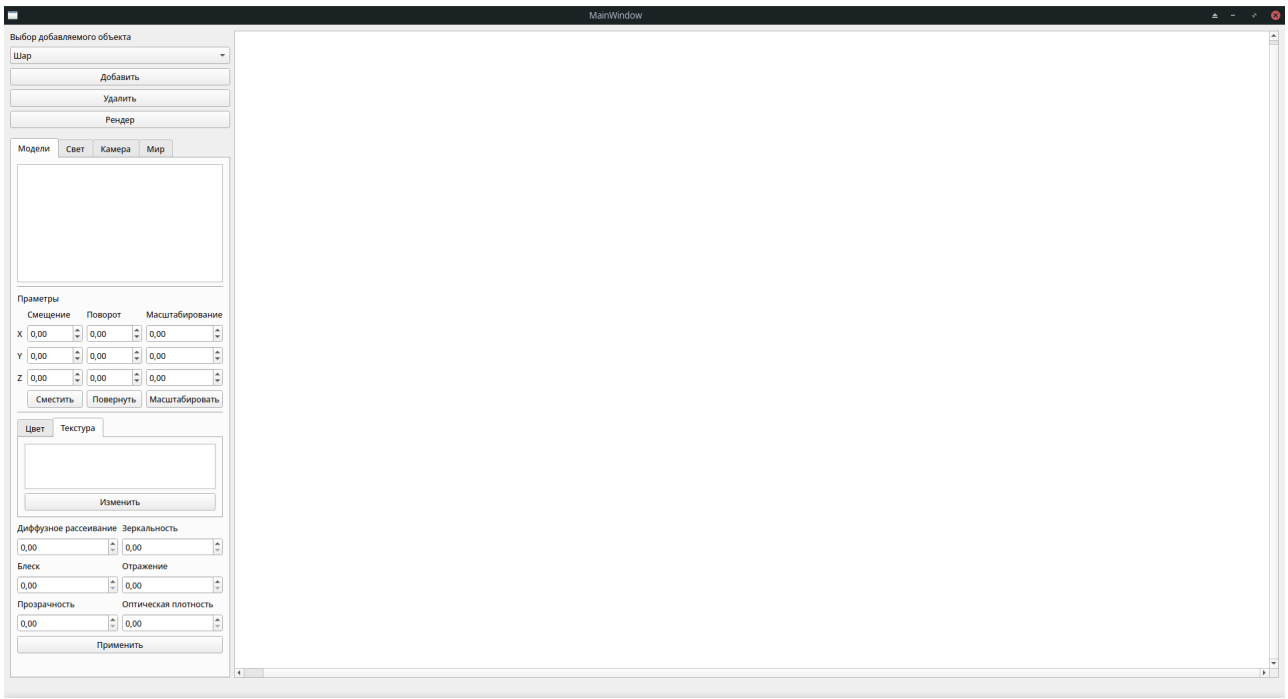

Рисунок 3.1 – Интерфейс программы

Для добавления модели необходимо выбрать в выпадающем меню пункт «Полигональная модель» и нажать кнопку «Добавить». Появится диалоговое окно для выбора файла, из которого будет импортирована модель, как на рисунке 3.2. Модель добавиться в список объектов. Аналогично добавляются другие модели и источники света.

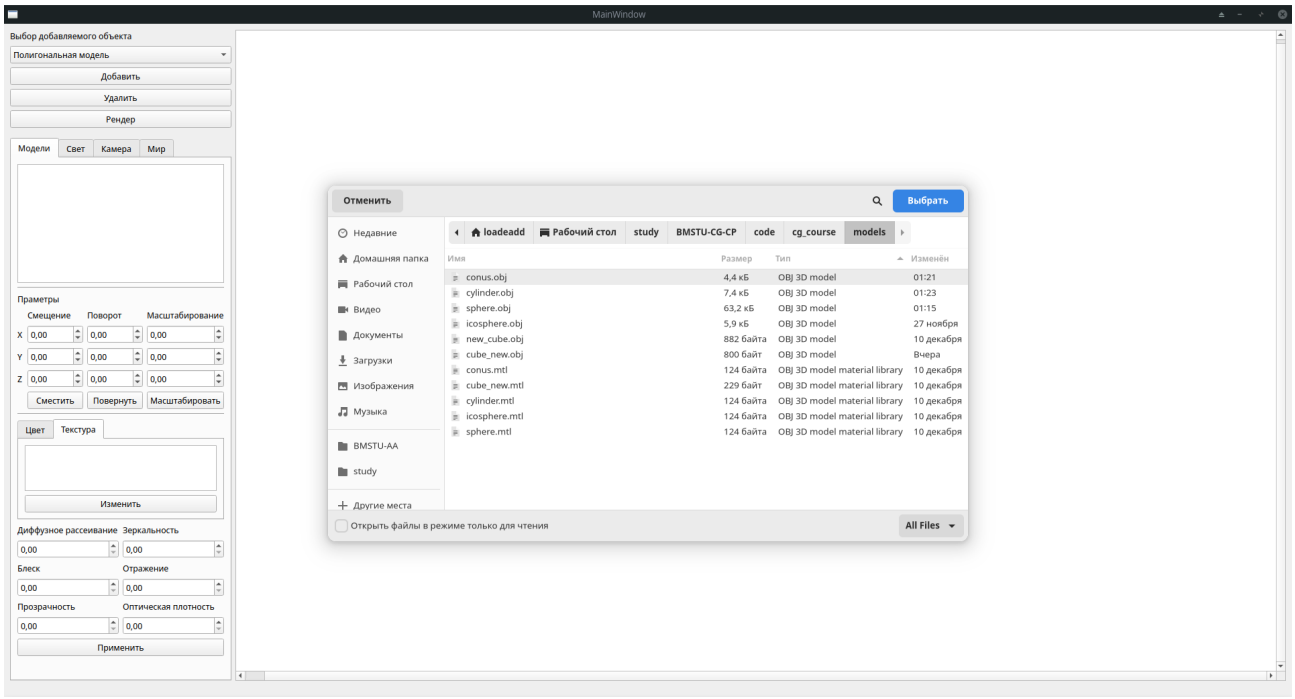

Рисунок 3.2 – Диалоговое окно для выбора модели

Для получения изображения необходимо нажать кнопку «Рендер» и в окне просмотра появится результат как на рисунке 3.3.

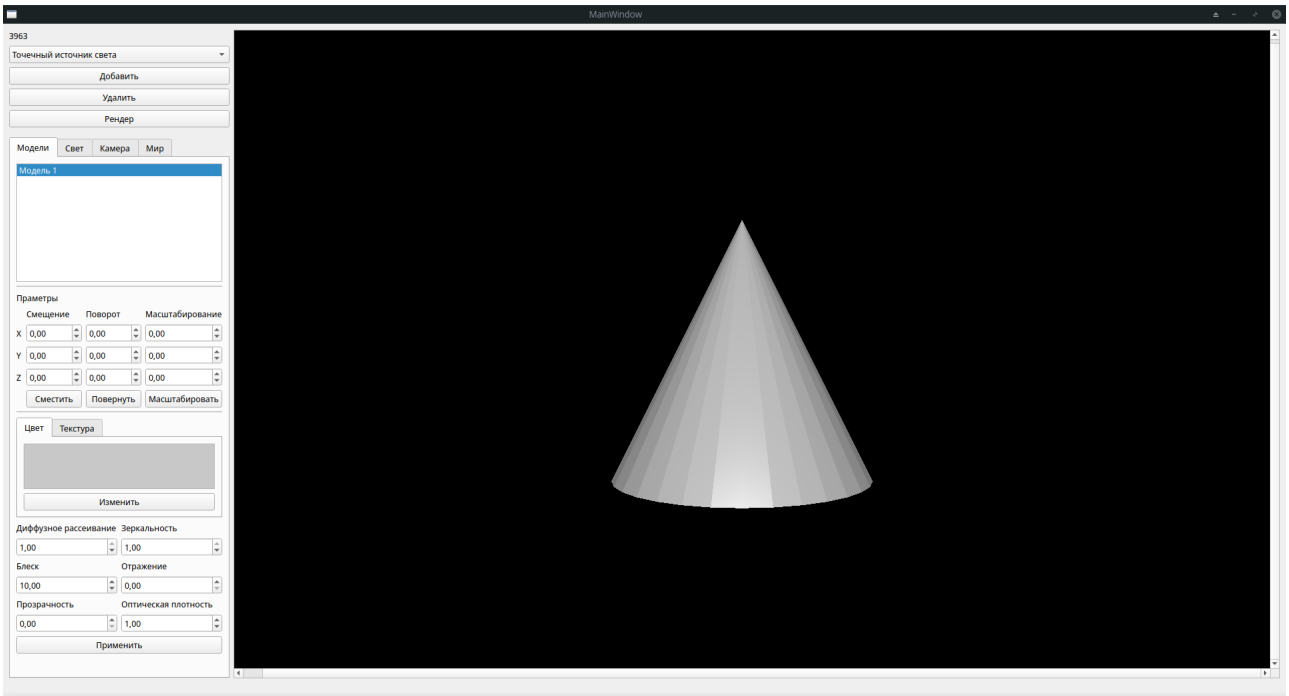

Рисунок 3.3 – Результат работы программы

В боковой панели можно выбрать один или несколько объектов и выполнить их преобразование или изменить оптические характеристики. Также есть возможность выбора цвета модели. При изменении цвета появляется

диалоговое окно, как на рисунке 3.4. Выбранный цвет появится в окне предпросмотра над кнопкой «Изменить».

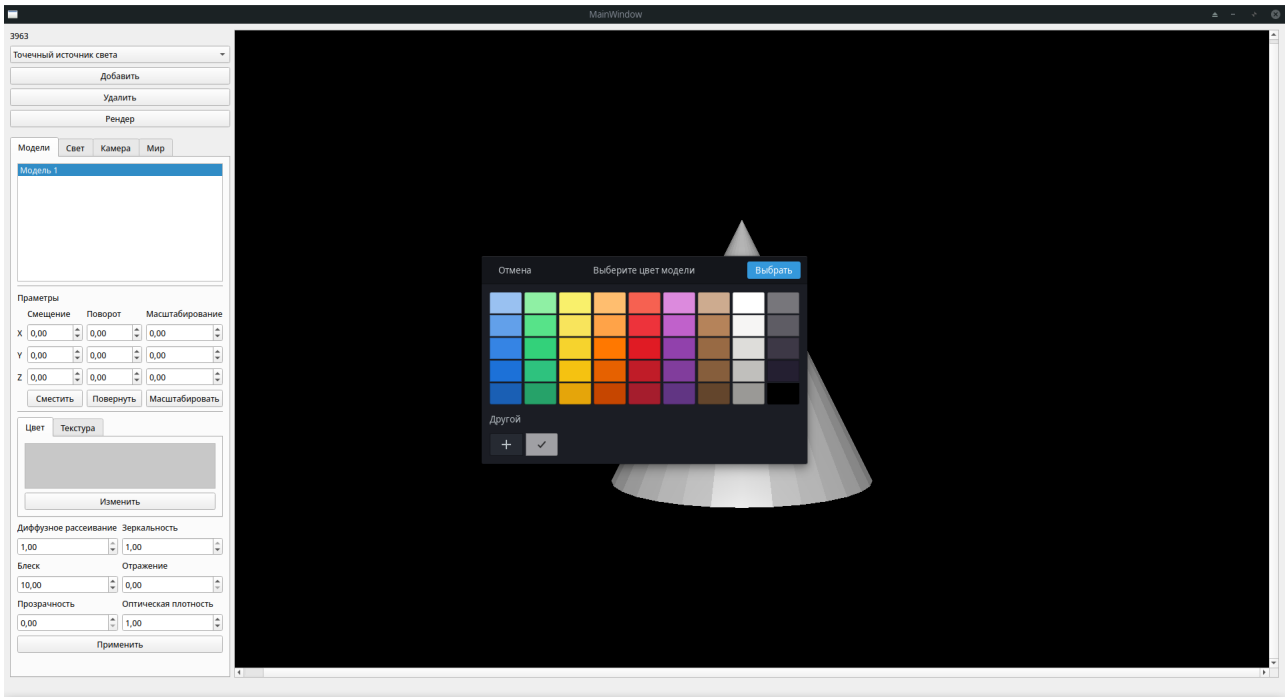

Рисунок 3.4 – Диалоговое окно для выбора цвета

Для наложения текстуры необходимо нажать на кнопку «Изменить» во вкладке «Текстура» и в диалоговом окне выбрать графическое изображение. Выбранная текстура появится в окне предпросмотра над кнопкой «Изменить», как на рисунке 3.5.

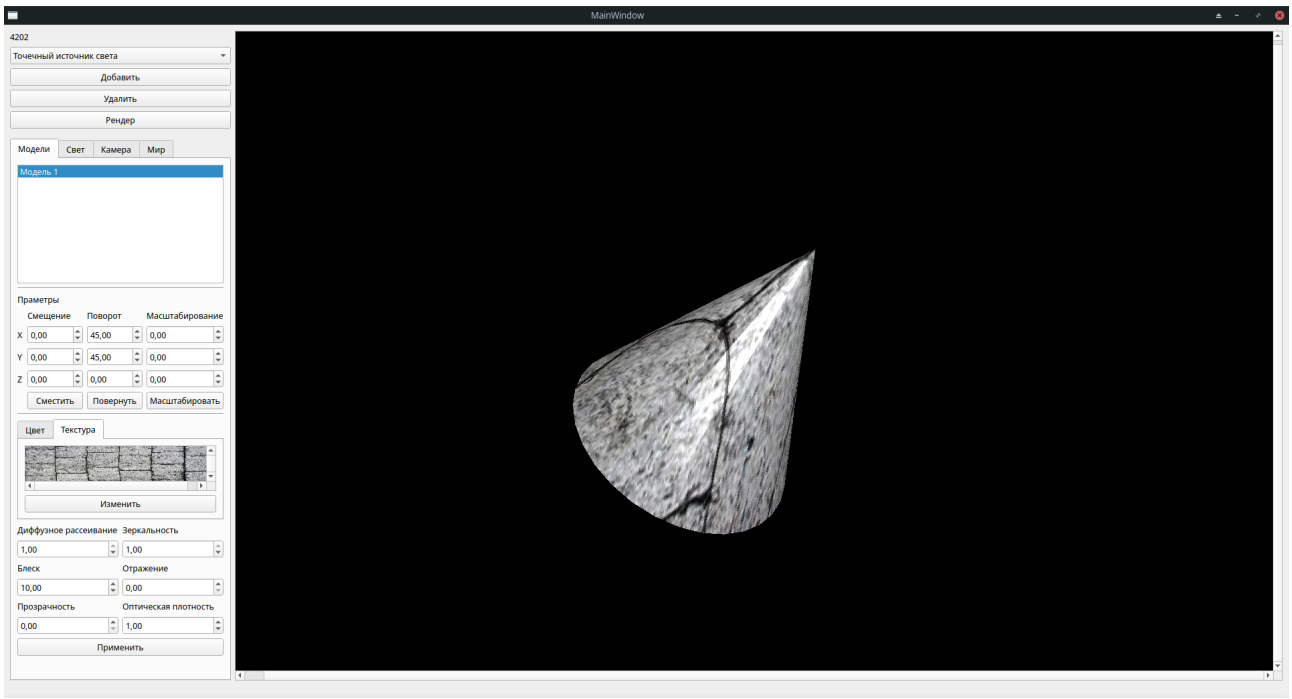

Рисунок 3.5 – Рендер объекта с наложенной текстурой

Аналогичным образом можно работать с источниками освещения и камерой. Дополнительные настройки окружающей среды и рендера находятся во вкладке «Мир», как на рисунке 3.6. При клике на любой объект сцены боковая панель заполняется его характеристиками.

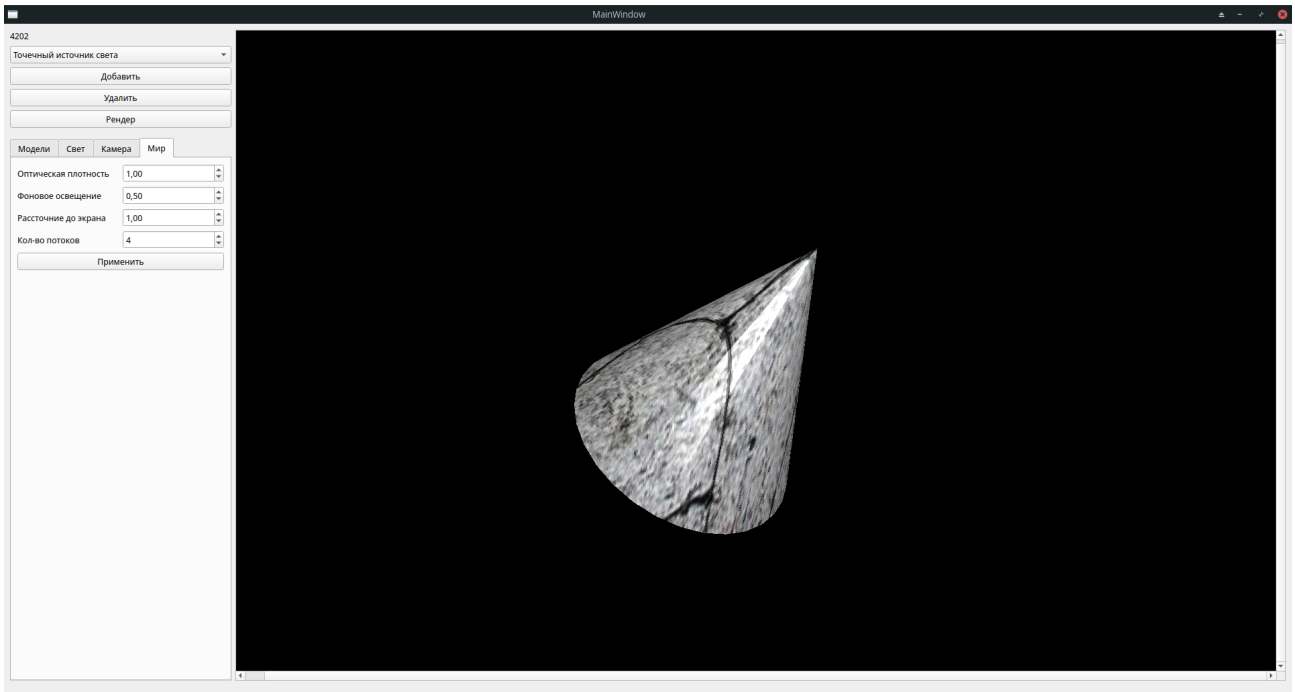

Рисунок 3.6 – Дополнительные настройки

Вывод

Было приведено описание структуры программы, выбраны средства реализации ПО, приведены листинги кода, и продемонстрирован интерфейс программы.

4 Экспериментальная часть

В данном разделе приведён пример работы программы, а также проведён анализ быстродействия программы в зависимости от количества потоков.

4.1 Технические характеристики

Технические характеристики устройства, на котором выполнялись замеры времени, представлены далее.

— Операционная система: Manjaro Linux 86_64 Xfce 4.16.

— Оперативная память: 8 Гбайт.

— Процессор: 11th Gen Intel i5-1135G7 (8) @ 4.200 Гц [6].

При тестировании ноутбук был включен в сеть электропитания. Во время тестирования ноутбук был нагружен только встроенными приложениями окружения, а также системой тестирования.

4.2 Демонстрация работы программы

На рисунке 4.1 представлен пример работы программы со следующими объектами:

— растянутый куб с текстурой деревянного забора;

— прозрачный цилиндр фиолетового цвета;

— зеркальный шар;

— конус с текстурой камня;

— 2 точечных источника света.

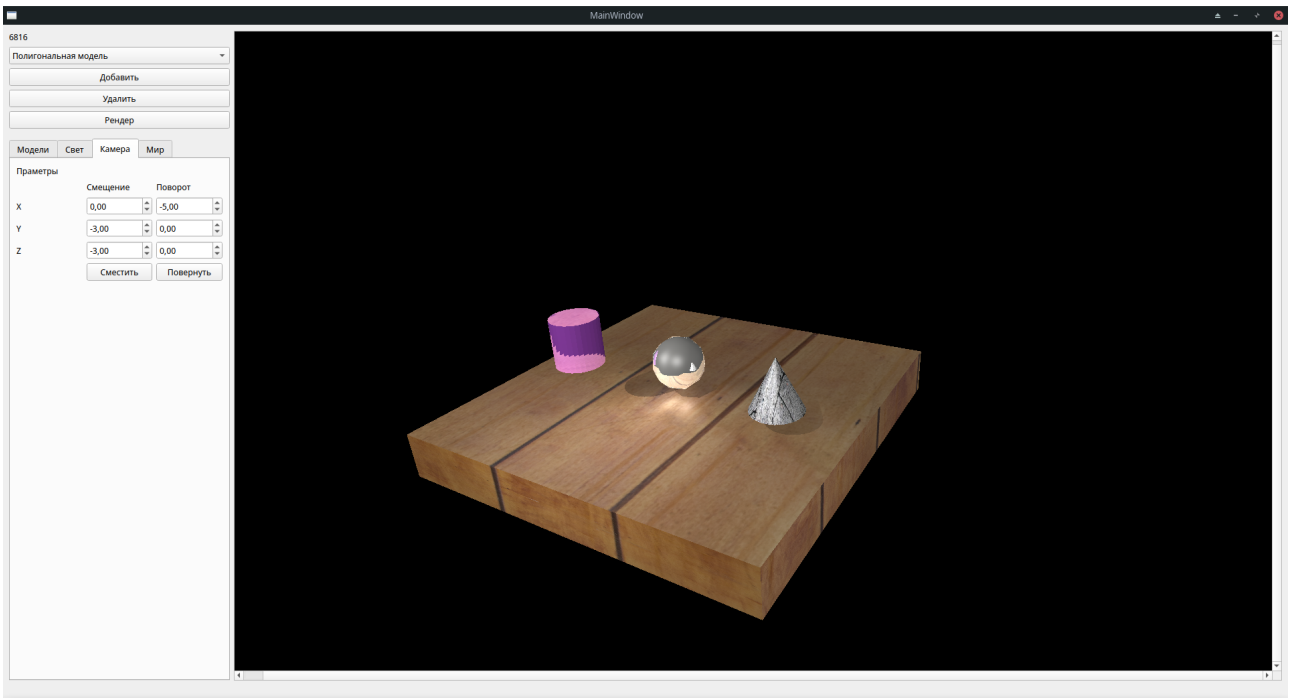

Рисунок 4.1 – Пример работы программы

На рисунке 4.2 представлен пример работы программы, демонстрирующий эффекты отражения и преломления.

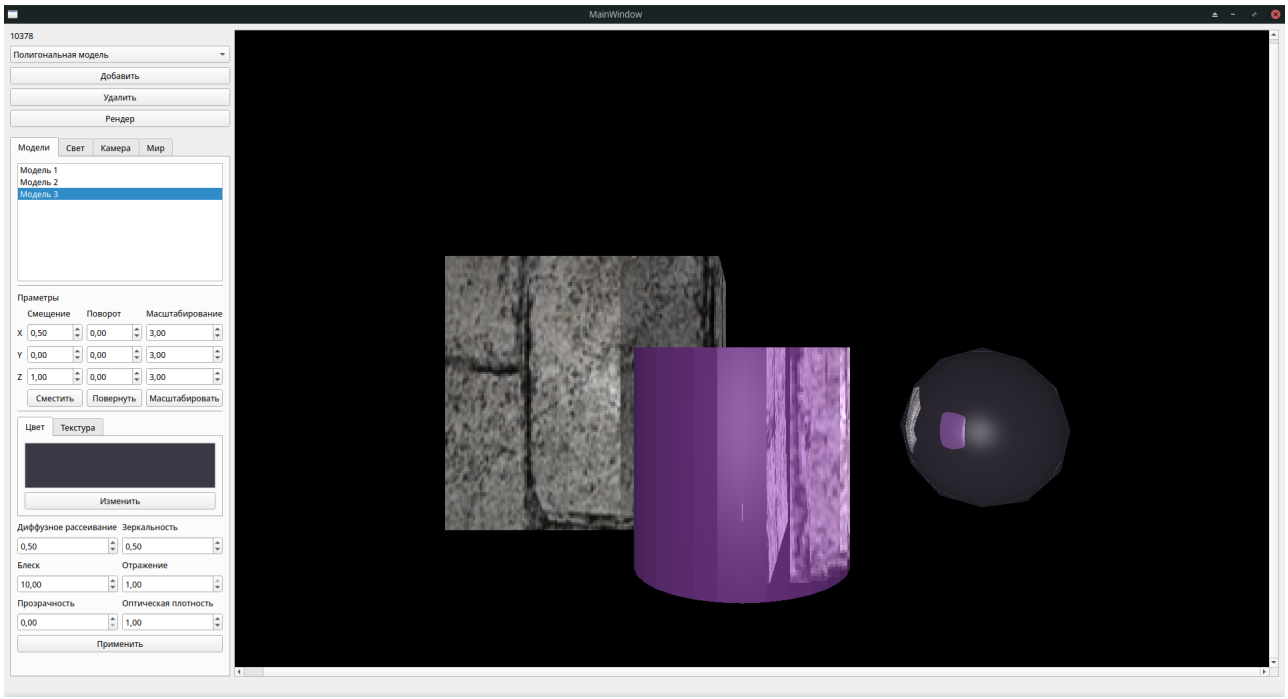

Рисунок 4.2 – Пример работы программы с глобальной моделью освещения

4.3 Описание эксперимента

Преимуществом алгоритма трассировки лучей является то, что он легко поддаётся распараллеливанию, так как все лучи считаются не зависимо друг от друга. В данной реализации сцена разделена на вертикальные полосы равного размера и каждый поток отвечает за отрисовку своего блока. Для проведения эксперимента будет использоваться сцена, как на рисунке 4.1.

Цель эксперимента — оценка времени работы реализации алгоритма в зависимости от количества потоков расчёта. Результат замеров приведен в таблице 4.1.

Таблица 4.1 – Результаты замеров

Кол-во потоков	Время, мс
1	19430
2	11719
4	10024
8	6050
12	6469
16	6927

На рисунке 4.3 представлен результат замеров времени работы алгоритма трассировки лучей. По горизонтальной оси отмечено количество потоков, по вертикальной — время в миллисекундах.

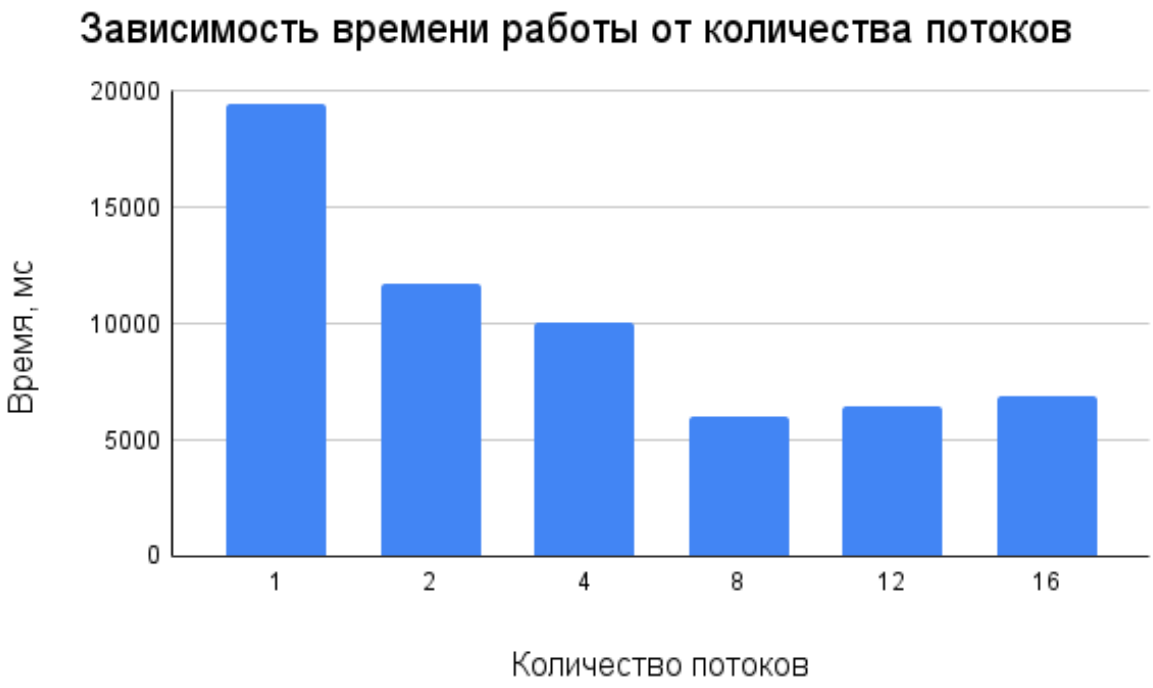

Рисунок 4.3 – Результаты замеров

4.4 Вывод

Были проведены замеры времени работы реализации алгоритма трассировки лучей. По результатам эксперимента видно, что программа затрачивает меньше всего времени на рендер сцены, когда количество выполняющих рендер потоков совпадает с количеством логических ядер процессора. Дальнейшее увеличение количества потоков не даёт сильного увеличения производительности.

Заключение

В рамках курсового проекта было создано ПО для создания трёхмерных реалистичных изображений с использованием метода трассировки лучей, учитывающее цвета и оптические свойства объектов.

Цель, поставленная в начале лабораторной работы, была достигнута: реализован и исследован алгоритм построения реалистичного изображения с применением трассировки лучей и глобальной моделью освещения.

В ходе выполнения курсовой работы были решены все задачи:

— описана визуализируемая сцена;

— описаны существующие алгоритмы построения реалистичных изображений, текстурирования, моделей освещения и способы представления объектов сцены, выбраны подходящие для построения реалистичного изображения;

— разработаны выбранные алгоритмы и структуры данных;

— все алгоритмы реализованы в виде программы с графическим интерфейсом;

— проведено исследование быстродействия разработанного ПО в зависимости от количества параллельно работающих потоков, выполняющих рендер изображения сцены.

Список использованных источников

1. Методы представления дискретных данных [Электронный ресурс]. Режим доступа: https://www.graphicon.ru/oldgr/ru/library/multires_rep/index.html (дата обращения: 21.09.22).

2. Роджерс Д. Алгоритмические основы машинной графики. — М.: Мир, 1989. — 512 с.

3. Гамбетта Г. Компьютерная графика. Рейтрейсинг и растеризация. — СПб.: Питер, 2022. — 224 с.

4. Простые модели освещения. Режим доступа: http://grafika.me/node/344 (дата обращения: 27.12.2022).

5. Поре в В. Н. Компьютерная графика. — СПб.: БХВ-Петербург, 2002. – 432 с. : ил.

6. Процессор Intel® Core™ i5-7300HQ [Электронный ресурс]. Режим доступа: https://ark.intel.com/content/www/ru/ru/ark/products/97456/intel-core-i5-7300hq-processor-6m-cache-up-to-3-50-ghz.html (дата обращения: 23.10.2022).