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

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К КУРСОВОМУ ПРОЕКТУ

НА ТЕМУ:

«Визуализация решения задачи Стефана»

Студент ИУ7-52Б _____ **И. А. Глотов**
(Группа) (Подпись, дата) (И.О.Фамилия)

Руководитель курсового проекта _____ **А. С. Кострицкий**
(Подпись, дата) (И.О.Фамилия)

2022 г.

Оглавление

Введение

В современной жизни мы постоянно сталкиваемся с компьютерной графикой. Средства визуализации крайне важны для инженеров и архитекторов, огромную роль играет компьютерная графика в рекламе и индустрии развлечений. Без нее было бы невозможным создание многих компьютерных игр. Компьютерная графика используется в науке и промышленности для моделирования и визуализации различных физических процессов. Компьютерная графика развивается быстрыми темпами, постоянно появляются новые методы и алгоритмы, позволяющие показывать сложные и захватывающие эффекты, затрачивая для этого все меньше и меньше вычислительных ресурсов [1].

Например, алгоритмы компьютерной графики могут использоваться для визуализации решения задачи Стефана.

Целью данной работы является разработка программного обеспечения, позволяющего получить реалистичное изображение полого кусочка льда с жидкостью внутри.

Чтобы достигнуть поставленной цели, требуется решить следующие задачи:

1) провести анализ алгоритмов построения реалистичных изображений;

2) разработать метод построения реалистичного изображения полого кусочка льда с жидкостью внутри;

3) реализовать метод построения реалистичного изображения полого кусочка льда с жидкостью внутри;

4) исследовать зависимость времени выполнения однопоточной и многопоточной реализаций метода построения реалистичного изображения полого кусочка льда с жидкостью внутри от размера изображения.

1. Аналитическая часть

В данном разделе проводится анализ задачи построения трёхмерного изображения сцены, рассматриваются различные методы, решающие данную задачу.

1.1. Описание модели трёхмерного объекта на сцене

Модели могут задаваться следующими способами [2]:

1) Каркасная (проволочная) модель. В этой модели задается информация о вершинах и рёбрах объектов. Это простейший вид моделей. Этим видам модели присущ недостаток: нельзя отличить видимые грани от невидимых.

2) Поверхностная модель. Поверхность может описываться аналитически, либо может задаваться другим способом. Недостаток: отсутствует информация о том, с какой стороны поверхности находится материал.

3) Объёмная модель. Эта форма модели отличается от поверхностной тем, что в объёмных моделях к информации о поверхности добавляется информация о том, с какой стороны расположен материал.

Для решения поставленной задачи не подойдёт проволочная модель, так как в этом случае нельзя будет отличить видимые грани от невидимых, что является существенным недостатком при построении реалистичного изображения. Поверхностная модель также не подойдёт, потому что для достижения поставленной цели курсовой работы необходимо знать, с какой стороны поверхности расположен материал. Таким образом, была выбрана объёмная модель.

1.2. Описание способа задания трёхмерного объекта на сцене

Существует несколько способов задания объёмной модели [2]:

1) Аналитический способ. Этот способ характеризуется описанием объекта в неявной форме, то есть для получения визуального представления нужно вычислять значения некоторой функции в различных точках пространства.

2) Полигональная сетка. Это совокупность вершин, рёбер и граней, которые определяют форму многогранного объекта в трёхмерной компьютерной графике и объёмном моделировании. Гранями обычно являются треугольники, так как любой полигон можно представить в виде треугольника. В свою очередь существуют различные способы хранения информации о полигональной сетке.

    1) Список граней. Характеризуется множеством граней и множеством вершин. В каждую грань входят как минимум три вершины.

    2) «Крылатое» представление. Представляет вершины, грани и ребра сетки. Его основные недостатки – дополнительные требования к объёму занимаемой памяти из-за содержания множества индексов и генерирование списка индексов граней.

    3) Вершинное представление. Описывает объект как множество вершин, соединенных с другими вершинами. Это простейшее представление, но оно не широко используемое, так как информация о гранях и ребрах не выражена явно. Поэтому нужно обойти все данные чтобы сгенерировать список граней для рендеринга.

При выборе способа задания объекта в курсовой работе определяющим фактором стала скорость выполнения геометрических преобразований.

Оптимальное представление – полигональная сетка. Такая модель позволит легко описывать сложные объекты сцены.

Способом хранения информации о полигональной сетке был выбран список граней, потому что этот способ даёт явное описание граней, что позволяет

выполнять геометрические преобразования над объектами сцены без генерирования дополнительных списков, как в вершинном и «крылатом» представлениях.

1.3. Формализация объектов сцены

Сцена состоит из следующих объектов:

1) Геометрический объект представляется в виде полигональной сетки. Для описания геометрического объекта требуется указать координаты вершин, связи вершин (рёбра), фоновое освещение (цвет), диффузное освещение (цвет), зеркальное освещение (цвет), коэффициент фонового освещения, коэффициент диффузного освещения, коэффициент зеркального освещения, степень, аппроксимирующая пространственное распределение зеркально отражённого света, коэффициент отражения, коэффициент преломления, показатель преломления.

2) Источник света представляется в виде точечного объекта. К его характеристикам относятся расположение, цвет и интенсивность излучения.

3) Камера характеризуется пространственным положением и направлением взгляда.

1.4. Анализ задачи построения трёхмерного изображения сцены из объектов

Построение трёхмерного изображения сцены заключается в преобразовании объектов этой сцены в изображение на растровом дисплее. Для создания реалистичного изображения необходимо учитывать несколько факторов.

1) Невидимые линии и поверхности.

2) Тени.

3) Освещение.

4) Свойства материалов объекта: способность отражать свет, способность преломлять свет.

1.4.1. Удаление невидимых линий и поверхностей

Задача удаления невидимых линий и поверхностей является одной из наиболее сложных в компьютерной графике [2]. Алгоритмы удаления невидимых линий и поверхностей служат для определения линий ребер, поверхностей или объемов, которые видимы или невидимы для наблюдателя, находящегося в заданной точке пространства.

Алгоритмы удаления невидимых линий и поверхностей делятся на:

1) Алгоритмы, работающие в объектном пространстве (мировая система координат, высокая точность).

2) Алгоритмы, работающие в пространстве изображений (система координат связана с дисплеем, точность ограничена разрешающей способностью дисплея).

Рассмотрим алгоритмы удаления невидимых линий и поверхностей.

Алгоритм Робертса

Данный алгоритм работает в объектном пространстве, решая задачу только с выпуклыми телами [2].

Алгоритм выполняется в три этапа.

1) Этап подготовки исходных данных. На данном этапе задана информация о телах. Для каждого тела сцены сформирована матрица тела V. Размер матрицы – $4n$, где n – количество граней тела.

   Каждый столбец матрицы представляет собой четыре коэффициента уравнения плоскости, проходящей через очередную грань:

$$ax + by + cz + d = 0. \tag{1.1}$$

   Таким образом, матрица тела будет представлена в следующем виде:

$$V = \begin{pmatrix} a_1 & a_2 & \dots & a_n \\ b_1 & b_2 & \dots & b_n \\ c_1 & c_2 & \dots & c_n \\ d_1 & d_2 & \dots & d_n \end{pmatrix}. \quad (1.2)$$

Матрица тела сформирована корректно, то есть любая точка, расположенная внутри тела, должна располагаться по положительную сторону от каждой грани тела. В случае, если для очередной грани условие не выполняется, соответствующий столбец матрицы надо умножить на -1.

2) Этап удаления ребер, экранируемых самим телом. На данном этапе рассматривается вектор взгляда $E = \{0, 0, -1, 0\}^T$. Для определения невидимых граней нужно умножить вектор E на матрицу тела V. Отрицательные компоненты полученного вектора будут соответствовать невидимым граням.

3) Этап удаления невидимых ребер, экранируемых другими телами сцены. На данном этапе для определения невидимых точек ребра требуется построить луч, соединяющий точку наблюдения с точкой на ребре. Точка будет невидимой, если луч на своем пути встречает в качестве преграды рассматриваемое тело. Если тело является преградой, то луч должен пройти через тело. Если луч проходит через тело, то он находится по положительную сторону от каждой грани тела.

Свойства алгоритма Робертса:

1) алгоритм работает в объектном пространстве, точность вычислений высокая;

2) теоретический рост сложности алгоритма – квадрат числа объектов;

3) все тела сцены должны быть выпуклыми.

Алгоритм Робертса не подходит для решения поставленной задачи по следующим причинам.

1) Возникновение проблем при наличии невыпуклых тел на сцене.

2) Невозможность визуализации зеркальных поверхностей.

Алгоритм, использующий z-буфер

Данный алгоритм работает в пространстве изображения [2]. Используется два буфера: буфер кадра, в котором хранятся атрибуты каждого пикселя в пространстве изображения и z-буфер, куда помещается информация о координате z для каждого пикселя.

Алгоритм выполняется в несколько этапов.

1) Первоначально в z-буфере находятся минимально возможные значения z, а в буфере кадра располагаются пиксели, описывающие фон.

2) Каждый многоугольник преобразуется в растровую форму и записывается в буфер кадра.

3) В процессе подсчета глубины нового пикселя, он сравнивается с тем значением, которое уже лежит в z-буфере. Если новый пиксель расположен ближе к наблюдателю, чем предыдущий, то он заносится в буфер кадра и происходит корректировка z-буфера.

4) Для решения задачи вычисления глубины z каждый многоугольник описывается уравнением $ax + by + cz + d = 0$. При $c = 0$ многоугольник для наблюдателя вырождается в линию.

   Для решения задачи вычисления глубины z каждый многоугольник описывается уравнением

   $$ax + by + cz + d = 0. \tag{1.3}$$

   При $c = 0$ многоугольник для наблюдателя вырождается в линию.

   Для некоторой сканирующей строки $y = const$, поэтому имеется возможность рекуррентно высчитывать z' для каждого $x' = x + dx$:

   $$z' - z = -\frac{ax' + d}{c} + \frac{ax + d}{c} = \frac{a(x - x')}{c} \tag{1.4}$$

   Получим: $z' = z - \frac{a}{c'}$, так как $x - x' = dx = 1$.

5) Для невыпуклых многогранников предварительно потребуется удалить нелицевые грани.

Свойства алгоритма, использующего z-буфер:

1) алгоритм имеет линейную сложность.

2) большой объём требуемой памяти (два буфера размером $n \cdot m$, где n – количество пикселей растра в горизонтальном измерении, m – количество пикселей растра в вертикальном измерении);

3) реализация эффектов прозрачности сложна;

4) реализация эффектов зеркальности невозможна;

5) дополнительные вычислительные операции в случае невыпуклых тел.

Алгоритм, использующий z-буфер не подходит для решения поставленной задачи по следующим причинам.

1) Сложность визуализации прозрачных поверхностей.

2) Невозможность визуализации зеркальных поверхностей.

Другие алгоритмы растеризации

Стоит отметить, что ни один из алгоритмов растеризации (в том числе рассмотренные алгоритмы Робертса и использующий z-буфер) не способен визуализировать отражающие поверхности [2]. Из-за этого существенного для данной задачи недостатка не имеет смысла рассматривать другие алгоритмы растеризации.

Алгоритм обратной трассировки лучей

Главная идея, лежащая в основе алгоритма трассировки лучей, заключается в том, что наблюдатель видит любой объект посредством испускаемого луча неким источником света, который падает на этот объект и затем каким-то путём доходит до наблюдателя. Свет может достичь наблюдатель, отразившись от поверхности, преломившись или пройдя через неё. Если проследить за лучами света, выпущенными источниками, то можно убедиться, что весьма немногие из них дойдут до наблюдателя. Следовательно, этот процесс

был бы вычислительно неэффективным. Аппель первым предложил отслеживать (трассировать) лучи в обратном направлении, т. е. от наблюдателя к объекту [2].

Агоритм выполняется в несколько этапов.

1) Сцена преобразовывается в пространство изображения. Перспективное преобразование не используется. Считается, что точка зрения или наблюдатель находится в бесконечности на положительной полуоси z. Поэтому все световые лучи параллельны оси z.

2) Каждый луч, исходящий от наблюдателя, проходит через центр пикселя на растре до сцены.

3) Необходимо проверить пересечение каждого объекта сцены с каждым лучом.

4) Если луч пересекает объект, то определяются все возможные точки пересечения луча и объекта. Эти пересечения упорядочиваются по глубине.

5) Пересечение с максимальным значением глубины представляет видимую поверхность для данного пикселя. Свойства материала этого объекта используются для определения характеристик пикселя.

6) Если материал объекта обладает отражающими и/или преломляющими свойствами, то луч рекурсивно отражается и/или преломляется.

7) Для учёта теней испускается луч из точки пересечения с объектом к источнику света.

8) Если данный луч пересекает какой-либо объект сцены, то точка пересечения первичного луча с объектом считается теневой.

9) Если точка зрения находится не в бесконечности, строится одноточечная центральная проекция на картинную плоскость.

Свойства алгоритма обратной трассировки лучей:

1) высокая реалистичность синтезируемого изображения;

2) реализация алгоритма предполагает учёт теней;

3) простота визуализации зеркальных и прозрачных поверхностей.

4) время выполнения рендеринга изображения, по сравнению с алгоритмом Робертса и алгоритмом, использующего z-буфер, увеличивается из-за рекурсивных погружений.

Вывод

Таким образом, в качестве алгоритма удаления невидимых рёбер и поверхностей был выбран алгоритм обратной трассировки лучей из-за высокой реалистичности синтезируемого изображения и возможности визуализации отражающих и прозрачных поверхностей.

1.4.2. Учёт теней

При использовании алгоритма обратной трассировки лучей, выбранного в качестве алгоритма удаления невидимых рёбер и поверхностей в главе 1.4.1., построение теней происходит по ходу выполнения алгоритма: пиксель будет затемнён, если луч испускаемый из точки попадания первичного луча, испускаемого из камеры, попадает на другой объект.

1.4.3. Учёт освещения

Модель освещения предназначена для того, чтобы рассчитать интенсивность отражённого к наблюдателю света в каждой точке (пикселе) изображения [2]. Глобальная модель учитывает не только свет и ориентацию поверхностей, но также и свет, отражённый от других объектов (или пропущенный через них) Благодаря этому глобальная модель освещённости способна воспроизводить эффекты зеркального отражения и преломления лучей (прозрачность и полупрозрачность), а также затенение, что является необходимым для решения поставленной в курсовой работе задачи. Глобальная модель является составной частью алгоритма удаления невидимых рёбер и поверхностей с помощью обратной трассировки лучей.

Глобальная модель освещения для каждого пикселя изображения определяет его интенсивность. Сначала определяется непосредственная освещённость источниками без учёта отражений от других поверхностей (вторичная освещённость): отслеживаются лучи, направленные ко всем источникам. Тогда наблюдаемая интенсивность (или отражённая точкой энергия) выражается следующим соотношением:

$$I = k_0 I_0 + k_d \sum_j I_j (n \cdot l_j) + k_r \sum_j I_j (s \cdot r_j)^\beta + k_r I_r + k_t I_t, \tag{1.5}$$

где

— k_0 – коэффициент фонового освещения,

— k_d – коэффициент диффузного отражения,

— k_r – коэффициент зеркального отражения,

— k_t – коэффициент пропускания,

— n – единичный вектор нормали к поверхности в точке,

— l_j – единичный вектор, направленный к j-му источнику света,

— s – единичный локальный вектор, направленный в точку наблюдения,

— r_j – отражённый вектор l_j,

— I_0 – интенсивность фонового освещения,

— I_j – интенсивность j-го источника света,

— I_r – интенсивность, приходящая по зеркально отражённому лучу,

— I_t – интенсивность, приходящая по преломлённому лучу.

1.5. Выводы из аналитической части

Были рассмотрены способы задания трёхмерных моделей и выбрана объёмная форма задания моделей.

Также были рассмотрены алгоритмы удаления невидимых рёбер и поверхностей:

1) алгоритм Робертса;

2) алгоритм, использующий z-буфер;

3) алгоритм обратной трассировки лучей.

В качестве алгоритма удаления невидимых рёбер и поверхностей был выбран алгоритм обратной трассировки лучей с глобальной моделью освещения из-за высокой реалистичности синтезируемого изображения и возможности визуализации зеркальных и прозрачных поверхностей.

2. Конструкторская часть

В данном разделе представлены математические основы алгоритма обратной трассировки лучей, разработка алгоритма обратной трассировки лучей и разработка типов и структур данных.

2.1. Математические основы алгоритма обратной трассировки лучей

2.1.1. Поиск пересечения луча с полигонами

Для поиска пересечения луча с полигонами используется барицентрический тест [3]. Это самый известный тест на пересечение «луч-треугольник». Имея три точки на плоскости, можно выразить любую другую точку через ее барицентрические координаты.

Пусть луч $R(t)$ с началом в точке O и нормализованным вектором направления D определяется как:

$$R(t) = O + tD \tag{2.1}$$

Пусть вершины полигона обозначаются как V_0, V_1, V_2 Тогда, точка $T(u, v)$ в полигоне задаётся выражением:

$$T(u, v) = (1 - u - v)V_0 + uV_1 + vV_2, \tag{2.2}$$

где (u, v) – барицентрические координаты $(u \geq 0, v \geq 0, u + v \leq 1)$

Вычисление пересечения луча $R(t)$ и треугольника эквивалентно решению уравнения $R(t) = T(u, v)$. В этом случае получим:

$$O + tD = (1 - u - v)V_0 + uV_1 + vV_2 \tag{2.3}$$

В матричном виде:

$$\begin{bmatrix} -D & V_1 - V_0, V_2 - V_0 \end{bmatrix} \begin{bmatrix} t \\ u \\ v \end{bmatrix} = O - V_0 \tag{2.4}$$

Это означает, что барицентрические координаты (u, v) и расстояние t от точки испускания луча до точки пересечения луча с полигоном могут быть найдены путём решения СЛАУ (системы линейных алгебраических уравнений), которая написана выше.

Вышесказанное можно рассматривать геометрически как перевод полигона (треугольника) в начало координат и преобразование его в в треугольник с единичными длинами по y и z с направлением луча по оси x. Это показано на рисунке 2.1.

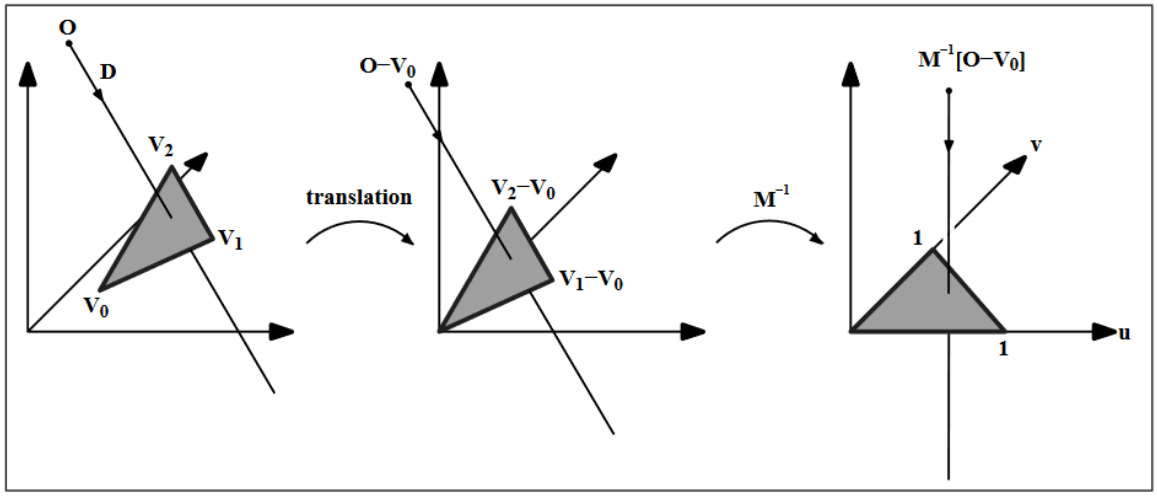


Рисунок 2.1 – Иллюстрация для расчёта отражённого луча

Пусть $E_1 = V_1 - V_0, E_2 = V_2 - V_0$ и $T = O - V_0$. Тогда, используя метод Крамера для (2.4):

$$\begin{bmatrix} t \\ u \\ v \end{bmatrix} = \frac{1}{(D \times E_2) \cdot E_1} \begin{bmatrix} (T \times E_1) \cdot E_2 \\ (D \times E_2) \cdot T \\ (T \times E_1) \cdot D \end{bmatrix} = \frac{1}{P \cdot E_1} \begin{bmatrix} Q \cdot E_2 \\ P \cdot T \\ Q \cdot D \end{bmatrix}, \tag{2.5}$$

где $P = D \times E_2$ и $Q = T \times E_1$.

2.1.2. Поиск нормали к полигонам

Для поиска нормали к полигонам необходимо найти векторное произведение двух векторов, которые лежат на полигоне [2].

$$N = (V_2 - V_0) \times (V_1 - V_0), \tag{2.6}$$

где V_0, V_1, V_2 – вершины полигона.

2.1.3. Поиск направления преломлённого и отражённого лучей

Для алгоритма обратной трассировки лучей нужно уметь находить отражённый и преломлённый лучи, при этом учитывая модель освещения Уиттеда. Отражённый луч можно найти, зная направление падающего луча и нормаль к поверхности [2].

Пусть L – направление луча, а n – нормаль к поверхности. Луч можно разбить на две части: L_p которая перпендикулярна нормали, и L_n – параллельна нормали.

Представленная ситуация изображена на рисунке 2.2.

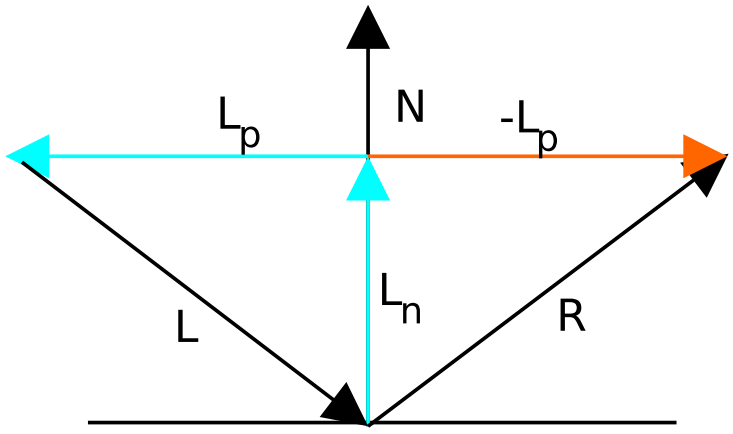


Рисунок 2.2 – Иллюстрация для расчёта отражённого луча

Учитывая свойства скалярного произведения $L_n = n \cdot (n, L)$ и $L_p = L - n \cdot (n, L)$ Так как отражённый луч выражается через разность этих векторов, то отражённый луч выражается по формуле (2.7):

$$R = 2n \cdot (n, L) - L \tag{2.7}$$

По закону преломления падающий, преломлённый лучи и нормаль к поверхности лежат в одной плоскости. Пусть $\{\mu_i\}$ – показатели преломления сред, а $\{\eta_i\}$ – углы падения и отражения света соответственно. Применяя закон Снеллиуса, параметры преломлённого луча можно вычислить по формуле (2.8):

$$R = \frac{\mu_1}{\mu_2} L + (\frac{\mu_1}{\mu_2} \cos(\eta_1) - \cos(\eta_2))n, \tag{2.8}$$

где $\cos(\eta_2) = \sqrt{1 - (\frac{\mu_1}{\mu_2})^2 \cdot (1 - \cos(\eta_1))^2}$

2.2. Разработка алгоритма обратной трассировки лучей

На рисунках 2.3 – 2.6 представлены схемы алгоритма испускания луча и алгоритмов поиска пересечения луча со сценой, с объектом и с полигоном.

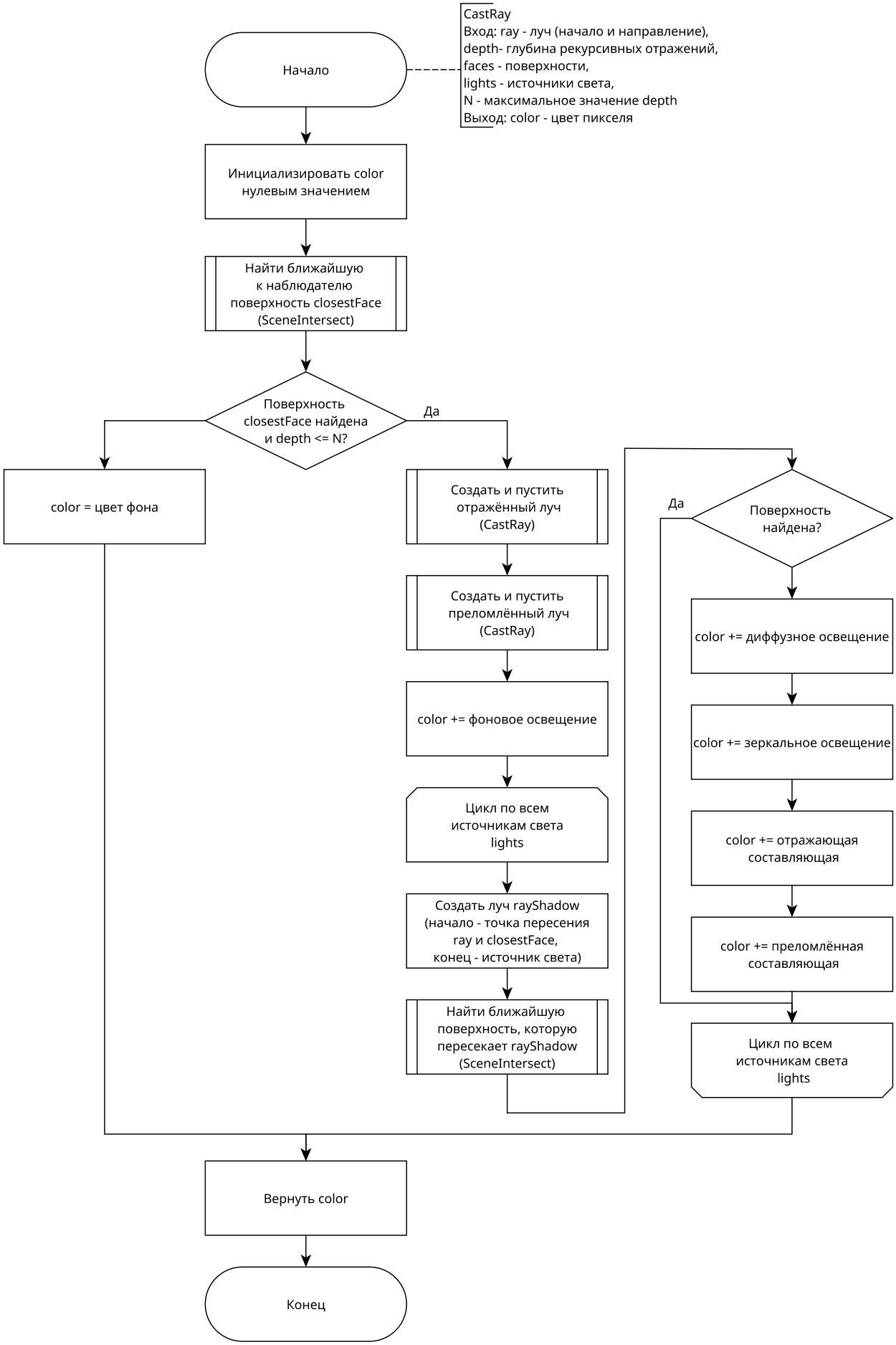


Рисунок 2.3 – Схема алгоритма испускания луча

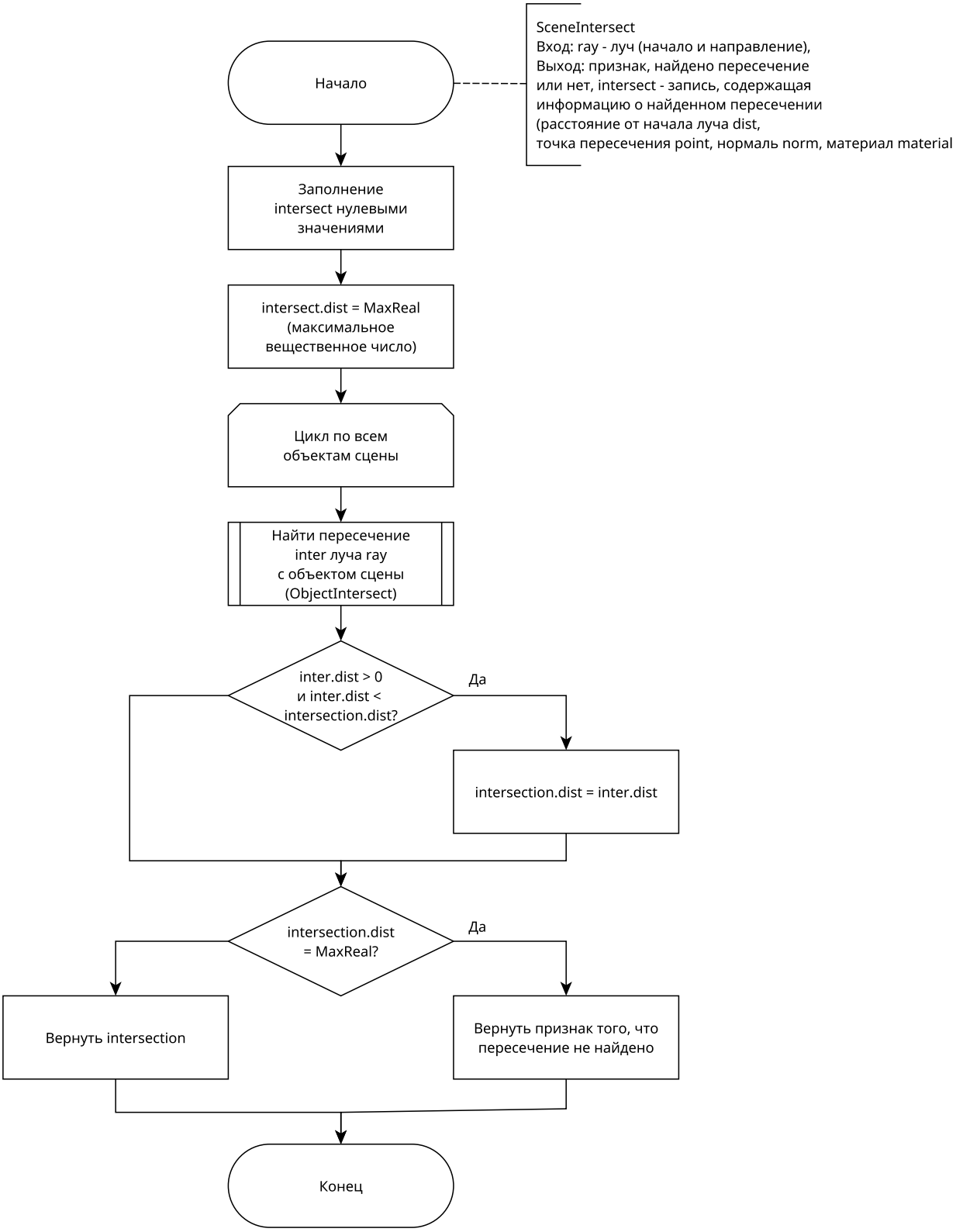


Рисунок 2.4 – Схема алгоритма пересечения луча со сценой

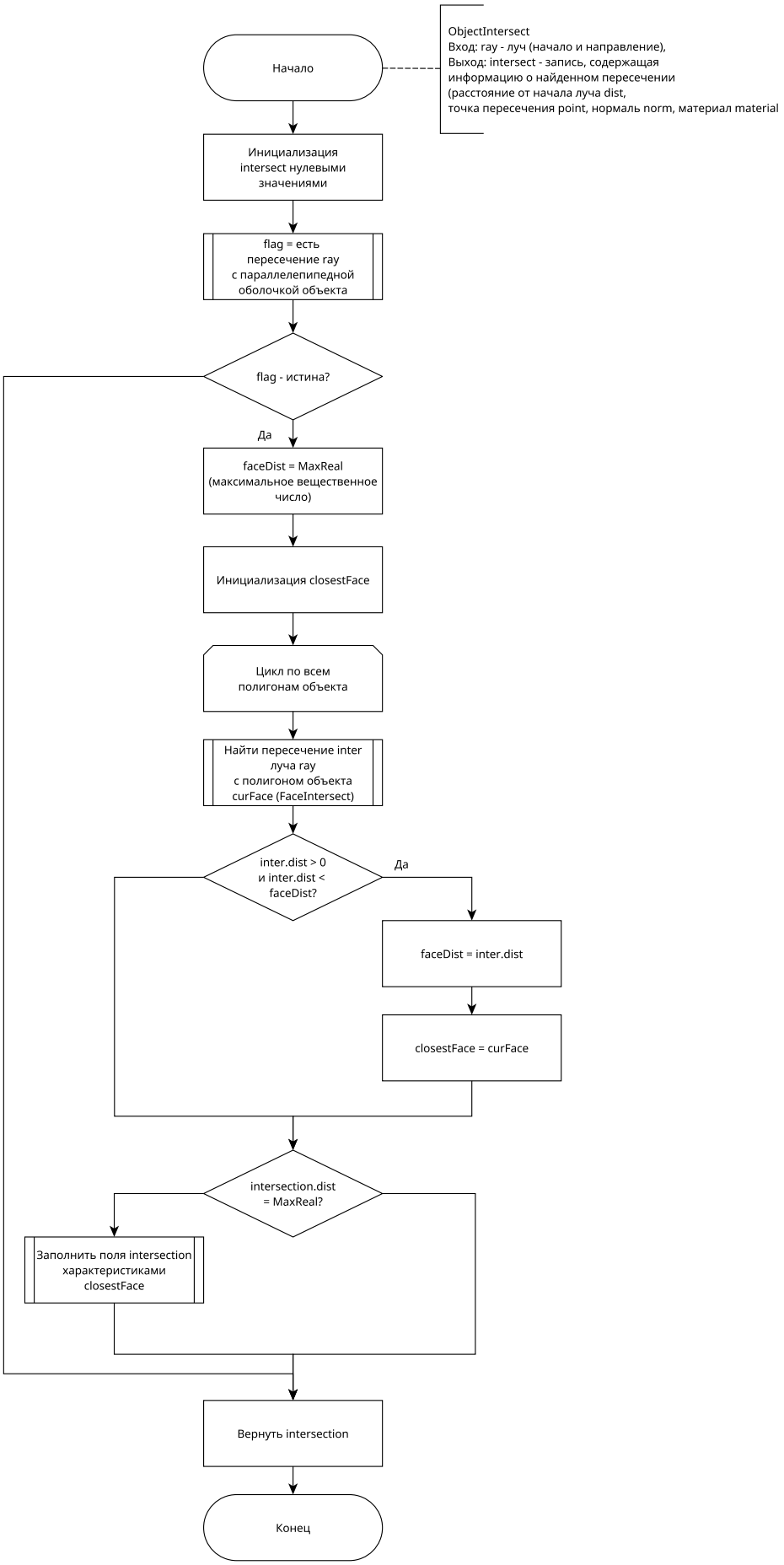


Рисунок 2.5 – Схема алгоритма пересечения луча с объектом

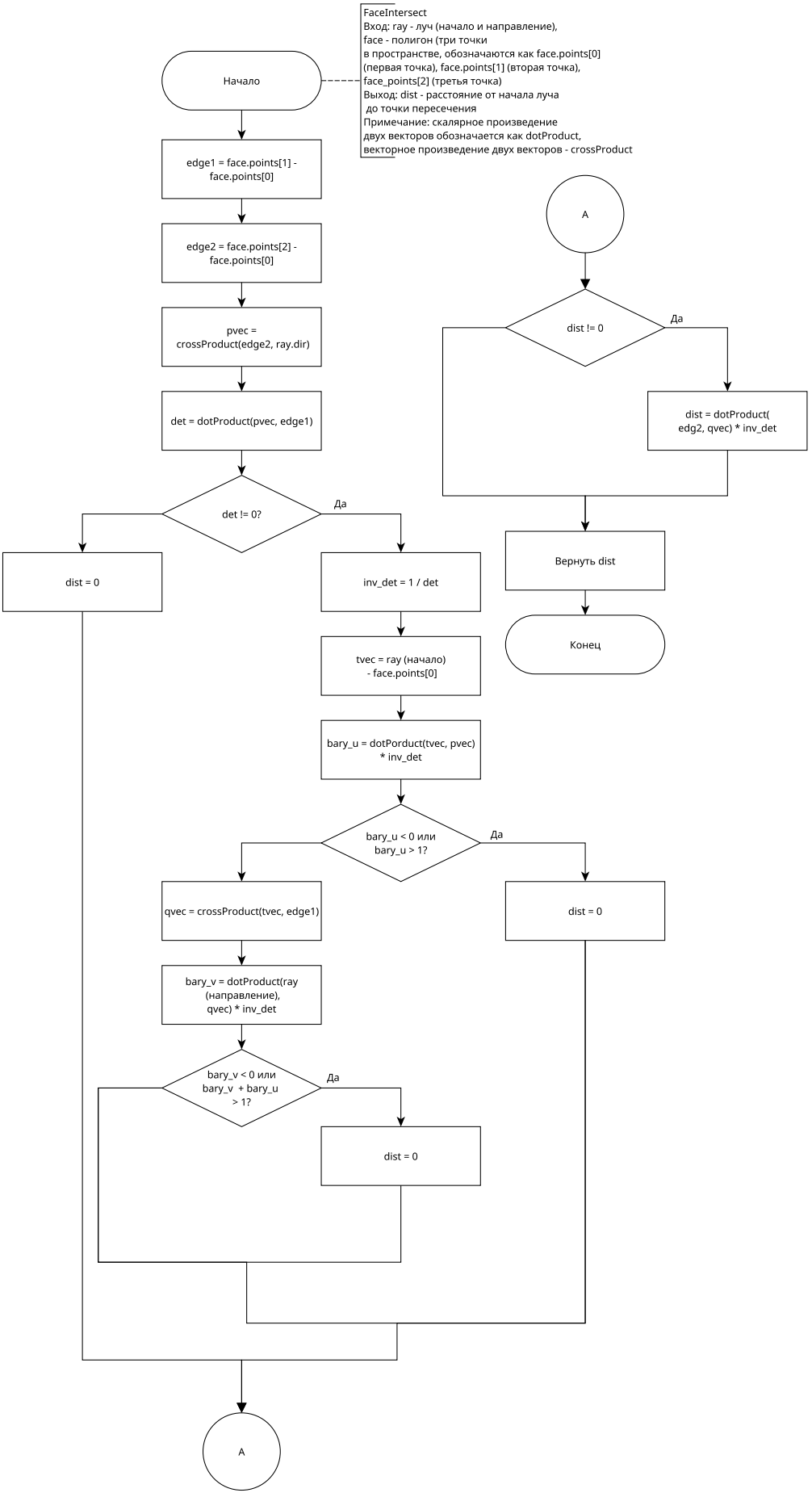


Рисунок 2.6 – Схема алгоритма пересечения луча с полигоном

2.3. Разработка типов и структур данных

Для формализации алгоритма синтеза изображения в программе, необходимо ввести использующиеся в ней типы и структуры данных.

1) Структура сцены представляет собой массив с произвольным числом моделей и массив с произвольным числом источников света.

2) Структура модели включает в себе следующие данные:

   — массив вершин модели;

   — массив полигонов модели;

   — структуру материала модели.

3) Структура материала содержит в себе:

   — Цвет фонового освещения (три целочисленных переменных, характеризующие цветовую модель RGB).

   — Цвет диффузного освещения (три целочисленных переменных, характеризующие цветовую модель RGB).

   — Цвет зеркального освещения (три целочисленных переменных, характеризующие цветовую модель RGB).

   — Коэффициент фонового освещения (вещественная переменная).

   — Коэффициент диффузного освещения (вещественная переменная).

   — Коэффициент зеркального освещения (вещественная переменная).

   — Степень, аппроксимирующая пространственное распределение зеркально отражённого света (целочисленная переменная).

   — Коэффициент отражения (вещественная переменная).

   — Коэффициент преломления (вещественная переменная).

   — Показатель преломления (вещественная переменная).

4) Структура камеры содержит:

   — Координаты положения камеры в пространстве (три вещественных переменных).

— Система координат камеры, задаваемая тремя ортогональными векторами.

— Границы пирамиды видимости (две целочисленных переменных).

5) Структура источника света содержит:

— Координаты положения источника света в пространстве (три вещественных переменных).

— Интенсивность излучения (вещественная переменная).

— Цвет излучения (три целочисленных переменных, характеризующие цветовую модель RGB).

2.4. Выводы из конструкторской части

На основе теоретических данных, полученных из аналитического раздела, были описаны математические основы алгоритма обратной трассировки лучей, алгоритм обратной трассировки лучей, а также было описано представление данных в программном обеспечении.

3. Технологическая часть

В данном разделе представлены выбор средств реализации, формат входных и выходных данных, требования к ПО, разработанные типы и структуры данных и алгоритмы, интерфейс ПО и тестирование ПО.

3.1. Выбор средств реализации

Для разработки программы был выбран язык C++. Данный выбор обусловлен следующими факторами.

1) C++ обладает высокой вычислительной производительностью, что очень важно для выполнения поставленной задачи [4].

2) C++ поддерживает парадигму объектно-ориентированного программирования [5].

3) Для C++ существует большое количество научной и учебной литературы по алгоритмам компьютерной графики ($\approx$ 17000 результатов запроса «c++ computer graphics algorithms»в поисковой системе Академия Google).

3.2. Формат входных и выходных данных и обоснование выбора

Входными данными для разрабатываемого программного обеспечения является информация о сцене (о всех её объетов). Для представления входных данных был выбран текстовый файл формата OBJ, так как согласно [6] формат OBJ:

1) не привязан к какой-либо программе, работающей с 3D моделированием;

2) занимает третье место в рейтинге по количеству моделей (на 17.10.2022 первое место в рейтинге, указанным в статье);

3) является читаемым и редактируемым форматом, в отличие от бинарных форматов, таких как 3DS и MAX, которые занимают первое и второе места соответственно в рейтинге по количеству моделей.

Выходными данными является растровое изображение. Из всех возможных форматов выходных данных (BMP, GIF, JPG, JPEG, PNG, PBM, PGM, PPM, XBM, XPM), которая предоставляет библиотека Qt, был выбран формат PNG, так как:

1) формат PNG является графическим, что было необходимым для создания фильма с помощью утилиты ffmpeg [7];

2) формат PNG является платформонезависимым, в отличие от формата BMP;

3) согласно [8], изображения в формате PNG более качественные, чем изображения в формате JPG по метрике типа PSNR (peak signal-to-noise ratio, пиковое отношение сигнала к шуму).

3.3. Требования к ПО

На рисунке 3.1 представлена IDEF0-диаграмма ПО, характеризующая требования к ПО.

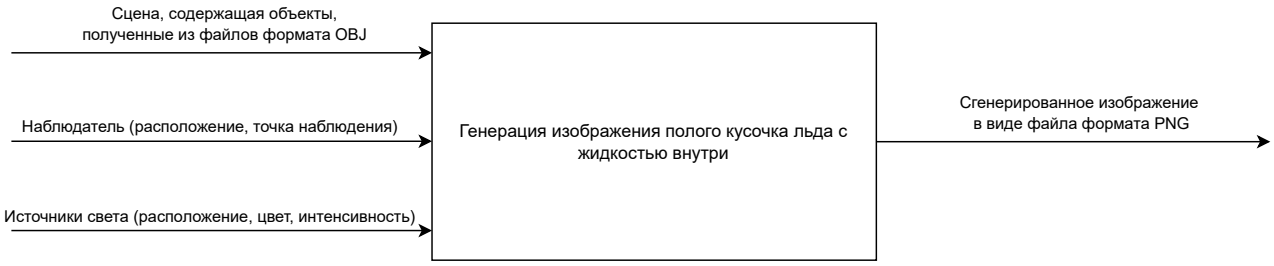


Рисунок 3.1 – IDEF0-диаграмма ПО

3.4. Реализация типов и структур данных

Разработанные в конструкторской части 2.3. типы и структуры данных были реализованы следующим образом.

1) Сцена представляет собой объект класса *Scene* с приватными полями _model типа std :: vector<std::shared_ptr<Model>> и _light типа std :: vector<std ::shared_ptr<Light>> .

2) Модель представляет собой объекта класса *Model* с приватными полями std :: vector<QVector3D> _points , std :: vector<Triangle> _faces и Material _material.

3) Материал представляет объекта класса *Material* с приватными полями _ambient, _diffuse, _specular , типа QColor, _ka, _kd, _ks, _k_refl, _k_refr, _refraction_index типа double и _k типа int

4) Камера представляет собой объект класса *Camera* с приватными полями _ic, _ij, _ik, _pos типа QVector3D и _img_width, _img_height типа int.

5) Источник света представляет собой объекта класса *Light* с приватными полями _position типа QVector3D, _color типа QColor и _intensity типа double.

3.5. Реализация алгоритмов

В листингах 3.1 – 3.3 приведена реализация алгоритма испускания луча.

Листинг 3.1 – Реализация алгоритма испускания луча (начало)

```
Color MainWindow::_cast_ray(Color& buf_color, const Ray ray, const
    int depth)
{
  Intersection intersect;

  Color color;

  if (_scene.intersect(ray, intersect) && depth <= N) {
    QVector3D reflect_dir = _reflects(-ray.get_dst(), intersect.norm
       );
    reflect_dir.normalize();
    QVector3D reflect_orig = QVector3D::dotProduct(reflect_dir,
       intersect.norm) < 0 ? intersect.point - intersect.norm * EPS
       : intersect.point + intersect.norm * EPS;
    Color reflect_color;
    if (intersect.material.get_k_refl() > 0)
      reflect_color = _cast_ray(buf_color, Ray(reflect_orig,
         reflect_dir), depth + 1);

    QVector3D refract_dir = _refract(ray.get_dst(), intersect.norm,
       intersect.material.get_refraction_index());
    refract_dir.normalize();
    QVector3D refract_orig = QVector3D::dotProduct(refract_dir,
       intersect.norm) < 0 ? intersect.point - intersect.norm * EPS
       : intersect.point + intersect.norm * EPS;
    Color refract_color;
    if (intersect.material.get_k_refr() > 0)
      refract_color = _cast_ray(buf_color, Ray(refract_orig,
         refract_dir), depth + 1);
    color.r = (intersect.material.get_ambient().r * intersect.
       material.get_ka());
    color.g = (intersect.material.get_ambient().g * intersect.
       material.get_ka());
    color.b = (intersect.material.get_ambient().b * intersect.
       material.get_ka());
```

Листинг 3.2 – Реализация алгоритма испускания луча (продолжение)

```
    for (size_t k = 0; k < _scene.get_lights().size(); k++) {
      QVector3D L = _scene.get_lights()[k]->get_position() -
          intersect.point;
      L.normalize();

      double fator_dif = QVector3D::dotProduct(L, intersect.norm);
      double light_dist = (_scene.get_lights()[k]->get_position() -
          intersect.point).length();

      QVector3D shadow_orig = fator_dif <= EPS ? intersect.point -
          intersect.norm * EPS : intersect.point + intersect.norm *
          EPS;

      Intersection tmp_intersect;
      Ray tmp_ray = Ray(shadow_orig, L);

      if (_scene.intersect(tmp_ray, tmp_intersect) && (tmp_intersect
          .point - shadow_orig).length() < light_dist && fabs(
          tmp_intersect.material.get_k_refr()) < EPS) {
        continue;
      }

      if (fator_dif <= EPS)
        fator_dif = 0.0;

      color.r = color.r + _scene.get_lights()[k]->get_color().r *
          _scene.get_lights()[k]->get_intensity() * fator_dif *
          intersect.material.get_diffuse().r * intersect.material.
          get_kd();
      color.g = color.g + _scene.get_lights()[k]->get_color().g *
          _scene.get_lights()[k]->get_intensity() * fator_dif *
          intersect.material.get_diffuse().g * intersect.material.
          get_kd();
      color.b = color.b + _scene.get_lights()[k]->get_color().b *
          _scene.get_lights()[k]->get_intensity() * fator_dif *
          intersect.material.get_diffuse().b * intersect.material.
          get_kd();
```

Листинг 3.3 – Реализация алгоритма испускания луча (окончание)

```
1         QVector3D reflexao = _reflects(L, intersect.norm);
2         double fator_esp = QVector3D::dotProduct(ray.get_vector(),
              reflexao);
3         if (fator_esp <= EPS)
4           fator_esp = 0.0;
5
6         color.r = color.r + _scene.get_lights()[k]->get_color().r *
              _scene.get_lights()[k]->get_intensity() * pow(fator_esp,
              intersect.material.get_k()) * intersect.material.
              get_specular().r * intersect.material.get_ks();
7         color.g = color.g + _scene.get_lights()[k]->get_color().g *
              _scene.get_lights()[k]->get_intensity() * pow(fator_esp,
              intersect.material.get_k()) * intersect.material.
              get_specular().g * intersect.material.get_ks();
8         color.b = color.b + _scene.get_lights()[k]->get_color().b *
              _scene.get_lights()[k]->get_intensity() * pow(fator_esp,
              intersect.material.get_k()) * intersect.material.
              get_specular().b * intersect.material.get_ks();
9
10        color.r = color.r + reflect_color.r * intersect.material.
              get_k_refl();
11        color.g = color.g + reflect_color.g * intersect.material.
              get_k_refl();
12        color.b = color.b + reflect_color.b * intersect.material.
              get_k_refl();
13
14        color.r = color.r + refract_color.r * intersect.material.
              get_k_refr();
15        color.g = color.g + refract_color.g * intersect.material.
              get_k_refr();
16        color.b = color.b + refract_color.b * intersect.material.
              get_k_refr();
17      }
18      color.normalize();
19    } else {
20      color = Color(0.07, 0.07, 0.07);
21    }
22    buf_color = color;
23    return color;
24  }
```

В листинге 3.4 приведена реализация алгоритма пересечения луча со сценой.

Листинг 3.4 – Реализация алгоритма пересечения луча со сценой

```
bool Scene::intersect(const Ray& ray, Intersection& intersect)
{
  double dist = std::numeric_limits<float>::max();
  ;
  Intersection closest;

  for (auto iter = _objects.begin(); iter != _objects.end(); iter++)
      {

    Intersection inter = (*iter)->intersection(ray);

    if (inter.dist > 0.0 && inter.dist <= dist) {
      dist = inter.dist;
      closest = inter;
    }
  }

  if (fabs(dist - std::numeric_limits<float>::max()) < EPS)
  return false;

  intersect = closest;

  return true;
}
```

В листингах 3.5 – 3.6 приведена реализация алгоритма пересечения луча с объектом.

Листинг 3.5 – Реализация алгоритма пересечения луча с объектом (начало)

```
Intersection Model::intersection(const Ray& ray) const
{
  Intersection intersectPoint;
  intersectPoint.dist = 0.0;

  if (!this->_ray_box_intersect(ray))
    return intersectPoint;

  float faceDist = std::numeric_limits<float>::max();
```

Листинг 3.6 – Реализация алгоритма пересечения луча с объектом (окончание)

```
  float currDist;
  Triangle face;

  std::vector<Triangle>::const_iterator face_it;

  for (auto it = _faces.begin(); it != _faces.end(); ++it) {
    if (this->_ray_face_intersect(ray, *it, currDist) && fabs(
       currDist) < faceDist) {
      faceDist = currDist;
      face = *it;
      face_it = it;
    }
  }

  if (faceDist == std::numeric_limits<float>::max())
    return intersectPoint;

  intersectPoint.norm = this->get_normal(face, ray);
  intersectPoint.dist = faceDist;
  intersectPoint.point = ray.get_src() + ray.get_dst() * faceDist;
  intersectPoint.material = this->get_material();

  return intersectPoint;
}
```

В листингах 3.7 – 3.8 приведена реализация алгоритма пересечения луча с полигоном.

Листинг 3.7 – Реализация алгоритма пересечения луча с полигоном (начало)

```
bool Model::_ray_face_intersect(const Ray& ray, const Triangle& face
   , float& ray_tvalue) const
{
  QVector3D edge1 = get_point(face.verts[1]) - get_point(face.verts
     [0]);
  QVector3D edge2 = get_point(face.verts[2]) - get_point(face.verts
     [0]);

  QVector3D pvec = QVector3D::crossProduct(ray.get_dst(), edge2);
  float det = QVector3D::dotProduct(edge1, pvec);
```

Листинг 3.8 – Реализация алгоритма пересечения луча с полигоном (окончание)

```
    if (fabs(det) < EPS)
      return false;

    auto inv_det = 1. / det;

    QVector3D tvec = ray.get_src() - get_point(face.verts[0]);

    float bary_u = QVector3D::dotProduct(tvec, pvec) * inv_det;
    if (bary_u < 0.0 || bary_u > 1.0)
      return false;

    QVector3D qvec = QVector3D::crossProduct(tvec, edge1);
    float bary_v = QVector3D::dotProduct(ray.get_dst(), qvec) *
       inv_det;

    if (bary_v < 0.0 || bary_u + bary_v > 1.0)
      return false;

    ray_tvalue = static_cast<float>(QVector3D::dotProduct(edge2, qvec)
        * inv_det);

    return ray_tvalue > EPS;
}
```

3.6. Описание интерфейса программы

На рисунках 3.2 – 3.4 представлен интерфейс программного продукта. На рисунке 3.2 изображён режим работы с геометрией объектов, взаимодействуя с которым можно добавить объект на сцену, удалить объект со сцены, переместить, масштабировать и повернуть выбранный из выпадающего списка объект. На рисунке 3.3 представлен режим работы с источниками света. С помощью данного интерфейса можно добавить источник света на сцену, удалить источник света со сцены, изменить его свойства (расположение, интенсивность). На рисунке 3.4 проиллюстрирован режим работы со свойствами материалов объектов, с помощью которого можно устанавливать параметры

фонового освещения, диффузного освещения, зеркального освещения, коэффициенты отражения, преломления материалов объектов.

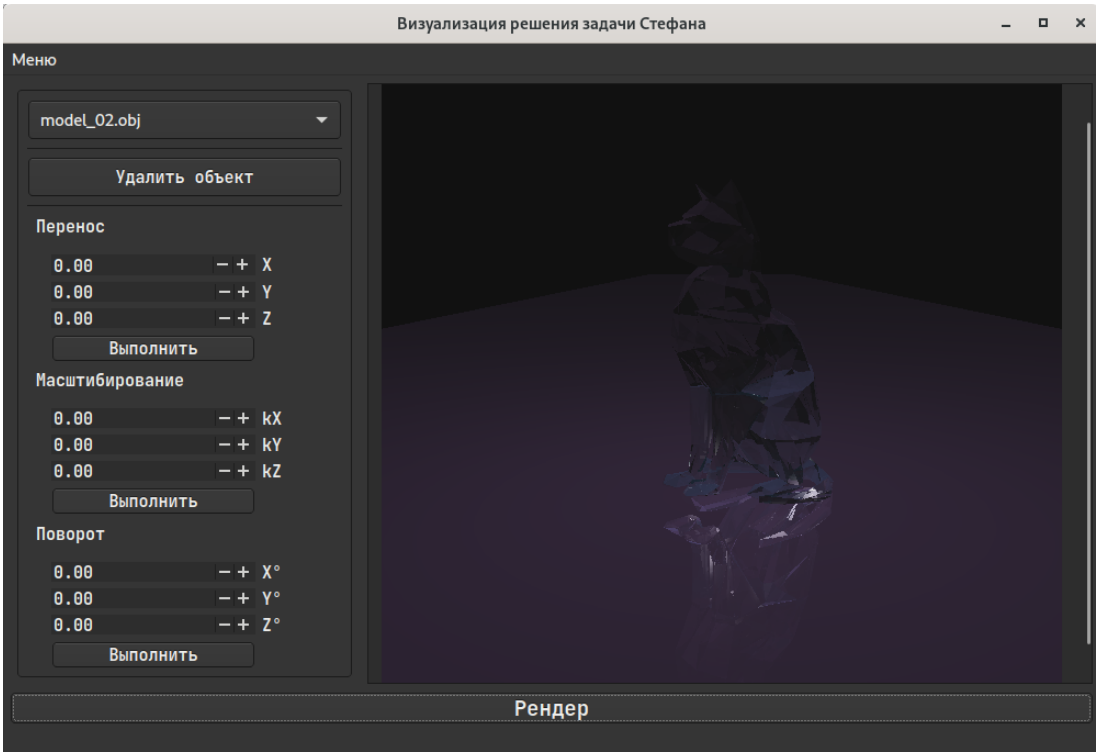


Рисунок 3.2 – Интерфейс ПО (режим работы с геометрией объектов)

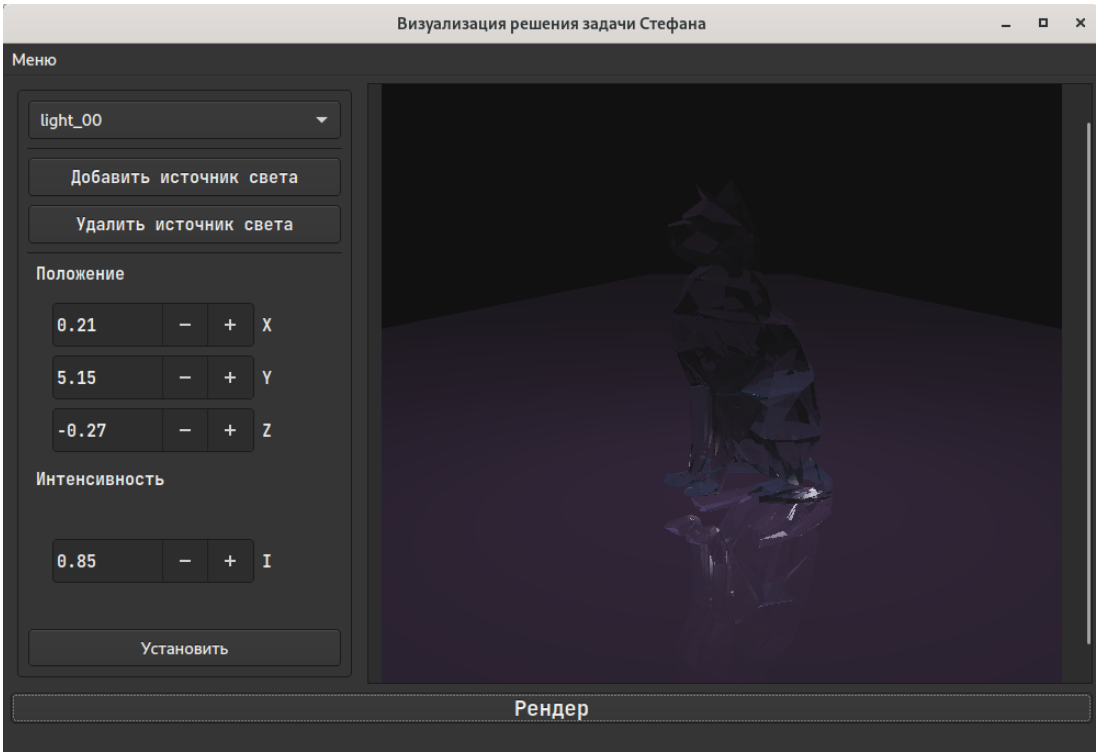


Рисунок 3.3 – Интерфейс ПО (режим работы с источниками света)

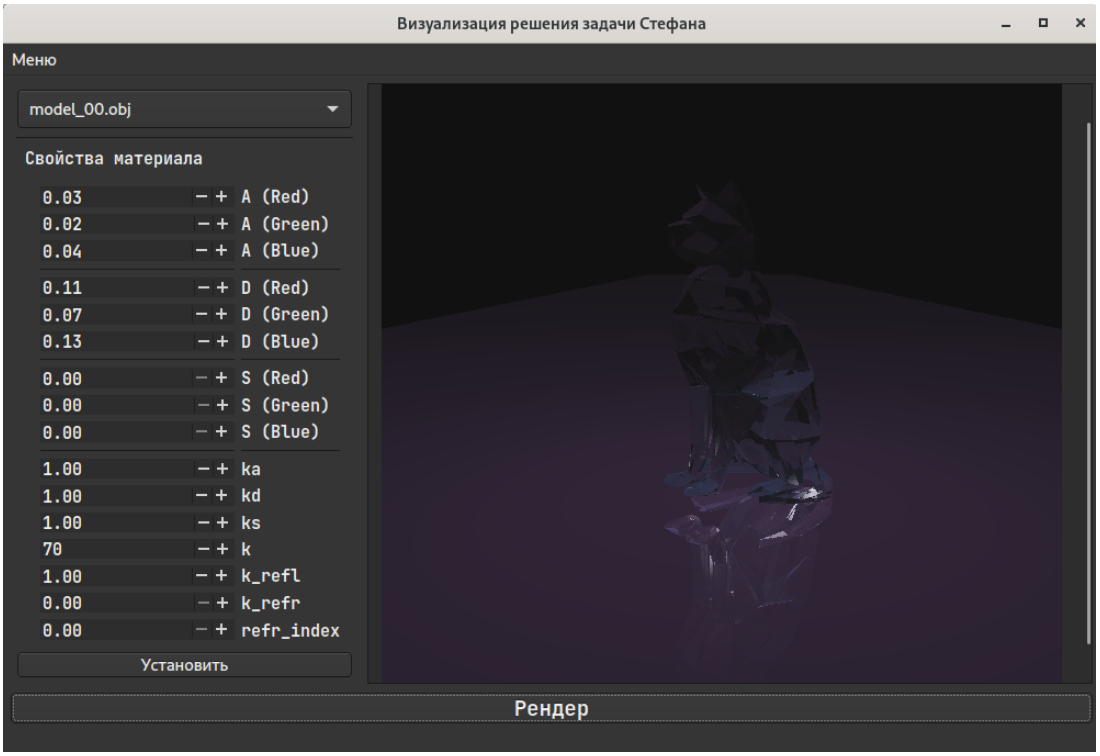
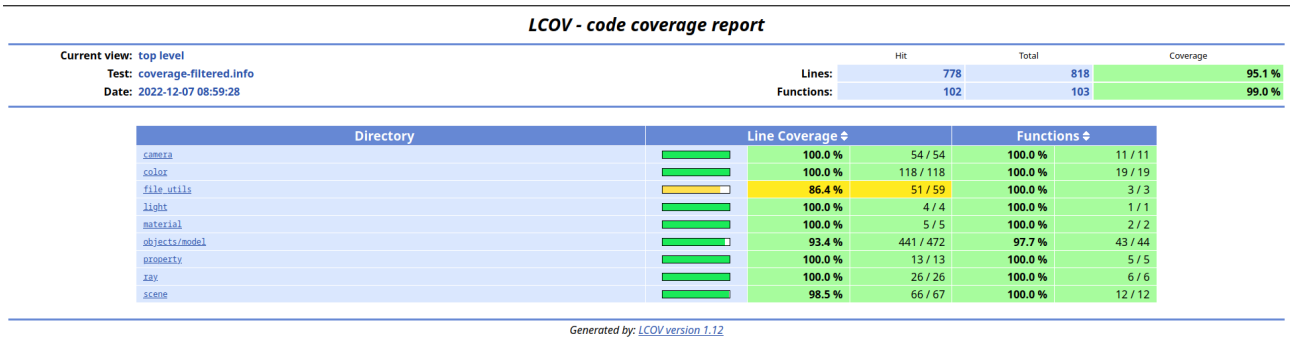


Рисунок 3.4 – Интерфейс ПО (режим работы со свойствами материалов объектов)

3.7. Тестирование ПО

Было проведено модульное тестирование всех неинтерфейсных модулей программы. Для тестирования был выбран фреймворк Qt Test [9]. Результаты тестирования, обработанные утилитой lcov, представлены на рисунке 3.5.



LCOV - code coverage report

		Hit	Total	Coverage
Current view: top level				
Test: coverage-filtered.info	Lines:	778	818	95.1 %
Date: 2022-12-07 08:59:28	Functions:	102	103	99.0 %

Directory	Line Coverage		Functions	
camera	100.0 %	54 / 54	100.0 %	11 / 11
color	100.0 %	118 / 118	100.0 %	19 / 19
file_utils	86.4 %	51 / 59	100.0 %	3 / 3
light	100.0 %	4 / 4	100.0 %	1 / 1
material	100.0 %	5 / 5	100.0 %	2 / 2
objects/model	93.4 %	441 / 472	97.7 %	43 / 44
property	100.0 %	13 / 13	100.0 %	5 / 5
ray	100.0 %	26 / 26	100.0 %	6 / 6
scene	98.5 %	66 / 67	100.0 %	12 / 12

Generated by: LCOV version 1.12

Рисунок 3.5 – Результаты модульного тестирования

3.8. Выводы из технологической части

В данном разделе были обоснованы выбор средств реализации, формат входных и выходных данных, разработаны типы и структуры данных и алгоритмы, рассмотрен интерфейс разработанного ПО и проведено тестирование разработанного продукта.

4. Исследовательская часть

В данном разделе приведены результаты работы программного обеспечения и проведено измерение и сравнение времени работы однопоточной и многопоточной реализаций алгоритма обратной трассировки лучей.

4.1. Технические характеристики

Технические характеристики устройства, на котором выполнялось исследование, следующие.

— Операционная система Linux Mint 21 [10].

— Оперативная память: 8 ГБ.

— Процессор: Intel(R) Core(TM) i3-10100F CPU @ 3.60 ГГц [11].

4.2. Результаты работы ПО

На рисунках 4.1 – 4.4 представлены изображения, полученные с помощью разработанного ПО.

Рисунок 4.1 – Изображение №1, полученное с помощью разработанного ПО

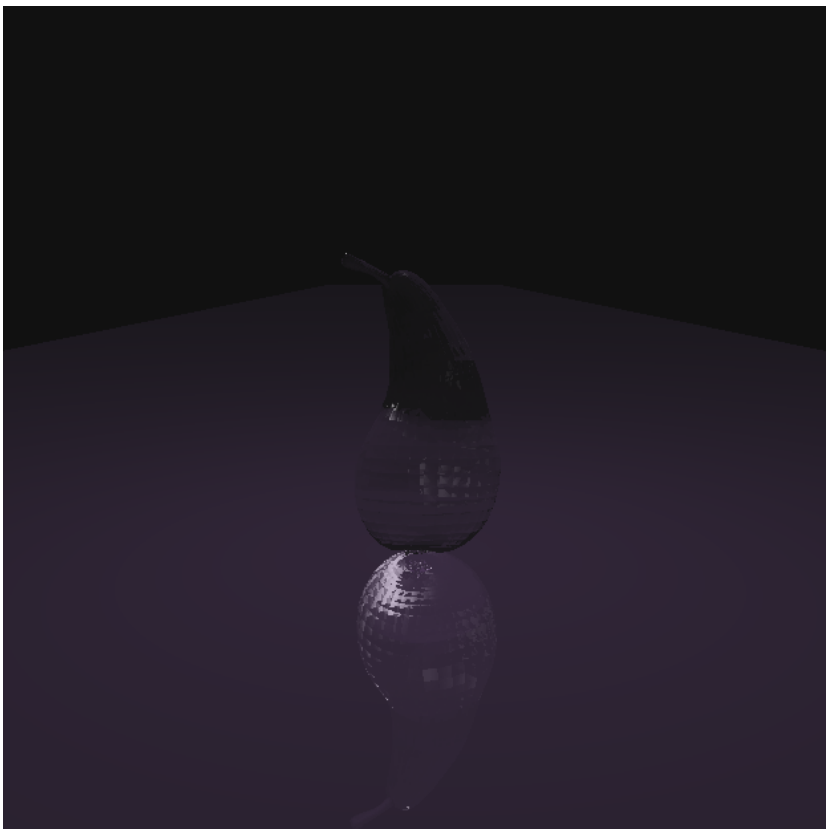


Рисунок 4.2 – Изображение №2, полученное с помощью разработанного ПО

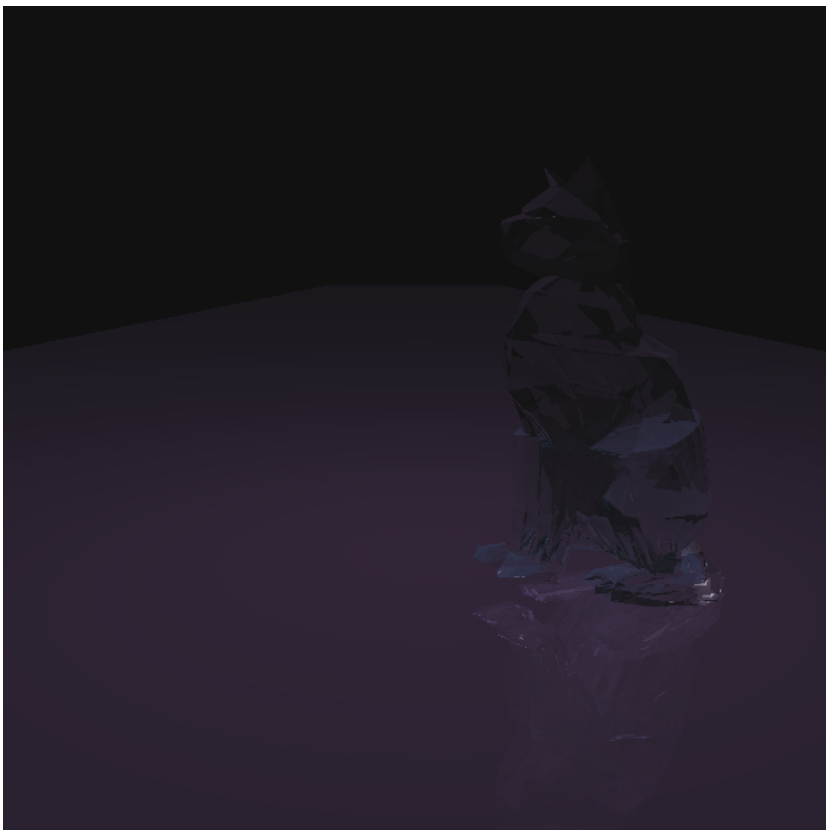


Рисунок 4.3 – Изображение №3, полученное с помощью разработанного ПО

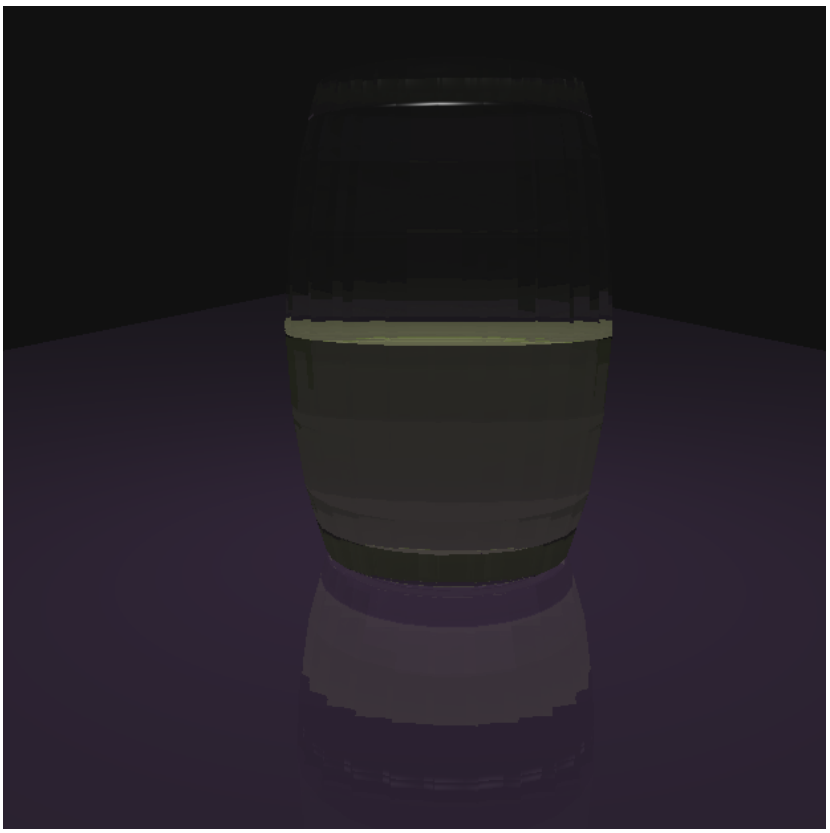


Рисунок 4.4 – Изображение №4, полученное с помощью разработанного ПО

4.3. Измерение реального времени выполнения реализаций алгоритма

Время работы алгоритма обратной трассировки лучей было замерено с помощью класса *std::chrono::system_clock* [5], который представляет реальное время. В исследуемых изображениях растр имеет одинаковое количество пикселей в горизонтальном и вертикальном измерениях. Для таблицы введено обозначение: S — размер изображения (в пикселях).

Результаты замеров приведены в таблице 4.1.

Таблица 4.1 – Таблица времени выполнения алгоритма в мкс.

S	Не распараллеленный	Количество потоков				
		1	2	4	8	16
128	273,311	556,595	339,305	278,957	279,667	292,878
256	2,023,442	3,103,714	1,829,316	1,395,264	1,397,786	1,384,682
352	3,739,262	5,731,390	3,464,986	2,533,300	2,478,940	2,608,794
448	6,302,434	9,693,502	5,783,250	4,178,672	4,100,138	4,188,324
512	9,245,362	13,801,280	8,086,786	5,764,098	5,463,540	6,166,600
640	14,072,760	20,832,840	12,204,240	8,941,954	8,551,252	9,452,858

На рисунке 4.5 приведен график, отражающий зависимость времени работы алгоритма обратной трассировки лучей от размера изображения при различном количестве потоков.

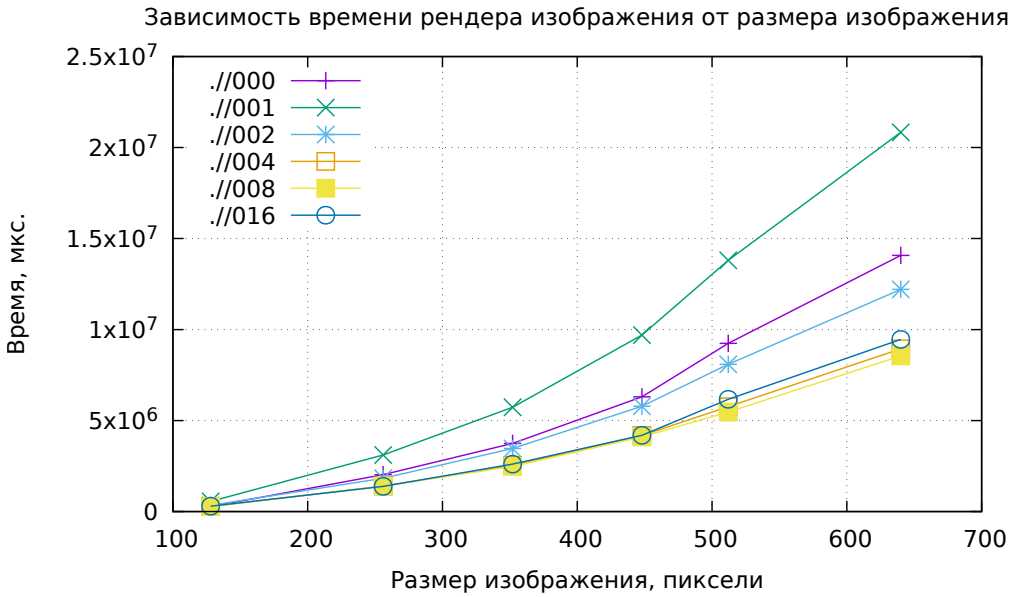


Рисунок 4.5 – Зависимость времени работы алгоритма обратной трассировки лучей от размера изображения при различном количестве потоков

4.4. Выводы из исследовательской части

Наилучшее время выполнения распараллеленный алгоритм показал при 8 потоках, что соответствует количеству логических процессоров компьютера, на котором проводилось измерение. На изображениях размером 640 на 640 пикселей, параллельный алгоритм с 8 потоками работает в ≈ 2.44 раза быстрее однопоточной реализации. При количестве потоков, большем восьми, время выполнения увеличивается по сравнению с реализацией с восемью потоками. Таким образом, рекомендуется использовать число потоков, равное числу логических процессоров. Не распараллеленная реализация работает быстрее однопоточной, поскольку в однопоточной уходит время на создание потока.

Заключение

В ходе выполнения курсовой работы было разработано программное обеспечение, которое позволяет получить реалистичное изображение полого кусочка льда с жидкостью внутри. В процессе выполения данной работы были выполнены все задачи:

1) проведён анализ алгоритмов построения реалистичных изображений;
2) разработан метод построения реалистичного изображения полого кусочка льда с жидкостью внутри;
3) реализован метод построения реалистичного изображения полого кусочка льда с жидкостью внутри;
4) исследована зависимость времени выполнения однопоточной и многопоточной реализаций метода построения реалистичного изображения полого кусочка льда с жидкостью внутри от размера изображения.

Список использованных источников

1. Боресков А. В., Шикин Е. В. «Компьютерная графика» / Шикин Е. В. Боресков А. В. — Издательство Юрайт (Москва), 2017. — С. 219.

2. Д. Роджерс, Дж. Адамс. «Математические основы машинной графики» / Дж. Адамс Д. Роджерс. — Москва-Мир, 2001. — С. 603.

3. Fast, Minimum Storage Ray/Triangle Intersection [Электронный ресурс]. — Режим доступа: `https://cadxfem.org/inf/Fast%20MinimumStorage%20RayTriangle%20Intersection.pdf` (дата обращения: 17.10.2022).

4. Technical Report on C++ Performance [Электронный ресурс]. — Режим доступа: `https://www.open-std.org/Jtc1/SC22/wg21/docs/papers/2004/n1666.pdf` (дата обращения: 17.10.2022).

5. C++ Draft International Standard [Электронный ресурс]. — Режим доступа: `https://isocpp.org/files/papers/N4860.pdf` (дата обращения: 17.10.2022).

6. An Overview of 3D Data Content, File Formats and Viewers [Электронный ресурс]. — Режим доступа: `http://isda.ncsa.illinois.edu/peter/publications/techreports/2008/NCSA-ISDA-2008-002.pdf` (дата обращения: 17.10.2022).

7. Документация по утилите ffmpeg [Электронный ресурс]. — Режим доступа: `https://ffmpeg.org/documentation.html` (дата обращения: 17.10.2022).

8. Performance Evaluation of Secrete Image Steganography Techniques Using Least Significant Bit (LSB) Method [Электронный ресурс]. — Режим доступа: `http://www.ijcstjournal.org/volume-6/issue-2/IJCST-V6I2P30.pdf` (дата обращения: 17.10.2022).

9. Документация по библиотеке Qt Test [Электронный ресурс]. — Режим доступа: `https://doc.qt.io/qt-6/qttest-index.html` (дата обращения: 17.10.2022).

10. Linux Mint 21 overview [Электронный ресурс]. — Режим доступа: https://linuxmint.com (дата обращения: 17.10.2022).

11. Intel(R) Core(TM) i3-10100F [Электронный ресурс]. — Режим доступа: https://ark.intel.com/content/www/ru/ru/ark/products/203473/intel-core-i310100f-processor-6m-cache-up-to-4-30-ghz.html (дата обращения: 17.10.2022).

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

Визуализация решения задачи Стефана

Студент: Глотов Илья Анатольевич ИУ7-52Б

Руководитель курсового проекта: Кострицкий Александр Сергеевич

Цель и задачи

Целью курсовой работы является разработка программного обеспечения, позволяющего получить реалистичное изображение полого кусочка льда с жидкостью внутри.

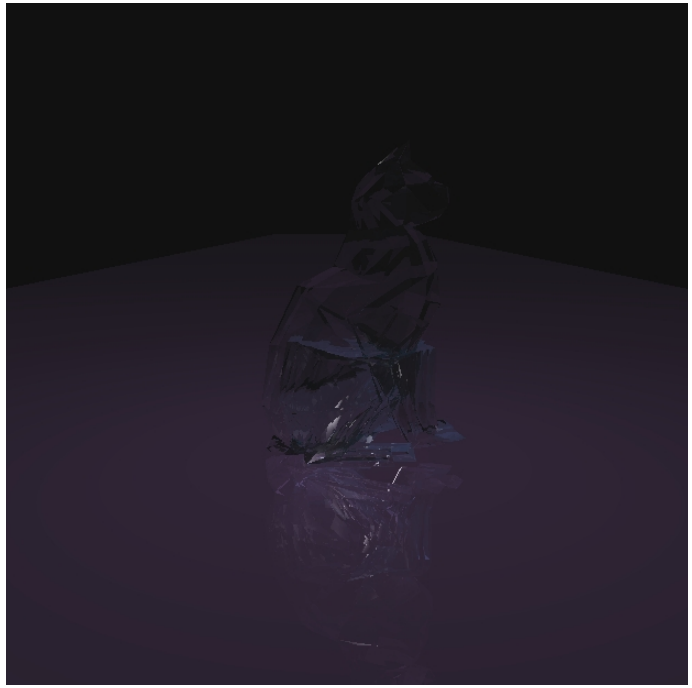
Задачи:

- провести анализ алгоритмов построения реалистичных изображений;
- разработать метод построения реалистичного изображения полого кусочка льда с жидкостью внутри;
- реализовать метод построения реалистичного изображения полого кусочка льда с жидкостью внутри;
- исследовать зависимость времени выполнения однопоточной и многопоточной реализаций метода построения реалистичного изображения полого кусочка льда с жидкостью внутри от размера изображения.

Описание и формализация объектов сцены

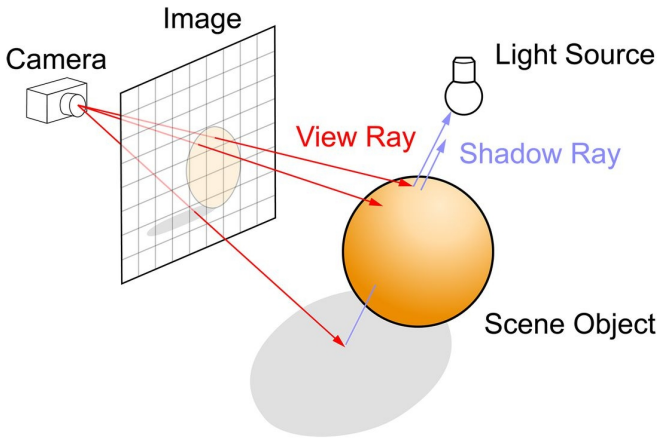
Объекты сцены:

- Геометрический объект (представляется в виде полигональной сетки).
- Источник света (задаётся расположением в пространстве, цветом излучения и интенсивностью излучения).
- Камера (задаётся расположением в пространстве и направлением взгляда).



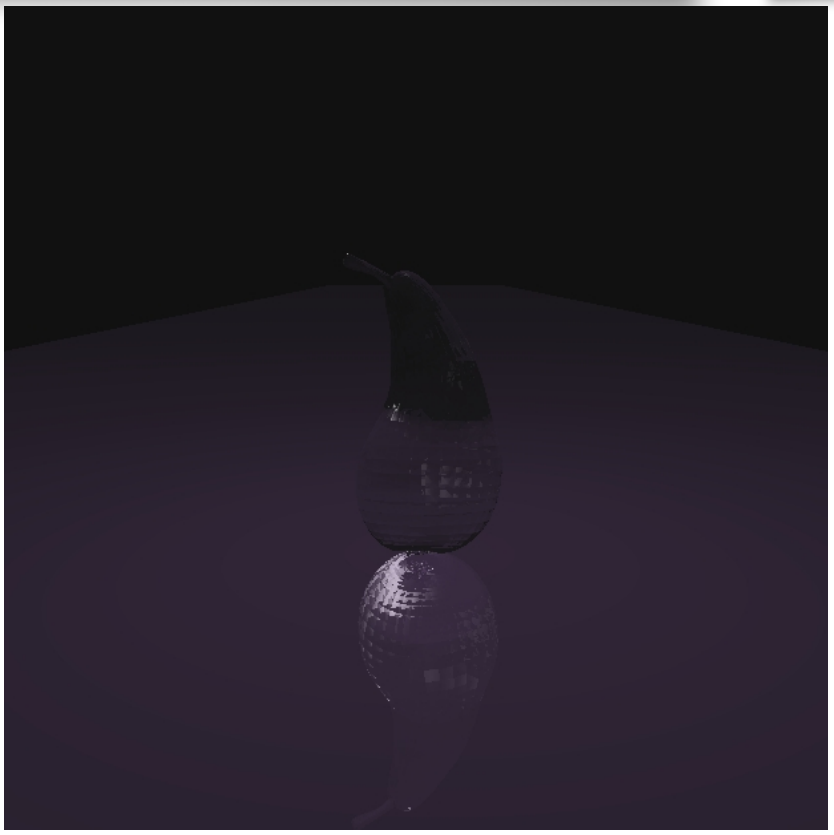
Алгоритмы трёхмерной графики

В качестве алгоритма удаления невидимых рёбер и поверхностей был выбран алгоритм обратной трассировки лучей с глобальной моделью освещения из-за высокой реалистичности синтезируемого изображения и возможности визуализации зеркальных и прозрачных поверхностей.

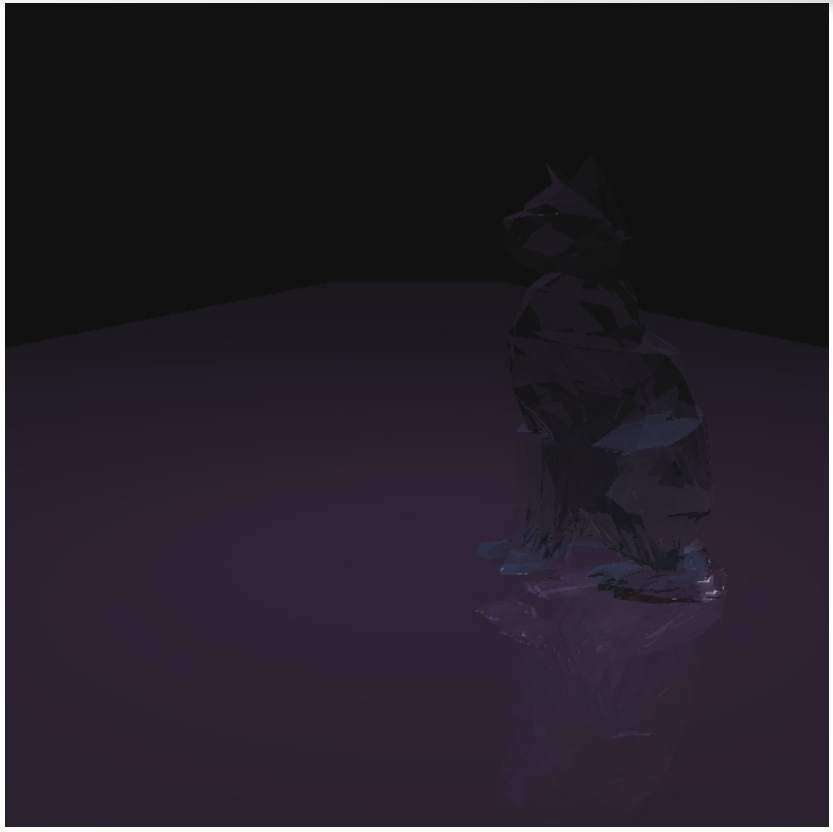


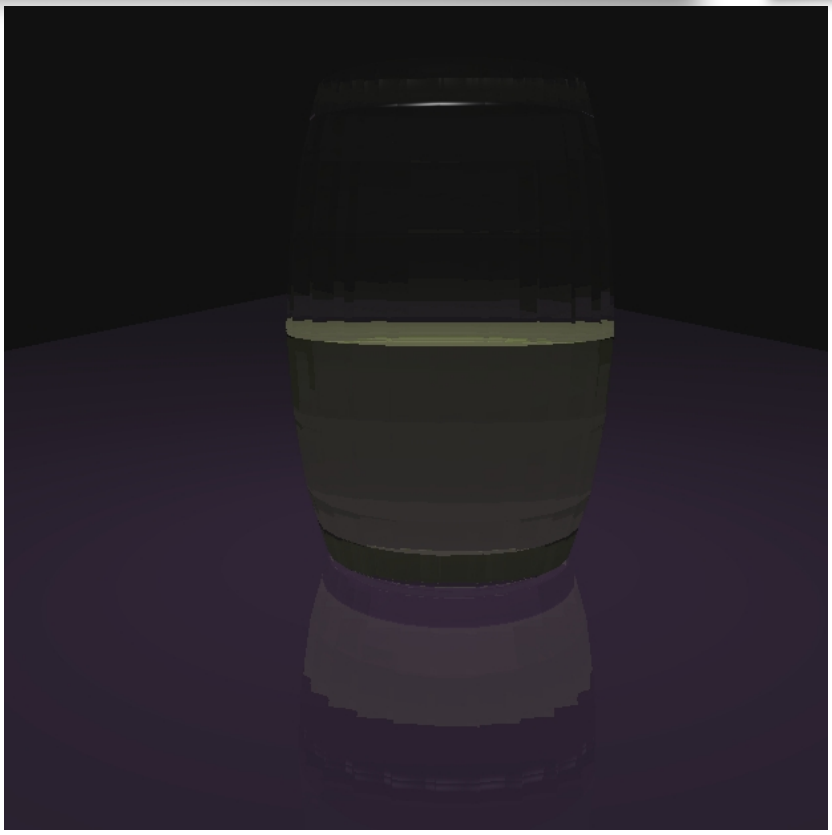
Требования к ПО

Примеры работы программного обеспечения



Примеры работы программного обеспечения



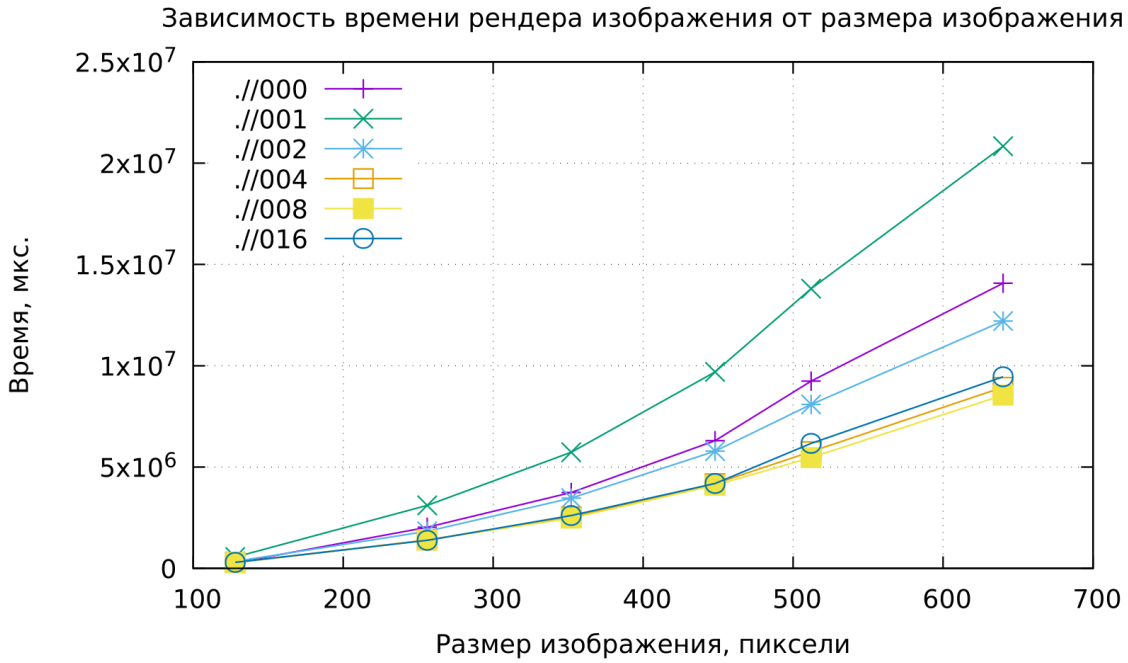


Проведение исследования

Цель: определение зависимости времени генерации изображения от размера изображения при различном количестве потоков.

Размеры изображения	Время выполнения без использования распараллеливания, мкс	Время выполнения с использованием распараллеливания, мкс				
		1	2	4	8	16
128x128	273,311	556,595	339,305	278,957	279,667	292,878
256x256	2,023,442	3,103,714	1,829,316	1,395,264	1,397,786	1,384,682
352x352	3,739,252	5,731,390	3,464,986	2,533,300	2,478,940	2,608,794
448x448	6,302,434	9,693,502	5,783,250	4,178,672	4,100,138	4,188,324
512x512	9,245,352	13,801,280	8,086,786	5,764,098	5,463,540	6,166,600
640x640	14,072,760	20,832,840	12,204,240	8,941,954	8,551,252	9,542,858

Проведение исследования



- Лучшее время распараллеленный алгоритм показал при 8 потоках, что соответствует количеству логических процессоров компьютера, на котором проводилось измерение
- Не распараллеленная реализация работает быстрее однопоточной, поскольку в однопоточной тратится дополнительное время на создание потока

Заключение

В ходе выполнения курсового проекта были выполнены все задачи:

- проведён анализ алгоритмов построения реалистичных изображений;
- разработан метод построения реалистичного изображения полого кусочка льда с жидкостью внутри;
- реализован метод построения реалистичного изображения полого кусочка льда с жидкостью внутри;
- исследована зависимость времени выполнения однопоточной и многопоточной реализаций метода построения реалистичного изображения полого кусочка льда с жидкостью внутри от размера изображения.

Поставленная цель была достигнута.