

2024 г.

РЕФЕРАТ

Расчетно-пояснительная записка 43 с., 18 рис., 2 таблиц, 17 источников, 2 приложения.

Цель работы — разработка программного обеспечения для визуализации шахматных фигур на шахматной доске.

В рамках курсовой работы был проведен анализ способов представления поверхностей трехмерных моделей и алгоритмов построения реалистичного изображения; была разработана функциональная модель программного обеспечения и описана его структура; было разработано программное обеспечение для визуализации шахматных фигур на шахматной доске.

Было проведено исследование, целью которого являлся сравнительный анализ зависимости времени генерации кадра от числа полигонов на сцене для варьируемого числа рабочих потоков. По результатам измерений были сделаны выводы о том, что при количестве полигонов на сцене меньшем 200, эффективнее по времени использовать алгоритм обратной трассировки лучей с одним потоком, а при количестве полигонов большем 200 эффективнее использовать 32 дополнительных рабочих потока.

СОДЕРЖАНИЕ

РЕФЕРАТ	3
ВВЕДЕНИЕ	6
1 Аналитическая часть	7
1.1 Формализация задачи	7
1.1.1 Функциональные требования к разрабатываемому программному обеспечению	7
1.1.2 Формализация объектов сцены	7
1.2 Анализ способов представления поверхностей трехмерных моделей	9
1.3 Анализ алгоритмов удаления невидимых линий и поверхностей	10
1.3.1 Алгоритм Робертса	10
1.3.2 Алгоритм, использующий z-буфер	11
1.3.3 Алгоритм обратной трассировки лучей	11
1.4 Анализ модели освещения	13
1.4.1 Локальная модель освещения	13
1.4.2 Глобальная модель освещения	14
2 Конструкторская часть	16
2.1 Математические основы алгоритма обратной трассировки лучей	16
2.1.1 Определение пересечения луча с полигоном	16
2.1.2 Определение вектора отражения	17
2.2 Функциональная модель программного обеспечения	18
2.3 Описание алгоритма определения цвета пикселя, методом обратной трассировки лучей	19
2.4 Структура разрабатываемого программного обеспечения . . .	21
3 Технологическая часть	28
3.1 Средства реализации	28
3.2 Реализации алгоритма обратной трассировки лучей	29
3.3 Описание процесса сборки приложения	30
3.4 Описание интерфейса приложения	31

3.5	Тестирование	33
3.5.1	Функциональное тестирование	33
3.5.2	Модульное тестирование	34
3.5.3	Результаты тестирования	34
4	Исследовательская часть	35
4.1	Технические характеристики	35
4.2	Цель исследования	35
4.3	Описание исследования	35
4.4	Результат исследования	36
4.5	Вывод из исследовательской части	37
	ЗАКЛЮЧЕНИЕ	38
	СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ	39
	Приложение А	41
	Приложение Б	43

ВВЕДЕНИЕ

Целью данной работы является разработка программного обеспечения для визуализации шахматных фигур на шахматной доске.

Для достижения поставленной цели требуется решить следующие задачи:

- 1) формализовать задачу;
- 2) выбрать алгоритм построения реалистичного изображения;
- 3) разработать функциональную модель программного обеспечения;
- 4) выбрать средства реализации и реализовать программное обеспечение для визуализации сцены;
- 5) исследовать зависимость времени генерации кадра от числа полигонов на сцене для варьируемого числа рабочих потоков.

1 Аналитическая часть

В данном разделе проводится формализация объектов сцены, рассматриваются способы представления поверхностей трехмерных моделей и методы создания трёхмерного реалистичного изображения.

1.1 Формализация задачи

1.1.1 Функциональные требования к разрабатываемому программному обеспечению

Программное обеспечение должно предоставлять возможность создавать кадр с изображением сцены, состоящей из моделей шахматных фигур и шахматной доски. Модели шахматных фигур могут находиться только в центре клеток на поверхности доски, должны вращаться вокруг своей оси и иметь возможность поменять позицию. Материал поверхности шахматной доски может быть матовым или глянцевым. Источник света на сцене должен быть один и задаваться внутри программы в фиксированной точке пространства вне поля зрения пользователя.

На рисунке 1.1 представлена функциональная модель программы в нотации IDEF0, характеризующая требования к программному обеспечению.

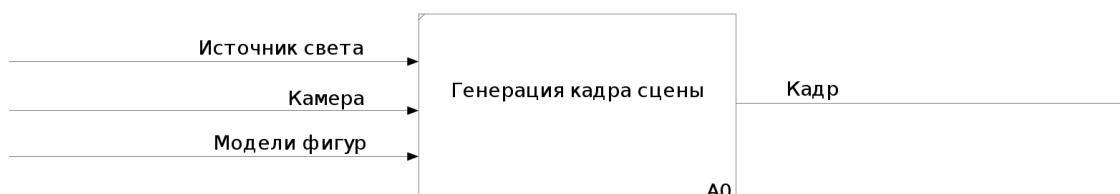


Рисунок 1.1 – Функциональная модель программы в нотации IDEF0

1.1.2 Формализация объектов сцены

Сцена состоит из следующих объектов:

- шахматная доска 8x8 клеток – основание шахматной доски цвета дерева, клетки – черного и белого цвета;
- набор шахматных фигуры двух цветов: пешка, ладья, конь, слон, ферзь, король;

- точечный источник света – задается положением в пространстве и интенсивностью излучения;
- камера – задается положением в пространстве и вектором взгляда.

Форма и размер шахматных фигур и шахматной доски соответствует стандарту шахматного оборудования и игровых площадок, предназначенных для проведения турниров ФИДЕ [1].

1.2 Анализ способов представления поверхностей трехмерных моделей

Поверхность трехмерной модели можно задать несколькими способами [2]:

— **Полигональная сетка.** В данном случае поверхность представляется как совокупность связанных между собой плоских многоугольников. Большинство объектов, не имеющих изгибов, например, таких как шахматная доска, можно точно описать полигональной сеткой. Этот способ также применяется для представления объектов, ограниченных криволинейными поверхностями, однако в таком случае объект будет описан достаточно приблизительно.

— **Аналитический способ.** Поверхность, заданная таким способом описывается функцией зависимости координат от некоторого параметра. Главным достоинством данного метода является высокая точность описания поверхности, которая нужна в большинстве вычислительных программ, однако из-за необходимости проведения большого количества математических вычислений при визуализации данных поверхностей, время их отрисовки будет значительно больше времени отрисовки поверхностей заданных множеством полигонов.

Вывод

Для решения задачи визуализации шахматной доски и шахматных фигур нет большой необходимости в точности представления поверхностей. Именно поэтому для уменьшения времени отрисовки сцены был выбран способ задания поверхностей моделей полигональной сеткой.

1.3 Анализ алгоритмов удаления невидимых линий и поверхностей

Для создания реалистичного изображения необходимо учитывать такие факторы как невидимые линии и поверхности, тени, освещение, свойства материалов объекта (способность отражать и преломлять свет).

1.3.1 Алгоритм Робертса

Алгоритм Робертса решает задачу удаления невидимых линий и работает в объектном пространстве исключительно с выпуклыми телами, если тело является не выпуклым, то его нужно разбить на выпуклые составляющие [3].

Этапы алгоритма:

- 1) **Подготовка исходных данных.** В данном алгоритме выпуклое многогранное тело представляется набором пересекающихся плоскостей. Формируется матрица тела V , где каждый столбец содержит коэффициенты уравнения плоскости грани $ax + by + cz + d = 0$:

$$V = \begin{pmatrix} a_1 & a_2 & \dots & a_n \\ b_1 & b_2 & \dots & b_n \\ c_1 & c_2 & \dots & c_n \\ d_1 & d_2 & \dots & d_n \end{pmatrix}. \quad (1.1)$$

- 2) **Удаление ребер, экранируемых самим телом.** Используется вектор взгляда $E = (0, 0, -1, 0)$ для определения невидимых граней. При умножении вектора E на матрицу тела V отрицательные компоненты полученного вектора будут соответствовать невидимым граням;
- 3) **Удаление невидимых ребер, экранируемых другими телами.** Для определения невидимых точек ребра строится луч, соединяющий точку наблюдения с точкой на ребре. Точка невидима, если луч на своём пути встречает в качестве преграды тело, т.е. проходит через него;

Преимущества алгоритма Робертса:

— Высокая точность благодаря работе в объектном пространстве;

Недостатки алгоритма Робертса:

- Не работает с невыпуклыми телами;
- Невозможность визуализации отражающих поверхностей;

1.3.2 Алгоритм, использующий z-буфер

Алгоритм Z-буфера работает в пространстве изображений и использует два буфера: буфер кадра для хранения цвета каждого пикселя и Z-буфер для хранения глубины каждого пикселя [4].

Этапы алгоритма:

- 1) Инициализация Z-буфера минимально возможными значениями и буфера кадра значениями пикселя, описывающими фон;
- 2) Преобразование каждой проекции грани многоугольников в растровую форму;
- 3) Вычисление для каждого пикселя с координатами (x, y) , его глубины $Z(x, y)$.
- 4) Сравнение глубины $Z(x, y)$ новых пикселей с текущими в Z-буфере и обновление буфера кадра при необходимости;

Преимущества алгоритма Z-буфера:

- Простота алгоритма и используемого в нем набора операций;

Недостатки алгоритма Z-буфера:

- Большой объём требуемой памяти;
- Невозможность визуализации отражающих поверхностей;

1.3.3 Алгоритм обратной трассировки лучей

Алгоритм обратной трассировки лучей заключается в отслеживании траектории лучей, которые исходят из точки наблюдателя и проходят через центр пикселя раstra в направлении сцены. [5].

Этапы алгоритма:

- 1) Преобразование сцены в пространство изображения, т. е. область видимости наблюдателя разбивается на пиксели.

- 2) Испускание лучей от наблюдателя через пиксели раstra к сцене.
- 3) Определение ближайшего пересечения лучей с объектами сцены.
- 4) Определение находится ли точка пересечения в тени путем испускания луча из этой точки в направлении источника света. И если луч пересекает какие-либо объекты сцены, то точка находится в тени.
- 5) Рекурсивное отражение и/или преломление лучей при наличии отражающих или прозрачных материалов.
- 6) Учёт теней путём проверки видимости световых источников из точки пересечения.

Преимущества алгоритма обратной трассировки лучей:

- Высокая реалистичность синтезируемого изображения.
- Учет теней и эффектов отражения и преломления.

Недостатки алгоритма обратной трассировки лучей:

- Увеличенное время выполнения из-за рекурсивных вычислений.

Вывод

В качестве алгоритма удаления невидимых линий и поверхностей был выбран алгоритм обратной трассировки лучей, так как с его помощью возможно реализовать эффект отражения, необходимый для решения поставленной задачи. А также данный алгоритм не требует дополнительной реализации алгоритмов закраски и построения теней.

1.4 Анализ модели освещения

Модель освещения определяет цвет поверхности объекта, отображаемого на экране и бывает двух видов: локальная и глобальная [3].

1.4.1 Локальная модель освещения

Локальная модель освещения состоит из трех компонент [6]:

1) **Фоновое освещение.** Данная составляющая позволяет учитывать свет постоянной яркости, созданный многочисленными отражениями от различных поверхностей. Такой свет практически всегда присутствует в реальной обстановке. Интенсивность рассеянного света можно рассчитать по формуле (1.2)

$$I_a = k_a \cdot i_a \quad (1.2)$$

где I_a — интенсивность рассеянного света, k_a — коэффициент фонового освещения, i_a — интенсивность источника рассеянного света.

2) **Диффузное отражение.** Идеальное диффузное отражение описывается законом Ламберта, согласно которому падающий свет рассеивается во все стороны с одинаковой интенсивностью. Интенсивность диффузного отражения света можно рассчитать по формуле (1.3)

$$I_d = k_d \cdot I_l \cdot \frac{\cos(\vec{L}, \vec{N})}{r + k} \quad (1.3)$$

где I_d — интенсивность диффузного отражения света, k_d — коэффициент диффузного отражения, I_l — интенсивность точечного источника света, \vec{L} — вектор направленный на источник света, \vec{N} — вектор нормали к поверхности, r — расстояние от центра проекции до поверхности k — произвольная постоянная.

3) **Зеркальное отражение** Направленное отражение которому на блестящих объектах образуются блики. Наблюдатель видит зеркально отраженный свет только в том случае, когда угол отражения от идеальной отражающей поверхности равен углу падения. Интенсивность зеркального отражения света можно рассчитать по формуле (1.4)

$$I_s = k_s \cdot I_l \cdot \frac{\cos^n(\vec{R}, \vec{S})}{r + k}, \quad (1.4)$$

где I_s — интенсивность зеркального отражения света, k_s — коэффициент зеркального отражения, I_l — интенсивность точечного источника света, \vec{R} — вектор отраженного луча, \vec{S} — вектор направленный на наблюдателя, n — степень, аппроксимирующая пространственное распределение зеркально отраженного света, r — расстояние от центра проекции до поверхности k — произвольная постоянная.

$$I = k_a \cdot I_a + k_d \cdot I_l \cdot \frac{\cos(\vec{L}, \vec{N})}{r + k} + k_s \cdot I_l \cdot \frac{\cos^n(\vec{R}, \vec{S})}{r + k} + k_s \cdot I_r + k_t \cdot I_t \quad (1.5)$$

1.4.2 Глобальная модель освещения

Глобальная модель освещения дополняет локальную модель и позволяет учитывать положение объектов сцены относительно друг друга, благодаря чему появляется возможность визуализировать эффекты отражения света от других объектов и пропускания света сквозь прозрачные объекты.

Глобальная модель освещения складывается из непосредственной освещенности точки источником света, которая рассчитывается по локальной модели освещения, и вторичной освещенности, которая в свою очередь состоит из интенсивности света отраженного и преломленного луча [7]. Интенсивность света в точке по глобальной модели освещения рассчитывается формулой (1.6):

$$I = I_a + I_d + I_s + k_s \cdot I_r + k_t \cdot I_t, \quad (1.6)$$

где I_a — интенсивность рассеянного света (1.2), I_d — интенсивность диффузного отражения света (1.3), I_s — интенсивность зеркального отражения света (1.4), k_s — коэффициент зеркального отражения, k_t — коэффициент пропускания, I_r — интенсивности света отраженного луча, I_t — интенсивности света преломленного луча.

Вывод

Так как для решения поставленной задачи необходимо реализовать отражающую поверхность шахматной доски, в работе использовалась глобальная модель освещения, которая являлась составной частью алгоритма обратной трассировки лучей.

Вывод из аналитической части

В данном разделе была формализована поставленная задача, были рассмотрены способы представления поверхностей трехмерных моделей и методы создания трёхмерного реалистичного изображения. В результате был выбран метод представления трехмерных поверхностей полигональной сеткой и для визуализации трехмерной сцены был выбран алгоритм обратной трассировки лучей, который включает в себя глобальную модель освещения.

2 Конструкторская часть

В данном разделе представлены математические основы и схема алгоритма обратной трассировки лучей, функциональная модель и структура программного обеспечения.

2.1 Математические основы алгоритма обратной трассировки лучей

2.1.1 Определение пересечения луча с полигоном

Так как любую плоскость можно однозначно задать тремя точками, для поиска пересечения луча с полигонами используется алгоритм Моллера – Трумбора [8], с помощью которого можно вычислить пересечение луча с треугольным полигоном.

Пусть треугольный полигон определен вершинами V_0, V_1, V_2 , а луч $R(t)$ с началом в точке O и единичным вектором направления D определен формулой

$$R(t) = O + tD \quad (2.1)$$

И если точку $T(u, v)$ на треугольнике $V_0V_1V_2$ выразить через ее барицентрические координаты (u, v) , так что $(u \geq 0, v \geq 0, u + v \leq 1)$:

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

тогда пересечение луча $R(t)$ и треугольника $V_0V_1V_2$ эквивалентно решению уравнения $R(t) = T(u, v)$ и однозначно определяются параметрами расстояния t от начала луча до точки пересечения и барицентрическими координатами (u, v) . В таком случае получим:

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

Уравнение 2.3 может быть представлено в матричном виде:

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

Пусть $E_1 = V_1 - V_0$, $E_2 = V_2 - V_0$, $T = O - V_0$. Решение уравнения 2.3 можно получить методом Крамера:

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

Если барицентрические координаты точки пересечения, полученной из формулы 2.5 удовлетворяют условию ($u \geq 0, v \geq 0, u + v \leq 1$), то луч $R(t)$ пересекает треугольный полигон заданный вершинами V_0, V_1, V_2 . Если определитель $(D \times E_2) \cdot E_1$ равен нулю, то луч лежит в плоскости треугольника $V_0V_1V_2$.

2.1.2 Определение вектора отражения

Для визуализации отражающих поверхностей в алгоритме обратной трассировки лучей необходим способ определения направления вектора отражения зная луч падения l и нормаль к поверхности n [9].

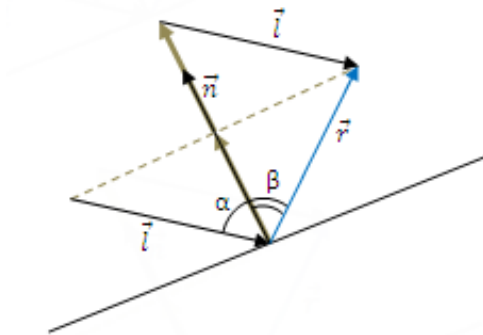


Рисунок 2.1 – Расчет направления вектора отражения

Вектор отражения представляется через разность вектора падения l и вектора нормали n , длина которого равняется длине двух проекций вектора l на n :

$$r = l - 2n \cdot \frac{(l, n)}{(n, n)} \quad (2.6)$$

2.2 Функциональная модель программного обеспечения

Разрабатываемое программное обеспечение должно генерировать кадр на основе информации о положении в пространстве объектов сцены, интенсивности источника света, вектора направления камеры и множестве полигонов, которыми заданы модели фигур. Для создания изображения программное обеспечение использует алгоритмы испускания луча и обратной трассировки луча, которые заключаются в испускании луча для каждого пикселя и вычислении цвета этого пикселя.

На рисунках 2.2-2.3 представлена функциональная модель программного обеспечения в нотации IDEF0.



Рисунок 2.2 – Функциональная модель программы в нотации IDEF0, уровень 0

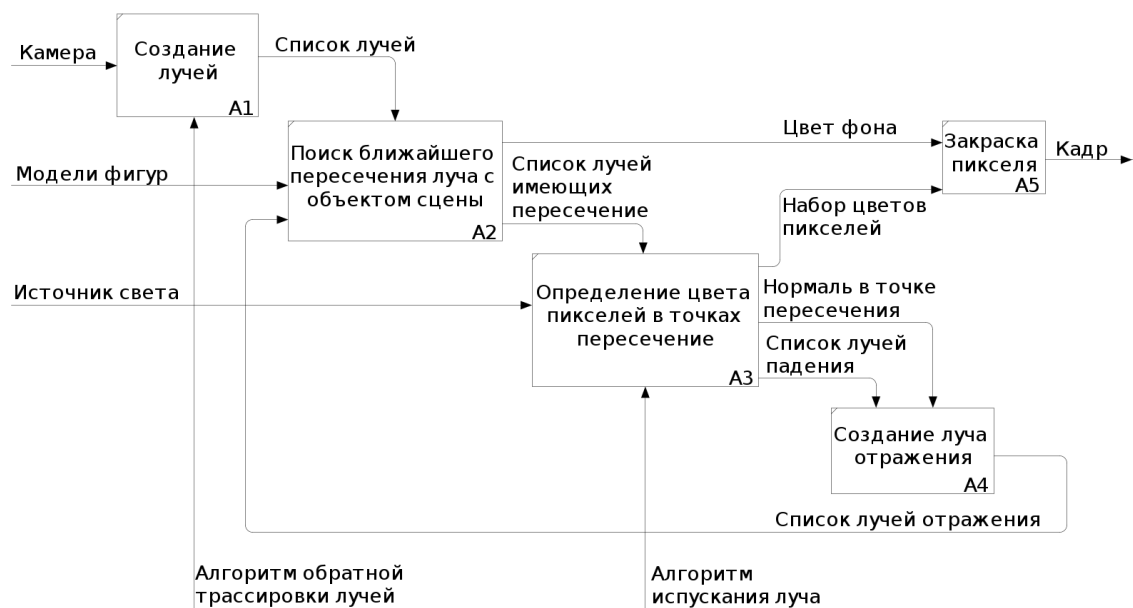


Рисунок 2.3 – Функциональная модель программы в нотации IDEF0, уровень 1

2.3 Описание алгоритма определения цвета пикселя, методом обратной трассировки лучей

Для генерации кадра сцены алгоритмом обратной трассировки лучей необходимо для каждого пикселя изображения вычислить его цвет, с использованием алгоритма трассировки луча, который представлен на рисунках 2.4-2.5. Данный алгоритм для каждого луча исходящего от наблюдателя и проходящего через центр пикселя вычисляет цвет этого пикселя на основе положения моделей сцены, их материалов и источника света.

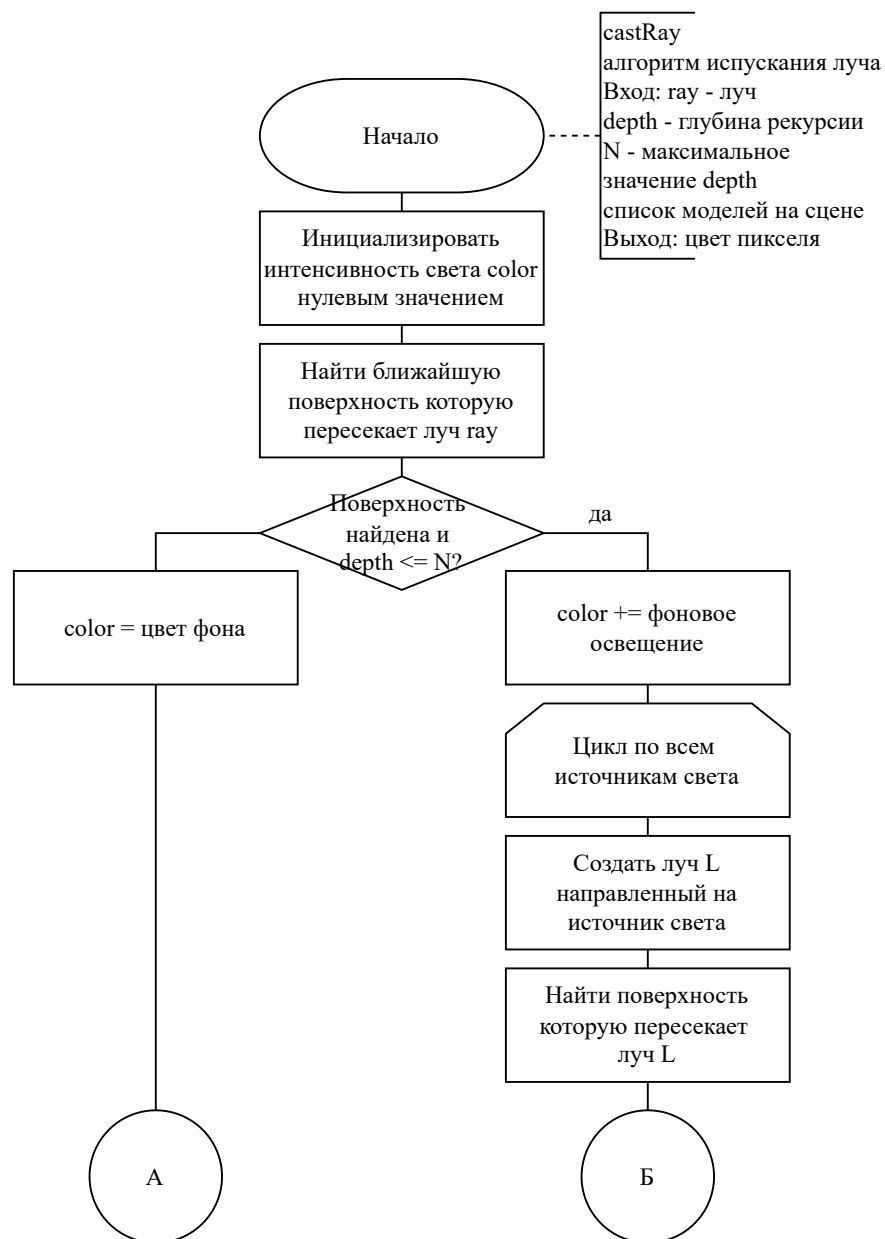


Рисунок 2.4 – Схема алгоритма трассировки луча

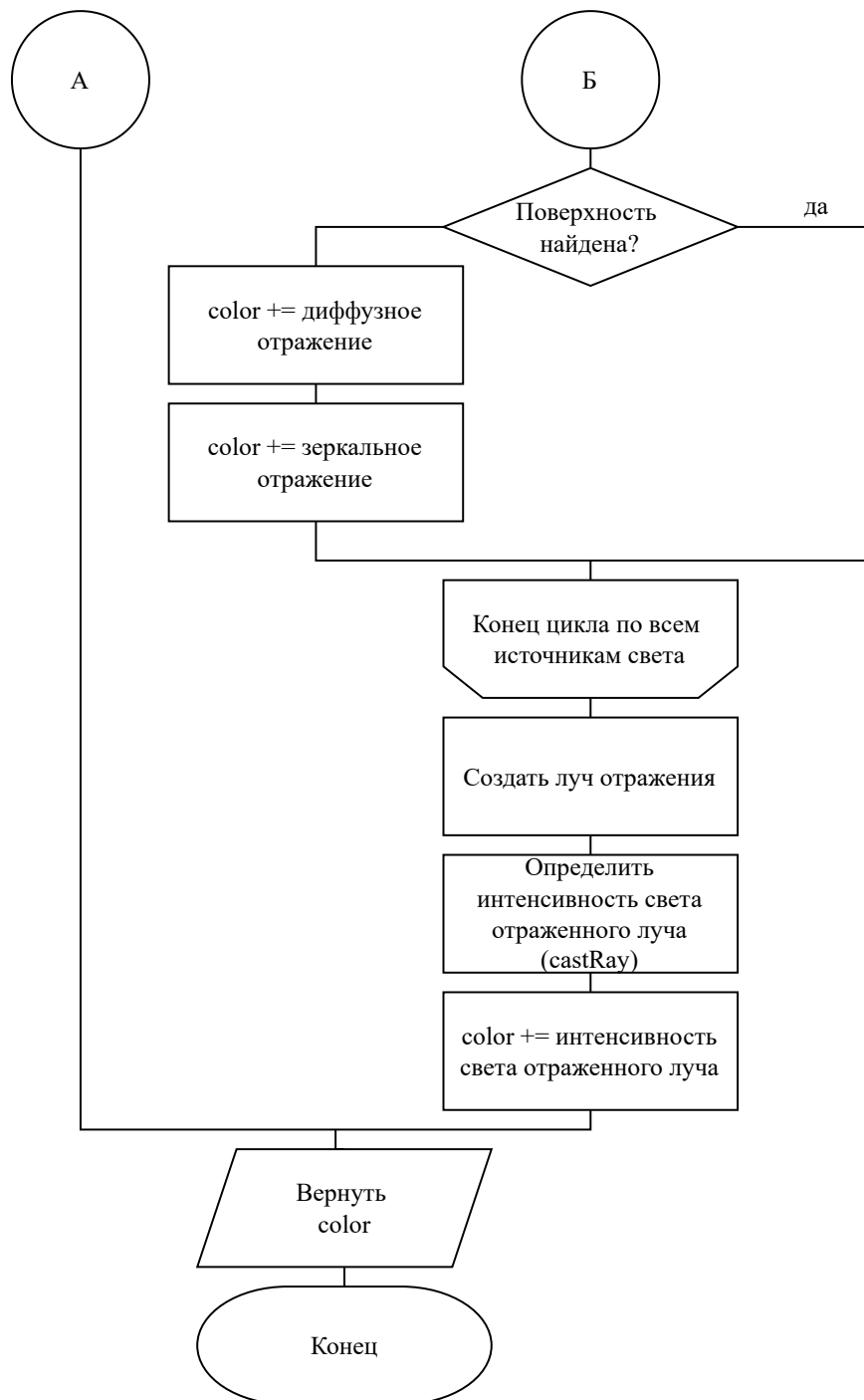


Рисунок 2.5 – Схема алгоритма трассировки луча

2.4 Структура разрабатываемого программного обеспечения

При разработке программного обеспечения использовался объектно-ориентированный подход и паттерны проектирования, для улучшения декомпозиции задачи и облегчения модификаций кода.

На рисунках 2.6-2.8 представлена диаграмма классов разрабатываемого программного обеспечения.

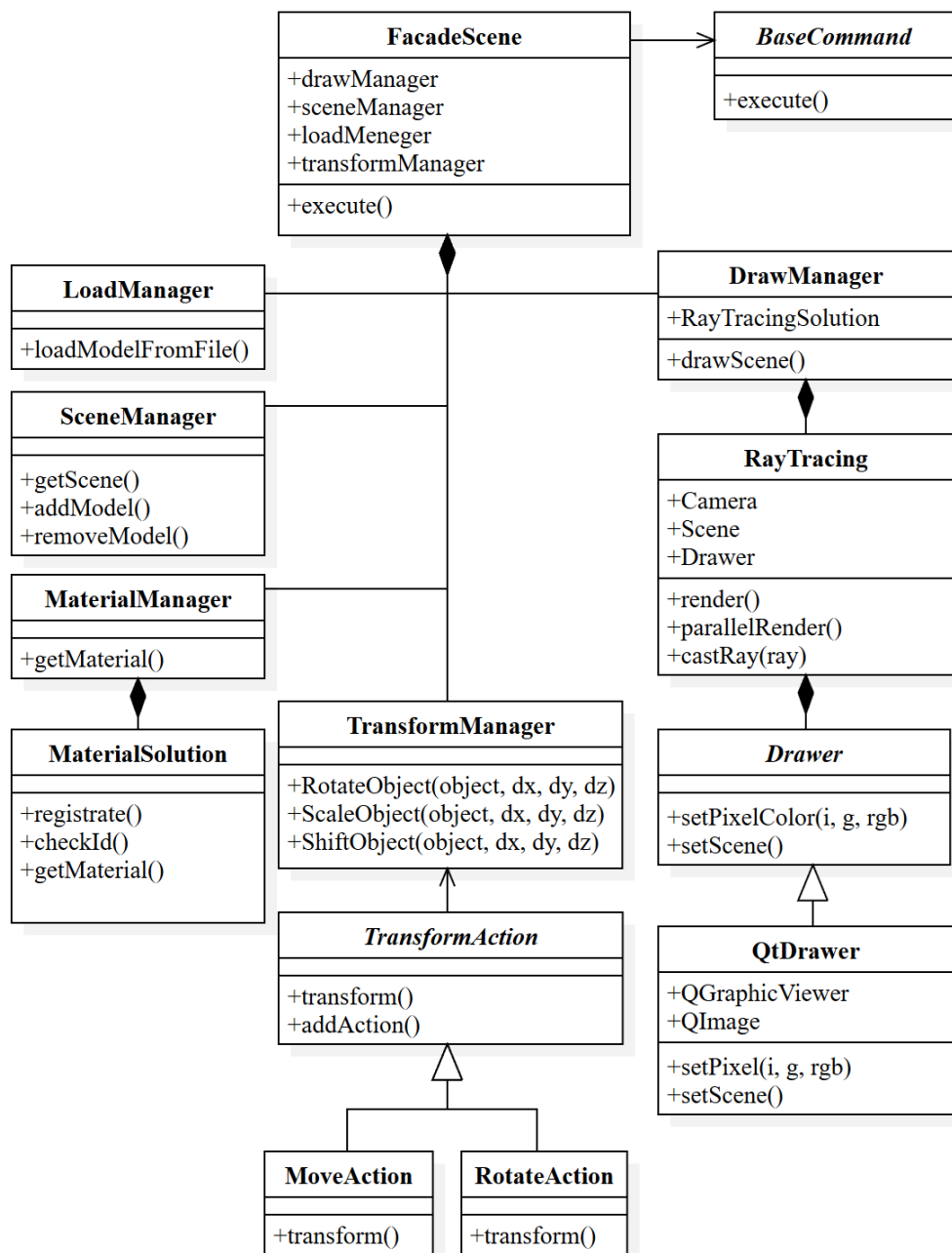


Рисунок 2.6 – Диаграмма классов, которые реализуют доступ графическому интерфейсу пользователя к программному обеспечению

- **FacadeScene** – класс, реализующий структурный паттерн «фасад». Предоставляет графическому интерфейсу пользователя унифицированный интерфейс к программному обеспечению, реализующему взаимодействие со сценой и отрисовку кадра, с помощью паттерна «команда»;
- **BaseCommand** – базовый класс, реализующий поведенческий паттерн «команда». Инкапсулирует запрос пользователя на выполнение действия в виде отдельного объекта. В программе также реализовано несколько классов производных от данного, которые отвечают за каждый запрос пользователя по отдельности;
- **DrawManager** – класс, реализующий операцию отрисовки сцены;
- **RayTracing** – класс, реализующий поведенческий паттерн «стратегия». Определяет реализацию алгоритма обратной трассировки лучей;
- **Drawer** – базовый класс, реализующий структурный паттерн «адаптер». Определяет операцию закрашивания пикселей;
- **QtDrawer** – производный класс от Drawer. Преобразует интерфейс класса QGraphicsViewer библиотеки Qt.
- **MaterialManager** – класс, реализующий взаимодействие со множеством материалов;
- **MaterialSolution** – класс, определяющий множество возможных материалов, которые могут быть использованы в программе;
- **TransformManager** – класс, реализующий операции преобразования объектов сцены;
- **TransformAction** – базовый класс, реализующий поведенческий паттерн «стратегия». Определяет алгоритм преобразования объектов сцены;
- **MoveAction** – производный класс от TransformAction. Определяет алгоритм переноса объектов сцены;
- **RotateAction** – производный класс от TransformAction. Определяет алгоритм вращения объектов сцены;

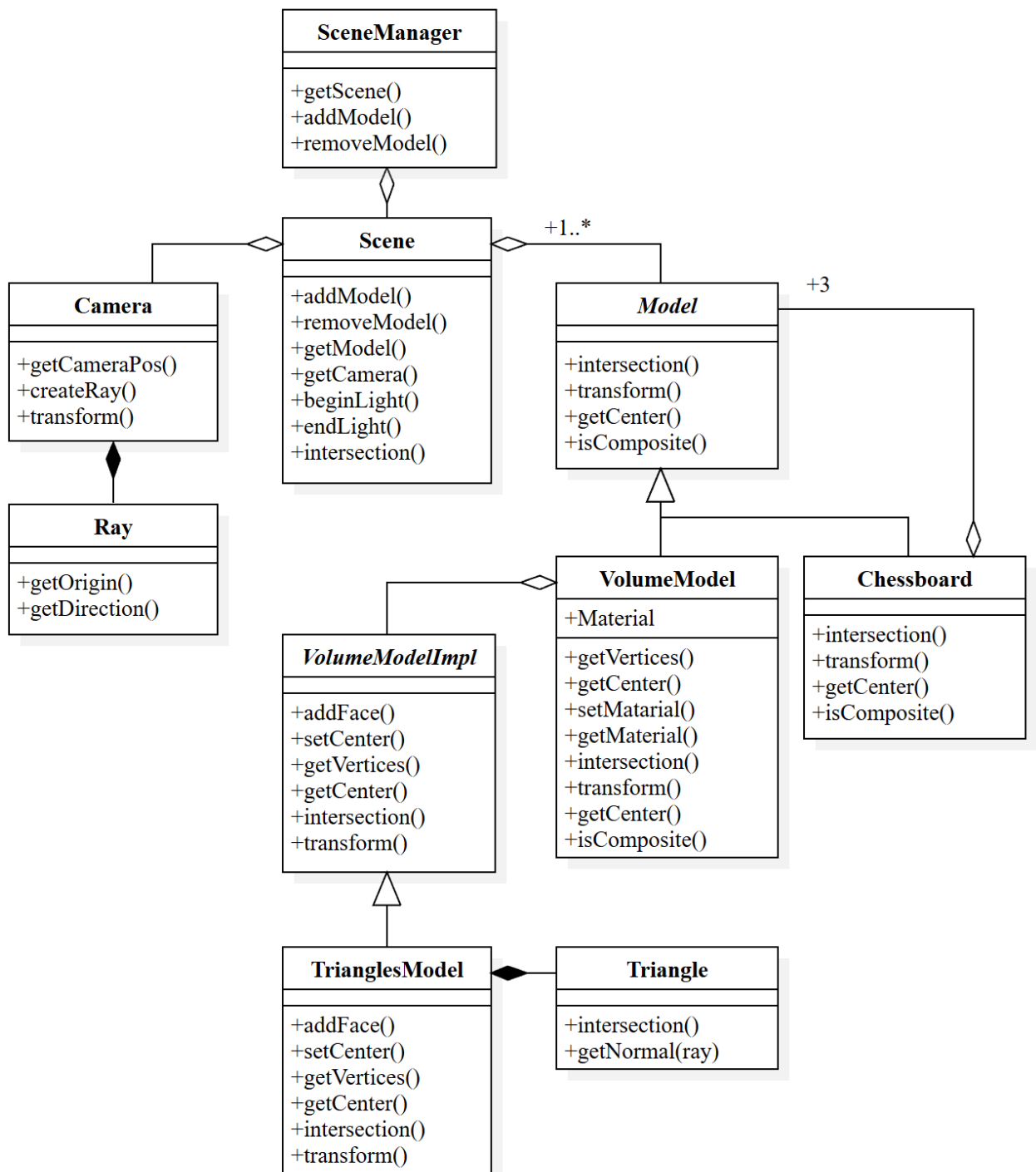


Рисунок 2.7 – Диаграмма классов, описывающих объекты сцены

- **SceneManager** – класс, реализующий взаимодействие с объектами сцены;
- **Scene** – класс сцены. Хранит информацию об объектах сцены;
- **Camera** – класс камеры. Хранит информацию о наблюдателе и реализует операцию создания луча;

- **Ray** – класс луча. Хранит информацию о начальной точке и векторе направления луча;
- **Model** – базовый класс фигуры, которая может быть изображена на сцене;
- **VolumeModel** – производный класс от **Model**, является элементом абстракции структурного паттерна «мост»;
- **VolumeModelImpl** – базовый класс, являющийся элементом реализации структурного паттерна «мост»;
- **TrianglesModel** – производный класс от **VolumeModelImpl**, определяет структуру трехмерной модели, как множество вершин и множество треугольных полигонов;
- **Triangle** – класс, описывающий треугольный полигон и операции над ним;
- **Chessboard** – класс фигуры шахматной доски, реализующий структурный паттерн «компоновщик». Шахматная доска состоит из трех моделей: модели определяющей деревянную основу доски и двух моделей определяющих множества черных и белых клеток;

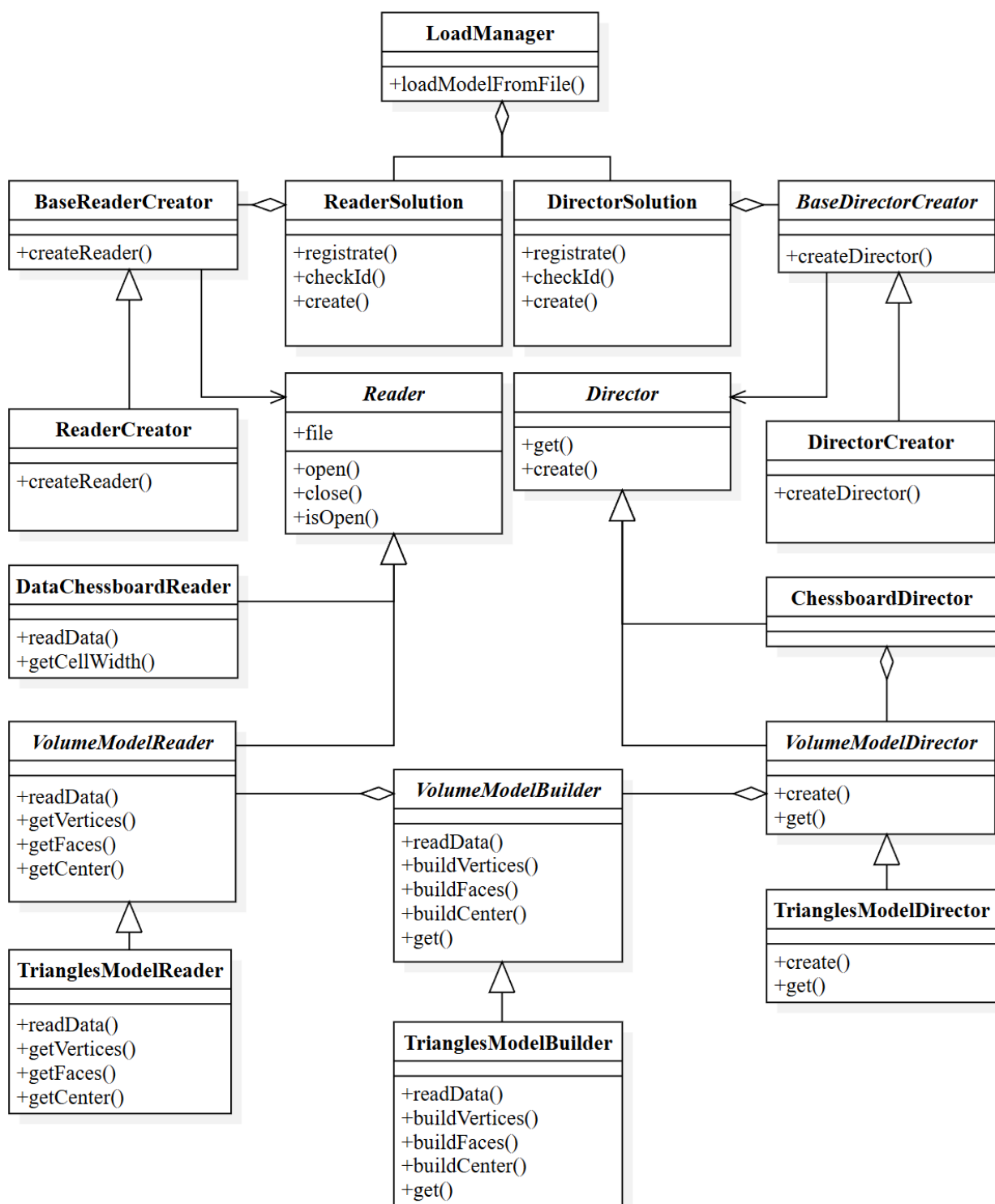


Рисунок 2.8 – Диаграмма классов, которые реализуют загрузку моделей фигур на сцену

- **LoadManager** – класс, реализующий загрузку моделей;
- **ReaderSolution** – класс, определяющий используемый метод чтения файла;
- **BaseReaderCreator** – базовый класс, реализующий порождающий

паттерн «фабричный метод». Определяет реализацию класса Reader, которая будет создана;

- **ReaderCreator** – производный класс от BaseReaderCreator;
- **Reader** – базовый класс, реализующий операцию чтения из файла;
- **DataChessboardReader** – производный класс от Reader. Определяет чтение из файла информации о шахматной доске;
- **VolumeModelReader** – базовый класс, определяющий операцию чтения информации о трехмерной модели;
- **TrianglesModelReader** – производный класс от VolumeModelReader. Определяет чтение из файла информации о трехмерной модели, представленной классом TrianglesModel;
- **DirectorSolution** – класс, определяющий используемый метод создания Model;
- **BaseDirectorCreator** – базовый класс, реализующий порождающий паттерн «фабричный метод». Определяет реализацию класса Director, которая будет создана;
- **DirectorCreator** – производный класс от BaseDirectorCreator;
- **Director** – базовый класс, реализующий порождающий паттерн «строитель», элемент «director». Определяет операцию создания фигуры (Model);
- **ChessboardDirector** – производный класс от Director. Определяет операцию создания класса шахматной доски (Chessboard);
- **VolumeModelDirector** – базовый класс, определяющий операцию создания класса модели (Model);
- **TrianglesModelDirector** – производный класс от VolumeModelDirector. Определяет операцию создания класса трехмерной модели, представленной классом TrianglesModel;

- **VolumeModelBuilder** – базовый класс, реализующий порождающий паттерн «строитель», элемент «builder». Определяет множество операций необходимых для создания фигуры (Model);
- **TrianglesModelBuilder** – производный класс от VolumeModelBuilder. Определяет множество операций необходимых для создания фигуры, представленной классом TrianglesModel;

Вывод из конструкторской части

В данном разделе были представлены математические основы алгоритма обратной трассировки лучей и спроектировано программное обеспечение, которое было описано функциональной моделью, схемой алгоритма обратной трассировки лучей и структурой, представленной в виде диаграммы классов.

3 Технологическая часть

В данном разделе представлены средства реализации, листинги основных алгоритмов, описаны процесс сборки, интерфейс приложения и методы тестирования.

3.1 Средства реализации

В качестве языка программирования был выбран C++ [10] в силу следующих причин:

- В стандартной библиотеке языка присутствует поддержка всех структур данных, выбранных по результатам проектирования;
- Средствами языка можно реализовать все алгоритмы, выбранные в результате проектирования;

Для создания пользовательского интерфейса был использован фреймворк QT [11], так как данный фреймворк предоставляет инструменты для создания пользовательских интерфейсов и поддерживается язык программирования C++.

Для сборки программного обеспечения использовалась утилита CMake [12], так как с ее помощью возможно управлять процессом компиляции и сборки проекта, написанного на C++.

Для модульного тестирования компонент программного обеспечения был выбран фреймворк GoogleTest [13], так как данный фреймворк предоставляет инструменты для написания модульных тестов на языке C++.

Для определения покрытия кода использовалась утилита gcov [14].

3.2 Реализации алгоритма обратной трассировки лучей

Для генерации кадра, состоящего из шахматных фигур, заданных полигонами, и шахматной доски с глянцевой или матовой поверхностью был использован алгоритм обратной трассировки лучей с глобальной моделью освещения. В этом методе для каждого пикселя генерируемого изображения создается луч, исходящий от наблюдателя, и для данного луча запускается алгоритм трассировки луча, представленный в приложении А.

Для поданного на вход луча определяется его пересечение с ближайшим видимым объектом сцены, в случае отсутствия пересечения, пиксель, определяемый лучом, закрашивается цветом фона. Если пересечение найдено, запускается луч, направленный на источник света для определения освещенности точки в пространстве, и запускается луч отражения если поверхность пересечения является глянцевой.

3.3 Описание процесса сборки приложения

Для сборки программного обеспечения использовалась утилита CMake [12]. Для сборки приложения необходимо в командной строке, находясь в директории проекта, выполнить следующие команды 3.1.

Листинг 3.1 – Сборка программного обеспечения

```
$ cd build  
$ cmake -S ..  
$ cmake --build .
```

3.4 Описание интерфейса приложения

На рисунках 3.1-3.5 представлен интерфейс программы.

На рисунке 3.1 представлена вкладка добавления фигуры определенного игрока из выпадающего списка на позицию шахматной доски, изменение цветового набора и изменения материала шахматной доски. На рисунке 3.2 представлена вкладка изменения позиции и вращения модели с индексом, определенным на вкладке 3.5. На рисунке 3.3 представлена вкладка изменения позиции и вращения камеры. На рисунке 3.4 представлена вкладка изменения содержимого сцены. На рисунке 3.4 представлена вывода информации о сцене.

The screenshot shows the 'Создание' (Creation) tab with the following elements:

- Buttons: Создание, Перемещение, Камера, Сцена
- Section: Загрузка фигуры (Load piece)
- Form fields:
 - Ладья (King) - dropdown menu
 - Позиция (Position) - 'd' - dropdown menu
 - 2 - dropdown menu
 - Сторона (Side) - Игрок 2 (Player 2) - dropdown menu
- Button: Загрузить (Load)
- Section: Изменение цветового набора (Change color set)
- Form field: Классический (Classic) - dropdown menu
- Button: Изменить (Change)
- Section: Материал шахматной доски (Board material)
- Form fields:
 - ☒ Глянцевый (Glossy)
 - ☐ Матовый (Matte)

Рисунок 3.1 – Интерфейс программы. Вкладка добавления моделей и настроек материалов

The screenshot shows the 'Перемещение' (Movement) tab with the following elements:

- Buttons: Создание, Перемещение, Камера, Сцена
- Form field: Фигура (Piece) - id=1 - dropdown menu
- Section: Перемещение (Movement)
- Form fields:
 - На позицию (To position) - 'a' - dropdown menu
 - 1 - dropdown menu
- Button: Переместить (Move)
- Section: Вращение (Rotation)
- Form field: Угол вращения (градусы) (Rotation angle (degrees)) - -30,0 - spinner
- Button: Вращать (Rotate)
- Form fields:
 - Удаление (Deletion)
 - Удалить (Delete)

Рисунок 3.2 – Интерфейс программы. Вкладка изменения позиции модели

Создание	Перемещение	Камера	Сцена
Вращение			
Угол вращения (градусы)		30,0	
Влево		Вверх	Вправо
		Вниз	
Перемещение			
dx:	20		
Приблизить			
Отдалить			




Рисунок 3.3 – Интерфейс программы. Вкладка изменения позиции камеры

Создание	Перемещение	Камера	Сцена
Загрузка сцены			
После загрузки новой сцены, данная сцена и фигуры на ней будут удалены без возможности восстановления			
Начальная расстановка			
Загрузить			
Очистить сцену			

Рисунок 3.4 – Интерфейс программы. Вкладка настроек сцены

Цветовой набор: Классический

Фигуры на сцене

id	Фигура	Цвет	Позиция
1	Пешка		d2
2	Конь		b1
3	Слон		g3

Показать

Рисунок 3.5 – Интерфейс программы. Вкладка вывода информации о сцене

3.5 Тестирование

3.5.1 Функциональное тестирование

Для функционального тестирования была рассмотрена сцена, на которой изображены все виды фигур, и выделены следующие классы эквивалентности: камера расположена под углом 30° (3.6); вид сбоку (3.7); вид сверху (3.8).

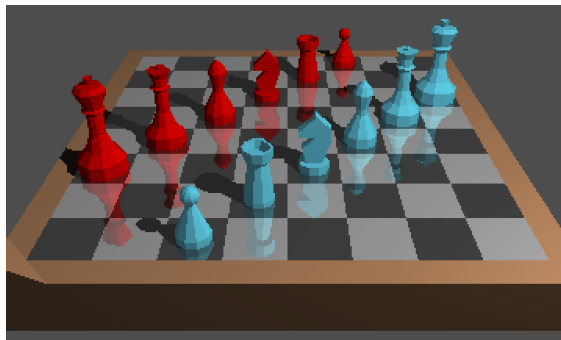


Рисунок 3.6 – Камера расположена под углом 30°



Рисунок 3.7 – Вид сбоку

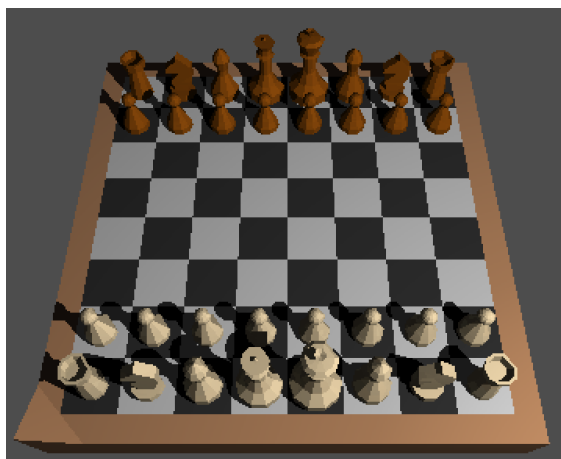


Рисунок 3.8 – Вид сверху

3.5.2 Модульное тестирование

Для модульного тестирования компонент программного обеспечения использовался фреймворк GoogleTest [13].

Были созданы параметризованные тесты для функций классов: *Camera*, *RayTracing*, *Ray* и *Triangle*. Для тестов каждого рассматриваемого объекта использовался класс фиксации, в котором производились настройки окружения, например создание классов *SceneManager* и *MaterialManager*, в функции *SetUp*, а также освобождение памяти, при необходимости, в функции *TearDown*.

3.5.3 Результаты тестирования

В качестве меры полноты тестирования был выбран процент покрытия строк кода. Результаты тестирования, были обработаны утилитой *lcov* [15] и представленный в таблице 3.1:

Таблица 3.1 – Результаты тестирования

Количество протестированных строк кода	Количество строк кода в проекте	Процент покрытия для созданного набора тестов
2376	3096	77 %

Вывод из технологической части

Было реализовано программное обеспечение и представлены средства реализации, описание алгоритма обратной трассировки лучей, описаны интерфейс приложения и методы тестирования.

4 Исследовательская часть

В данном разделе описано исследование зависимости времени генерации кадра от числа полигонов на сцене для варьируемого числа рабочих потоков.

4.1 Технические характеристики

- Гостевая операционная система — Ubuntu 22.04.4 LTS
- Установленная оперативная память (RAM) – 8,0 ГБ;
- Процессор – AMD Ryzen 7 5800H with Radeon Graphics, 3201 МГц, ядер: 8, логических процессоров: 16;

При проведении замеров времени ноутбук был включен в сеть электропитания, а не работал от аккумулятора, и были запущены только встроенное приложение окружения и система замеров времени.

4.2 Цель исследования

Цель исследования – сравнительный анализ зависимости времени генерации кадра от числа полигонов на сцене для варьируемого числа рабочих потоков.

4.3 Описание исследования

Был реализован алгоритм параллельной генерации кадра, для этого использовалась функция создания потоков `pthread_create` [16]. Каждый поток в данной реализации отвечал за определение цвета группы пикселей методом обратной трассировки лучей.

В ходе исследования были проведены замеры реального времени генерации кадра, для этого использовалась функция `gettimeofday` [17]. Было рассмотрено время отрисовки сцены для дополнительных рабочих потоков в размере от 0 (вычисление в основном потоке) до 64, по степеням числа 2, то есть было рассмотрено 0, 1, 2, 4, 8, 16, 32, 64 дополнительных рабочих потока.

Замеры времени проводились по принципу: для одних входных данных проводилось 5 замеров и если относительная стандартная ошибка среднего (rse) была $\geq 5\%$, то для этих данных замеры продолжались, в таблицу заносилось среднее значение.

4.4 Результат исследования

Результаты замеров времени с секундах приведены в таблице 4.1

Таблица 4.1 – Замеры времени отрисовки сцены в секундах от числа полигонов на сцене для варьируемого числа рабочих потоков

Количество полигонов	Количество потоков							
	0	1	2	4	8	16	32	64
100	8.34	12.48	10.53	11.26	11.50	10.64	10.18	8.36
200	17.83	18.26	15.66	14.06	12.53	11.71	11.47	9.25
300	23.28	21.86	17.76	16.34	14.69	13.61	12.86	10.32
400	38.25	31.23	24.48	19.60	17.99	16.69	15.76	12.25
500	47.95	37.93	29.48	22.94	20.95	19.95	18.92	15.45

График зависимости времени отрисовки сцены от числа полигонов на сцене для варьируемого числа рабочих потоков представлен на рисунке 4.1.

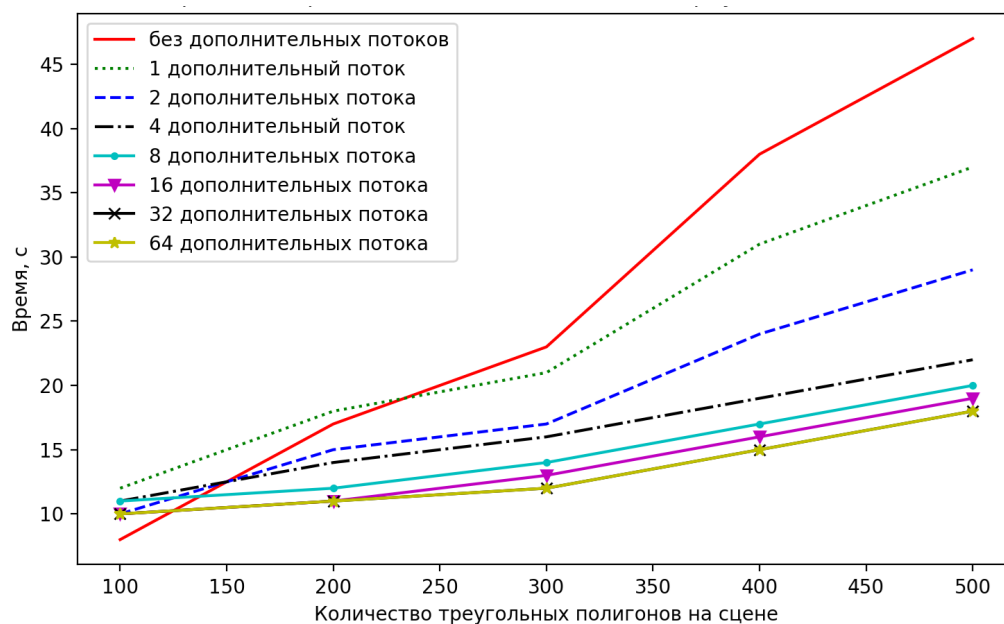


Рисунок 4.1 – Зависимость времени отрисовки сцены от числа полигонов на сцене для варьируемого числа рабочих потоков

Из проведённых замеров можно сделать следующие выводы:

— при увеличении количества потоков от 1 до 32 время обработки сцены уменьшается, при условии что на сцене 200 полигонов и более,

так как в таком случае время необходимое для создания и запуска дополнительных потоков, равно или больше времени отрисовки всей сцены этими потоками;

— при небольшом заполнении сцены (количество полигонов = 100) алгоритм использующий последовательную обработку данных работает быстрее, так как не тратит время на создание потоков;

— при работе 64 потоков и более время генерации кадра не уменьшается, так как затраты времени на создание и запуск потоков становятся слишком большими.

4.5 Вывод из исследовательской части

В данном разделе было проведено исследование зависимости времени генерации кадра от числа полигонов на сцене для варьируемого числа рабочих потоков и в результате сравнительного анализа были сделаны выводы о том, что при количестве полигонов на сцене меньшем 200, эффективнее по времени использовать алгоритм обратной трассировки лучей с одним потоком, а при количестве полигонов большем 200 эффективнее использовать 32 дополнительных рабочих потока.

ЗАКЛЮЧЕНИЕ

В ходе курсовой работы было разработано программное обеспечение, которое позволяет получить реалистичное изображение шахматных фигур на шахматной доске.

В процессе выполнения данной работы были выполнены все задачи:

- 1) формализовать задачу;
- 2) выбрать алгоритм построения реалистичного изображения;
- 3) разработать функциональную модель программного обеспечения;
- 4) выбрать средства реализации и реализовать программное обеспечение для визуализации сцены;
- 5) исследовать зависимость времени генерации кадра от числа полигонов на сцене для варьируемого числа рабочих потоков.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. ФИДЕ Техническая комиссия. Стандарты шахматного оборудования и игровых площадок, предназначенных для проведения турниров ФИДЕ. Положение о тай-брейках. 2015. URL: Режим доступа: <https://ruchess.ru/upload/iblock/8d7/8d7dfc75a3c874ab01612272fe6c5964.pdf> (дата обращения: 01.12.2024).
2. Дёмин А. Ю. Основы компьютерной графики: учебное пособие. Томск: Изд-во Томского политехнического университета, 2011. с. 191. Томский политехнический университет.
3. Д. Роджерс. Алгоритмические основы машинной графики. Москва-Мир, 1989. с. 512. Пер. с англ.
4. Шикин Е. В. Боресков А. В. Зайцев А. А. Начала компьютерной графики. ДИАЛОГ-МИФИ, 1993. с. 138.
5. Шикин Е. В. Боресков А. В. Компьютерная графика. Динамика, реалистические изображения. Москва: ДИАЛОГ-МИФИ, 1996. с. 288.
6. Bui-Tuong Phong. Illumination for Computer Generated Images. 1975. с. 317.
7. Шикин А. В., Боресков А. В. Компьютерная графика. Полигональные модели. Москва: ДИАЛОГ-МИФИ, 2001. с. 464.
8. Tomas Möller Ben Trumbore. Fast, Minimum Storage Ray/Triangle Intersection. Journal of Graphics Tools, 1997. с. 7. URL: Режим доступа: <https://www.graphics.cornell.edu/pubs/1997/MT97.pdf> (дата обращения: 01.12.2024).
9. Peter Shirley Trevor David Black Steve Hollasch. Ray Tracing in One Weekend. 2024. URL: Режим доступа: <https://raytracing.github.io/books/RayTracingInOneWeekend.html> (дата обращения: 01.12.2024).
10. Standard C++ [Электронный ресурс]. URL: Режим доступа: <https://isocpp.org/> (дата обращения: 01.12.2024).

11. Qt | Cross-platform software development for embedded & desktop [Электронный ресурс]. URL: Режим доступа: <https://www.qt.io/> (дата обращения: 01.12.2024).
12. CMake [Электронный ресурс]. URL: Режим доступа: <https://cmake.org/> (дата обращения: 01.12.2024).
13. GoogleTest [Электронный ресурс]. URL: Режим доступа: <https://google.github.io/googletest/advanced.html> (дата обращения: 01.12.2024).
14. gcov(1) - Linux manual page [Электронный ресурс]. URL: Режим доступа: <https://man7.org/linux/man-pages/man1/gcov.1.html> (дата обращения: 01.12.2024).
15. lcov documentation — lcov 1.15.5a0 documentation [Электронный ресурс]. URL: Режим доступа: <https://lcov.readthedocs.io/en/latest/index.html> (дата обращения: 01.12.2024).
16. pthread_create(3) — Linux manual page [Электронный ресурс]. Режим доступа: https://man7.org/linux/man-pages/man3/pthread_create.3.html. (дата обращения: 20.10.2024).
17. gettimeofday(2) — Linux manual page [Электронный ресурс]. Режим доступа: <https://man7.org/linux/man-pages/man2/gettimeofday.2.html>. (дата обращения: 20.10.2024).

Приложение А

Листинг 4.1 – Реализация алгоритма трассировки луча

```
Intensity StandardRayTracing::castRay(Ray &ray, const size_t
    depth, bool printing) const noexcept {
    if (!ray.getDirection().isNormalized())
        ray.getDirection().normalize();

    Intensity color(0, 0, 0);

    intersection_t intersect;
    if (scene->intersection(ray, intersect) && depth < maxDepth)
    {
        Point3 posLight;
        Vector3 L;
        intersection_t tmpIntersect;
        Vector3 diff, spec;
        double ddist;
        for (Scene::iteratorLight it = scene->beginLight(); it !=
            scene->endLight(); ++it) {
            if ((*it)->getType() == typeLight::POINT) {
                L = (posLight - intersect.point).normalized();

                if (fabs(intersect.normal.length()) < EPS) {
                    return Intensity(1, 0, 0);
                }
                if (!intersect.normal.isNormalized())
                    intersect.normal.normalize();
                double nL = L.scalarProduct(intersect.normal);

                if (nL > 0 && !scene->intersection(Ray(intersect.point
                    + 1e-3 * intersect.normal, L), tmpIntersect)) {

                    diff = intersect.material->getKd() * nL;

                    Vector3 reflectLight = L.reflect(intersect.normal);
                    double SR = reflectLight.scalarProduct(-ray.
                        getDirection());
                    if (SR < 0)
                        SR = 0;
                }
            }
        }
    }
    return color;
```

```

        spec = intersect.material->getKs() * pow(SR,
            intersect.material->getN());

        ddist = (posLight - intersect.point).length() / (
            posLight.length() + EPS);
        if (ddist > 1.0)
            ddist = 1;
        else if (ddist < 0.4)
            ddist = 0.4;

        color += (diff + spec) * (*it)->getIntensity() /
            ddist;
    }

    } else if ((*it)->getType() == typeLight::AMBIENT) {
        color += intersect.material->getKa() * (*it)->
            getIntensity();
    }
}

Vector3 reflectVec = ray.getDirection().reflect(intersect.
    normal);
Ray reflectRay(intersect.point + 1e-3 * intersect.normal,
    reflectVec);
Intensity reflectIntensity = castRay(reflectRay, depth +
    1);

    color += intersect.material->getKs() * reflectIntensity;
}
else
    color = Intensity(0.3, 0.3, 0.3);

return color;
}

```

Приложение Б

Презентация к курсовой работе 16 с.