**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

КУРСОВАЯ РАБОТА

***НА ТЕМУ:***

***«Визуализация шахматных фигур на шахматной доске»***

|  |  |  |  |
| --- | --- | --- | --- |
| Студент | ИУ7-52Б |  | Пермякова Е. Д. |
| Руководитель курсового | (группа) | (подпись, дата) | (И.О. Фамилия) |
| проекта |  |  | Силантьева А. В. |
|  |  | (подпись, дата) | (И.О. Фамилия) |

*2024 г.*

**Министерство науки и высшего образования Российской Федерации Федеральное государственное бюджетное образовательное учреждение высшего образования**

**«Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)»**

**(МГТУ им. Н.Э. Баумана)**

УТВЕРЖДАЮ

Заведующий кафедрой ИУ7

(индекс)

И.В.Рудаков

(И.О. Фамилия)

(подпись)

**З А Д А Н И Е**

# на выполнение курсовой работы

(дата)

по дисциплине Компьютерная графика Студент группы ИУ7-52Б Пермякова Екатерина Дмитриевна

(Фамилия, имя, отчество)

Тема курсовой работы

Визуализация шахматных фигур на шахматной доске

Направленность КР (учебная, исследовательская, практическая, производственная, др.) учебная

Источник тематики (кафедра, предприятие, НИР) кафедра

***Задание***

Разработать программное обеспечение с пользовательским интерфейсом для визуализации сцены, состоящей из шахматных фигур и шахматной доски. Пользователю должна быть

предоставлена возможность из предложенного списка выбрать шахматные фигуры, которые будут изображены на сцене. Шахматные фигуры могут располагаться только в центре клетки на поверхности шахматной доски. Материал и цвет фигур будет определяться пользователем из предложенных вариантов. Пользователю должна быть предоставлена возможность вращать

шахматные фигуры вокруг своей оси и перемещать их между клетками шахматной доски. Поверхность шахматной доски должна состоять из черных и белых клеток и в зависимости от выбора пользователя должна иметь глянцевую или матовую поверхность. Размер шахматной доски 8х8 клеток. На сцене должен быть один точечный источник света, который будет располагаться в фиксированной точке вне поля зрения пользователя. В начале работы

программы на шахматной доске не должно быть фигур.

***Оформление курсовой работы:***

Расчетно-пояснительная записка на 25-30 листах формата А4.

Расчетно-пояснительная записка должна содержать постановку введение, аналитическую часть, конструкторскую часть, технологическую часть, экспериментально-исследовательский раздел, заключение, список литературы, приложения.

Перечень графического материала (плакаты, схемы, чертежи и т.п.) На защиту проекта должна быть представлена презентация, состоящая из 15-20 слайдов. На слайдах должны быть

отражены: постановка задачи, использованные методы и алгоритмы, расчетные соотношения, структура комплекса программ, диаграмма классов, интерфейс, характеристики разработанного ПО, результаты проведенных исследований.

Дата выдачи задания « » 20 г.

|  |  |  |
| --- | --- | --- |
| **Руководитель курсовой работы** |  | Силантьева А В |
| **Студент** | (подпись, дата) | (И.О. Фамилия)  Пермякова Е Д |
|  | (подпись, дата) | (И.О. Фамилия) |

**СОДЕРЖАНИЕ**

[ВВЕДЕНИЕ](#_bookmark0) 6

1. [Аналитическая часть](#_bookmark1) 7
   1. [Формализация задачи](#_bookmark2) . . . . . . . . . . . . . . . . . . . . . . . . 7
      1. [Функциональные требования к разрабатываемому про-](#_bookmark3) [граммному обеспечению](#_bookmark3) . . . . . . . . . . . . . . . . . . 7
      2. [Формализация объектов сцены](#_bookmark5) . . . . . . . . . . . . . . . 7
   2. [Анализ способов представления поверхностей трехмерных моделей](#_bookmark6) 9
   3. [Анализ алгоритмов удаления невидимых линий и поверхностей](#_bookmark7) 10
      1. [Алгоритм Робертса](#_bookmark8) 10
      2. [Алгоритм, использующий z-буфер](#_bookmark9) 11
      3. [Алгоритм обратной трассировки лучей](#_bookmark10) 11
   4. [Анализ модели освещения](#_bookmark11) 13
      1. [Локальная модель освещения](#_bookmark12) 13
      2. [Глобальная модель освещения](#_bookmark16) 14
2. [Конструкторская часть](#_bookmark17) 17
   1. [Математические основы алгоритма обратной трассировки лучей](#_bookmark18) 17
      1. [Определение пересечения луча с полигоном](#_bookmark19) 17
      2. [Определение вектора отраженния](#_bookmark22) 18
   2. [Функциональная модель программного обеспечения](#_bookmark23) 19
   3. [Описание алгоритма обратной трассировки лучей](#_bookmark26) 20
   4. [Структура разрабатываемого программного обеспечения](#_bookmark29) 22
      1. [Диаграмма классов](#_bookmark30) 22
      2. [Описание классов используемых в программе](#_bookmark33) 25
3. [Технологическая часть](#_bookmark34) 30
   1. [Средства реализации](#_bookmark35) 30
   2. [Реализации алгоритмов](#_bookmark36) 31
   3. [Описание процесса сборки приложения](#_bookmark39) 34
   4. [Описание интерфейса приложения](#_bookmark41) 35
   5. [Тестирование](#_bookmark42) 36
      1. [Функциональное тестирование](#_bookmark43) 36
      2. [Модульное тестирование](#_bookmark44) 36

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ](#_bookmark44) 38

**РЕФЕРАТ**

# ВВЕДЕНИЕ

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

Для достижения поставленной цели требуется решить следующие задачи:

1. провести анализ алгоритмов построения реалистичных изображений;
2. выбрать способ представления обьектов сцены;
3. перечислить объекты, которые возможно разместить на сцене;
4. формализовать структуру разрабатываемого программного обеспече- ния;
5. реализовать программное обеспечения для визуализации сцены;
6. исследовать зависимость времени выполнения однопоточной и многопо- точной реализаций метода построения реалистичного изображения полого кусочка льда с жидкостью внутри от размера изображения.
7. исследовать временные характеристики отрисовки сцены для многопо- точной и однопоточной реализации алгоритма построения реалистичных изображений;

# Аналитическая часть

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

# Формализация задачи

# Функциональные требования к разрабатываемому про- граммному обеспечению

На рисунке [1.1](#_bookmark4) представлена функциональная модель программы в нотации IDEF0, характеризующая требования к програмному обеспечению.

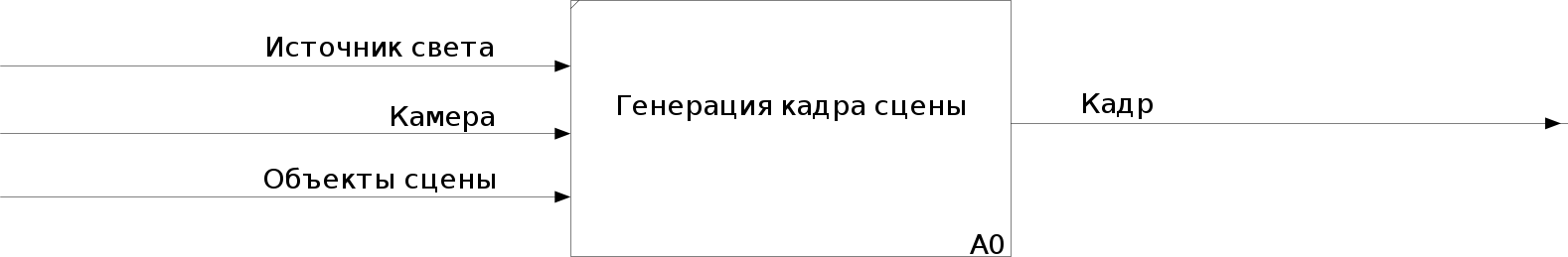


Рисунок 1.1 – Функциональная модель программы в нотации IDEF0

# Формализация объектов сцены

Сцена состоит из следующих обьектов:

* + - * Шахматная доска 8x8 клеток. Основание шахматной доски цвета дерева, клетки – черного и белого цвета;
      * Набор шахмантых фигуры двух цветов: пешка, ладья, конь, слон, ферзь, король;
      * Точечный источник света. Задается положением в пространстве и ин- тенсивностью излучейния;
      * Камера. Задается положением в пространстве и вектором взгляда.

Форма и размер шахматных фигур и шахматной доски соответсвует стан- дарту шахматного оборудования и игровых площадок, предназначенных для проведения турниров ФИДЕ [[1].](#_bookmark45)

# Анализ способов представления поверхностей трехмерных моделей

Поверхность трехмерной модели можно задать несколькими способами [[2]:](#_bookmark46)

* **Полигональная сетка.** В данном случае поверхность представляеися как совокупность связанных между собой плоских многоугольников. Боль- шинство обьектов, не имеющих изгибов, например, таких как шахматная достка, можно точно описать полигональной сеткой. Этот способ также применяется для представления объектов, ограниченных криволинейными поверхностями, однако в таком случае обьект будет описан достаточно приблизительно.
* **Аналитический способ.** Поверхность заданная таким способом описы- вается функцией зависимости координат от некоторого параметра. Главным достоинством данного метода является высокая точность описания поверх- ности, которая нужна в большинстве вычислительных программ, однако из-за необходимости проведения большого количества математических вычислений при визуализации данных поверхносей, 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 предсталвения поверхностей. Именно поэтому для уменьшения времени отрисовки сцены был выбран способ задания поверхностей моделей полигональной сеткой.

# Анализ алгоритмов удаления невидимых линий и поверх- ностей

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

# 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].](#_bookmark47)

## Этапы алгоритма:

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

⎛*𝑎*1 *𝑎*2 *. . . 𝑎𝑛*⎞

*𝑉* = ⎜*𝑏*1 *𝑏*2 *. . . 𝑏𝑛* ⎟ *.* (1.1)

⎜⎝*𝑐*1 *𝑐*2 *. . . 𝑐𝑛*⎟⎠

*𝑑*1 *𝑑*2 *. . . 𝑑𝑛*

1. **Удаление ребер, экранируемых самим телом.** Используется вектор взгляда *𝐸* = (0*,* 0*, −*1*,* 0) для определения невидимых граней. При умноже- нии вектора *𝐸* на матрицу тела *𝑉* отрицательные компоненты полученного вектора будут соответствовать невидимым граням;
2. **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;

# 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].](#_bookmark48)

## Этапы алгоритма:

1. Инициализация Z–буфера минимально возможными значениями и бу- фера 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*).

1. 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;

# Алгоритм обратной трассировки лучей

Алгоритм обратной трассировки лучей заключается в отслеживии траектории лучей, которые исходят из точки наблюдателя и проходят через центр пикселя растра в направлении сцены. [[5].](#_bookmark49)

## Этапы алгоритма:

1. Преобразование сцены в пространство изображения, т. е. область види- мости наблюдателя разбивается на пиксели.
2. Испускание лучей от наблюдателя через пиксели растра 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 отража- ющих 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 реа- лизовать эффект отражения, необходимый для решения поставленной задачи. А так же данный алгоритм не требует дополнительной реализации алгоритмов закраски и построения теней.

# Анализ модели освещения

Модель освещения определяет цвет поверхности обьекта отображаемого на экране. Модель освещения бывает двух видов: локальная и глобальная [[3].](#_bookmark47) В локальной модели освещения учитывается только свет падающий от источников и ориентация поверхности в пространстве. В глобальной модели освещения дополнительно еще учитывается свет, отражённый от других поверхностей и/или пропущенный через них.

# Локальная модель освещения

Локальная модель освещения состоит из трех компонент [[6]:](#_bookmark50)

1. **Фоновое освещение.** Данная составляющая позволяет учитывать свет постоянной яркости, созданный многочисленными отражениями от различ- ных поверхностей. Такой свет практически всегда присутствует в реальной обстановке. Интенсивность рассеяного света можно рассчитать по форму- ле [(1.2)](#_bookmark13)

*𝐼𝑎* = *𝑘𝑎 · 𝑖𝑎* (1.2)

где *𝐼𝑎* — интенсивность рассеяного света, *𝑘𝑎* — коэффициент фонового освещения, *𝑖𝑎* — интенсивность источника рассеяного света.

1. **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)](#_bookmark14)

*→− →−*

*𝐼𝑑* = *𝑘𝑑 · 𝐼𝑙 ·*

cos( *𝐿 , 𝑁* )

*𝑟* + *𝑘*

(1.3)

где *𝐼𝑑* — интенсивность диффузного отражения света, *𝑘𝑑* — коэффициент диффузного отражения, *𝐼𝑙* — интенсивность точечного источника света,

*→−*

*→−*

*𝐿* — вектор направленный на источник света, *𝑁* — вектор нормали к

поверхности, *𝑟* — расстояние от центра проекции до поверхности *𝑘* — произвольная постоянная.

1. **Зеркальное отражение** Направленное отражение благодары которому на блестящих обьектах образовываются блики. Наблюдатель видит зер-

кально 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)](#_bookmark15)

*→− →−*

( *𝑛*

*𝐼𝑠* = *𝑘𝑠 · 𝐼𝑙 ·*

*𝑅 , 𝑆* )

*𝑟* + *𝑘*

*,* (1.4)

где *𝐼𝑠* — интенсивность зеркального отражения света, *𝑘𝑠* — коэффициент

*→−*

*𝑅*

зеркального отражения, *𝐼𝑙* — интенсивность точечного источника света,

* вектор отраженного луча, *→−* — вектор направленый на наблюдателя, *𝑛* —

*𝑆*

степень, аппроксимирующая пространственное распределение зеркально отраженного света, *𝑟* — расстояние от центра проекции до поверхности *𝑘*

* произвольная постоянная.

# Глобальная модель освещения

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

Глобальная модель освещения складывается из непосредственной освещен- ности точки источником света, которая рассчитывается по локальной модели освещения, и вторичной освещенности, которая в свою очередь состоит из интенсиовности света отраженного и преломленного луча [[7].](#_bookmark51) Интенсивность света в точке по глобальной модели освещения расчитывается формулой (**??**):

*𝐼* = *𝐼𝑎* + *𝐼𝑑* + *𝐼𝑠* + *𝑘𝑠 · 𝐼𝑟* + *𝑘𝑡 · 𝐼𝑡,* (1.5)

где *𝐼𝑎* — интенсивность рассеяного света [(1.2),](#_bookmark13) *𝐼𝑑* — интенсивность диффузного отражения света [(1.3),](#_bookmark14) *𝐼𝑠* — интенсивность зеркального отражения света [(1.4),](#_bookmark15)

*𝑘𝑠* — коэффициент зеркального отражения, *𝑘𝑡* — коэффициент пропускания,

*𝐼𝑟* — интенсиовности света отраженного луча, *𝐼𝑡* — интенсиовности света преломленного луча.

# Вывод

Так как для решения поставленной задачи необходимо реализовать отражаю- щую 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 формализацована поставленная задача, были рас- смотрены 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.

# Конструкторская часть

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

# Математические основы алгоритма обратной трассировки лучей

# 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],](#_bookmark52) с помощью которого можно вычислить пересечение луча с треугольным полигоном. Рассмотрим данн

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* (2.1)

И если точку *𝑇* (*𝑢, 𝑣*) на треугольнике *𝑉*0*𝑉*1*𝑉*2 выразить через ее барицентри- ческие координаты (*𝑢, 𝑣*), так что (*𝑢 ≥* 0*, 𝑣 ≥* 0*, 𝑢* + *𝑣 ≤* 1):

*𝑇* (*𝑢, 𝑣*) = (1 *− 𝑢 − 𝑣*)*𝑉*0 + *𝑢𝑉*1 + *𝑣𝑉*2*,* (2.2)

тогда пересечение луча *𝑅*(*𝑡*) и треугольника *𝑉*0*𝑉*1*𝑉*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 (2.3) Уравнение [2.3](#_bookmark20) может быть представнено в матричном виде:

[︁*−𝐷 𝑉*1 *− 𝑉*0*, 𝑉*2 *− 𝑉*0

]︁ ⎡ *𝑡* ⎤

*𝑣*

⎢⎣*𝑢*⎦⎥ = *𝑂 − 𝑉*0 (2.4)

Пусть *𝐸*1 = *𝑉*1 *−𝑉*0*, 𝐸*2 = *𝑉*2 *−𝑉*0, *𝑇* = *𝑂 −𝑉*0. Решение уравнения [2.3](#_bookmark20) можно получить методом Крамера:

⎡ *𝑡* ⎤

1 ⎡(*𝑇 × 𝐸*1) *· 𝐸*2⎤

⎢*𝑢*⎥ =

⎣ ⎦

⎣ ⎦

*𝑣*

(*𝐷 × 𝐸*2

) *· 𝐸*1

⎢ (*𝐷 × 𝐸*2) *· 𝑇* ⎥ (2.5)

(*𝑇 × 𝐸*1) *· 𝐷*

Если баричентически координаты точки пересечения полученной из фор- мулы [2.5](#_bookmark21) удовлетворяют условию (*𝑢 ≥* 0*, 𝑣 ≥* 0*, 𝑢* + *𝑣 ≤* 1), то луч *𝑅*(*𝑡*) пересе- кает треугольный полигон заданный вершинами *𝑉*0*, 𝑉*1*, 𝑉*2. Если определитель (*𝐷 × 𝐸*2) *· 𝐸*1 равен нулю то луч лежит в плоскости треугольника *𝑉*0*𝑉*1*𝑉*2.

# Определение вектора отраженния

Для визуализации отражающих поверхностей в алгоритме обратной трасси- ровки лучей необходим способ определения направления вектора отражения зная луч падения *𝑙* и нормаль к поверхности *𝑛* [[9].](#_bookmark53)

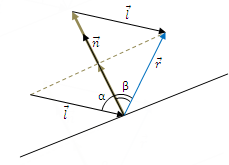


Рисунок 2.1 – Расчет направления вектора отражения

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

*𝑅* = *𝑙 −* 2*𝑛 ·*

(*𝑙, 𝑛*)

(*𝑛, 𝑛*)

(2.6)

# Функциональная модель программного обеспечения

На рисунках [2.2-2.3](#_bookmark25) представлена функциональная модель программного обеспечения в нотации IDEF0.



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

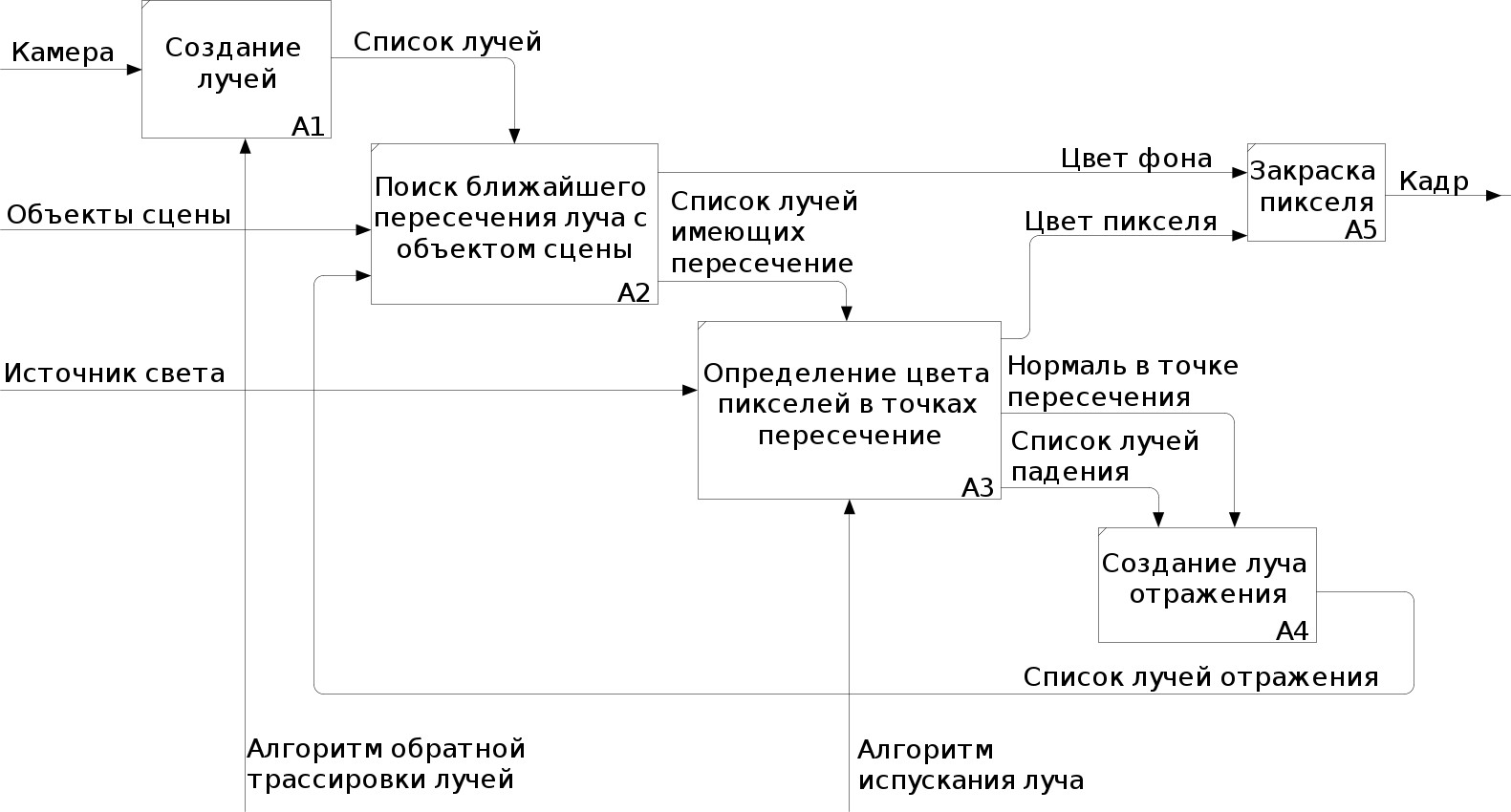


Рисунок 2.3 – Функциональная модель программы в нотации IDEF0, уровень 1

# Описание алгоритма обратной трассировки лучей

На рисунках [2.4-](#_bookmark27) [2.5](#_bookmark28) представлены схемы алгоритма испускания луча и алгоритма обратной трассировки лучей.

Конец

Конец цикла по i

Конец цикла по j

Закраска пикселя (i, j) цветом color

создать луч ray через пиксель (i, j)

Вычислить интенсивность света color для луча ray (castRay)

Цикл j от 1 до height с шагом 1

Цикл i от 0 до width с шагом 1

render

алгоритм трасировки лучей Вход:

width - ширина кадра (пиксели) height - высота кадра (пиксели) камера, список моделей на сцене Выход: кадр

Начало

Рисунок 2.4 – Схема алгоритма обратной трассировки лучей

Начало

castRay

алгоритм испускания луча Вход: ray - луч

color += зеркальное отражение

Создать луч отражения

Определенить иненсивность света отраженного луча (castRay)

color += иненсивность света отраженного луча

A

color += фоновое освещение

color += диффузное отражение

Найти поверхность которую пересекает луч L

Создать луч L направленный на источник света

Инициализировать интенсивность света color нулевым значением

Конец цикла по всем источникам света

да

Поверхность найдена?

Цикл по всем источникам света

color = цвет фона

да

Поверхность найдена и depth <= N?

Найти ближайшую поверхность которую пересекает луч ray

depth - глубина рекурсии

N - максимальное значение depth список моделей на сцене

Выход: цвет пикселя

A

Вернуть color

Конец

Рисунок 2.5 – Схема алгоритма испускания луча

# Структура разрабатываемого программного обеспечения

# Диаграмма классов

На рисунках [2.6-](#_bookmark31) [2.8](#_bookmark32) представлена диаграмма классов разрабатываемого программного обеспечения.

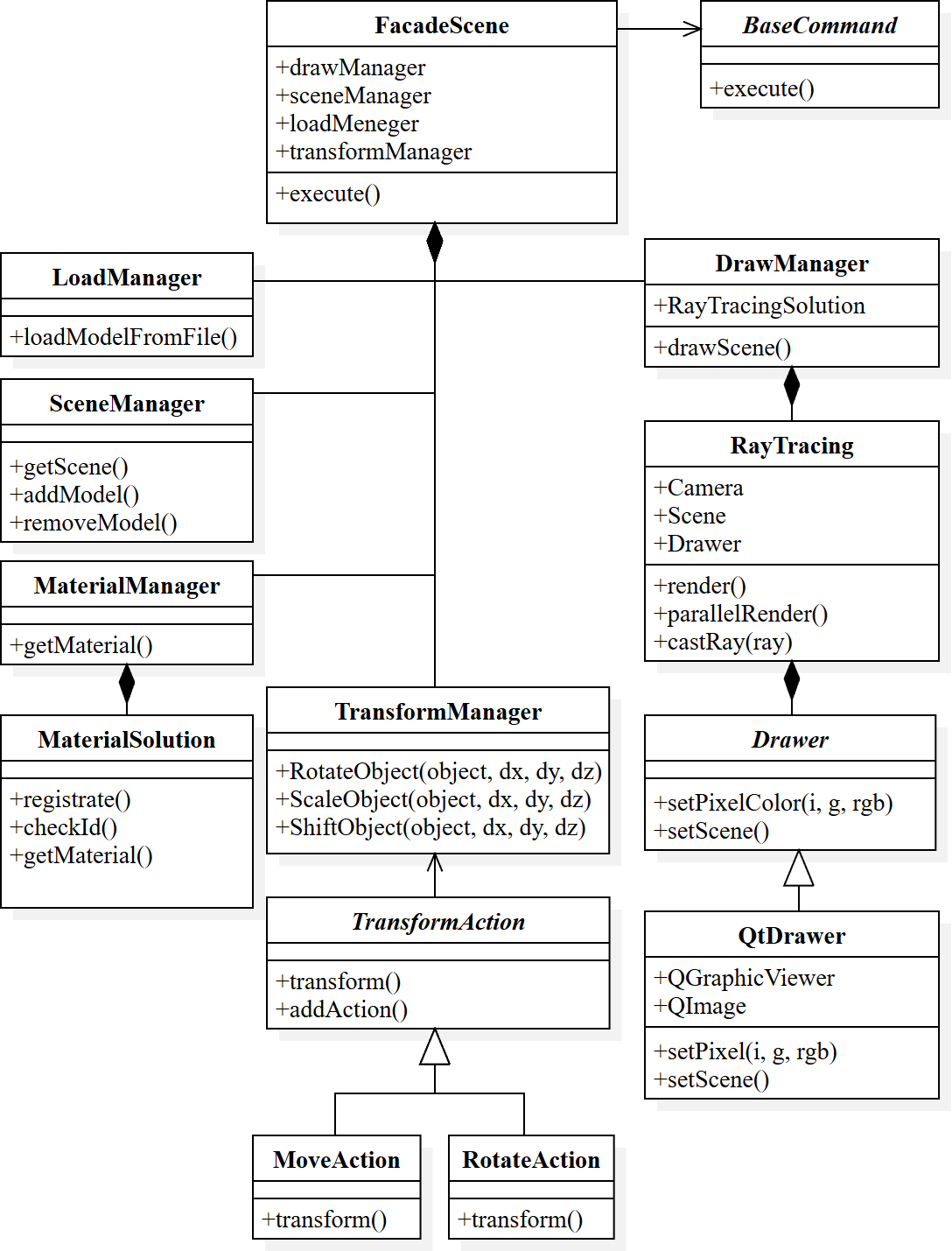


Рисунок 2.6 – Диаграмма классов

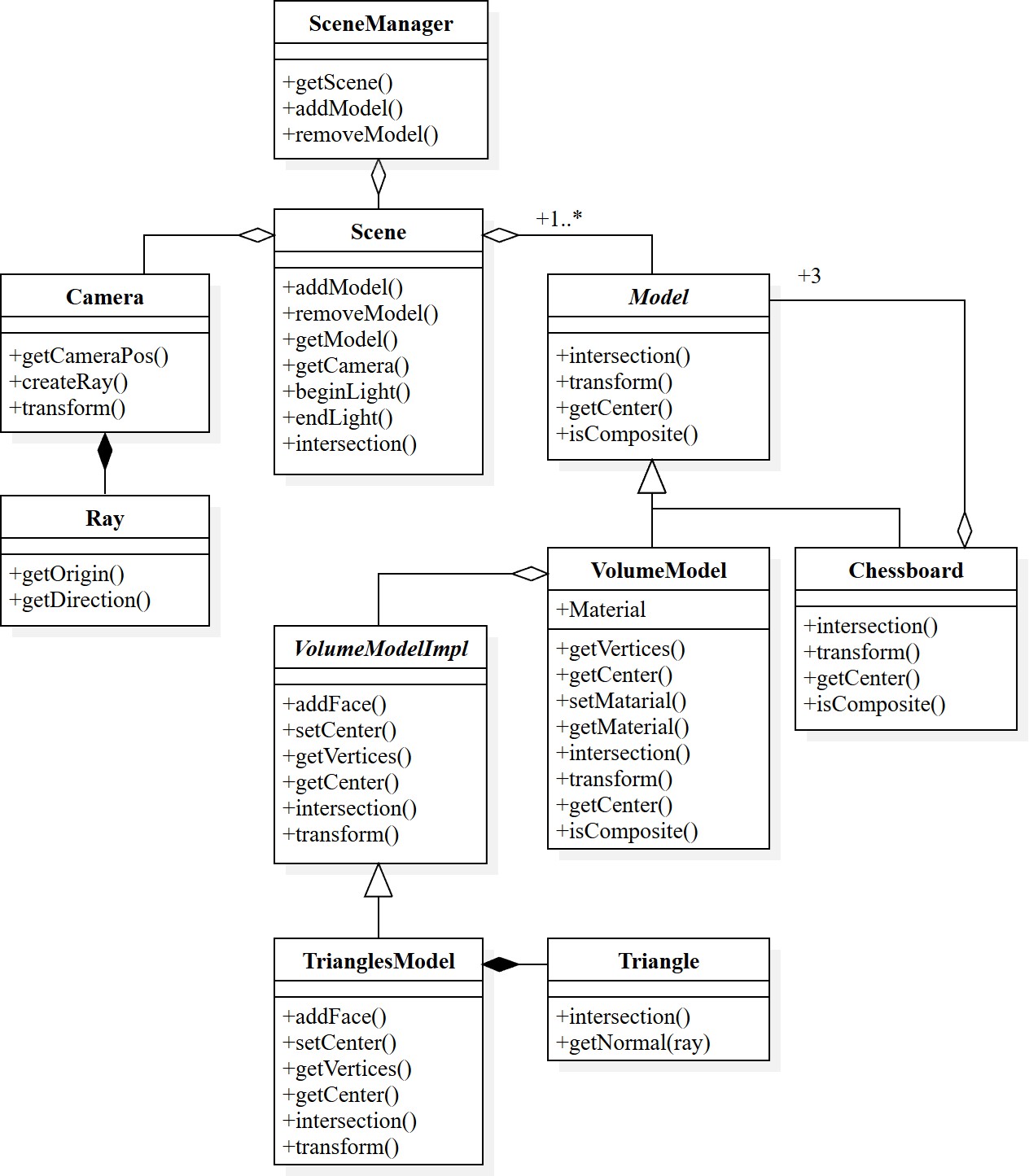


Рисунок 2.7 – Диаграмма классов

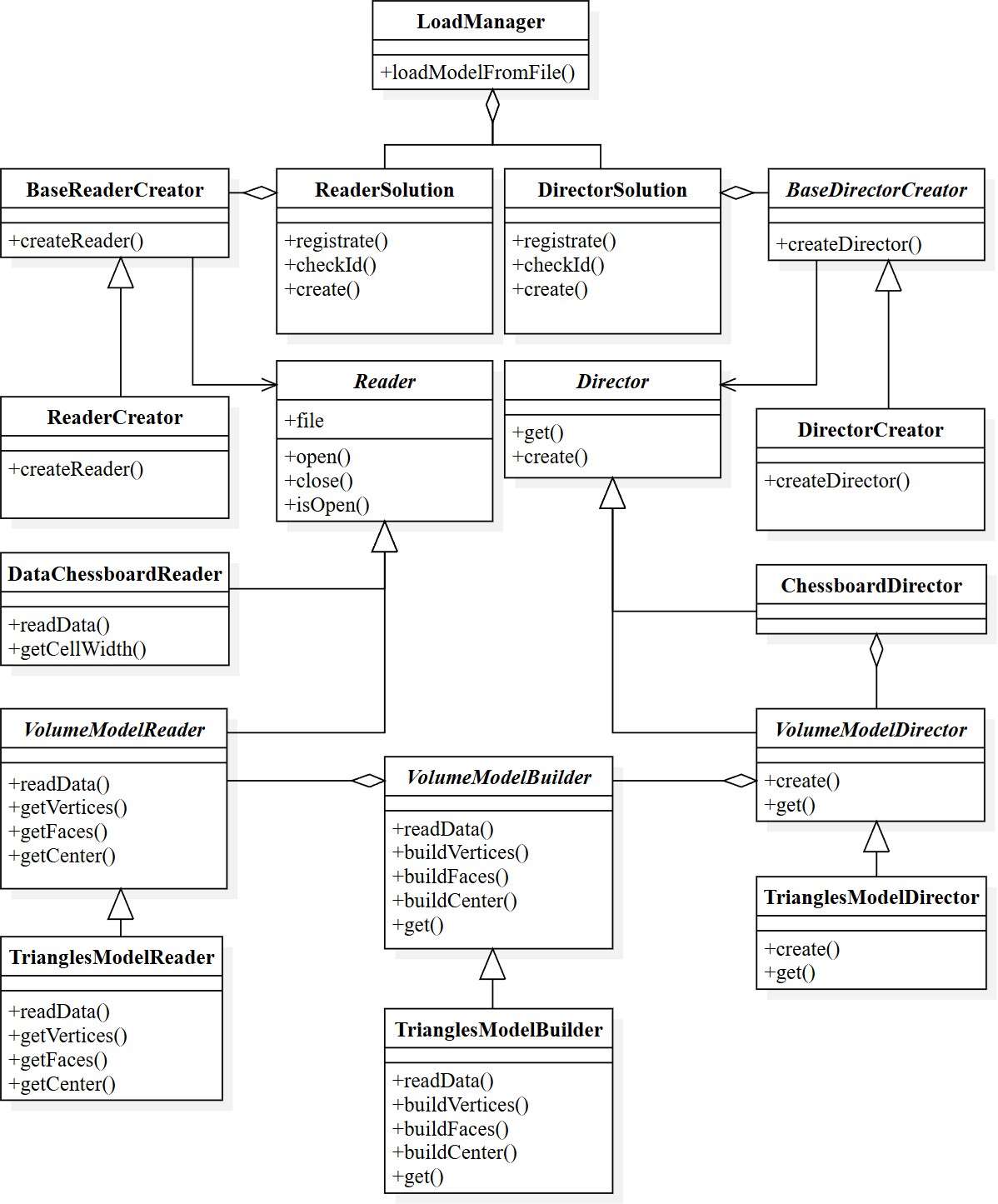


Рисунок 2.8 – Диаграмма классов

# Описание классов используемых в программе

* + - * **FacadeScene** – класс реализующий структурный паттерн «фасад». Предоставляет графическому интерфейсу пользователя унифицированный интерфейс к програмному обеспечению, реализующему взаимодействие со сценой и отрисовку кадра, с помощью паттерна «команда»;
      * **BaseCommand** – базовый класс реализующий поведенческий паттерн

«команда». Инкапсулировует запрос пользователся 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 QGraphic Viewr библиотеки Qt.
      * **MaterialManager** – класс реализующий взаимодействие с множеством материалов;
      * **MaterialSolution** – класс определяющих множество возможных матери- алов которые могут быть использованы в программе;
      * **TransformManager** – класс реализующий операции проеобразования обьектов сцены;
      * **TransformAction** – базовый класс реализующий поведенческий паттерн

«стратегия». Определяет алгоритм проеобразования обьектов сцены;

* + - * **MoveAction** – производный класс от TransformAction. Определяет алго- ритм переноса обьектов сцены;
      * **RotateAction** – производный класс от TransformAction. Определяет алгоритм вращения обьектов сцены;
      * **SceneManager** – класс реализующий взаимодействие с обьектами 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, фигуры которая может быть изображена на сцене;
      * **VolumeModel** – производный класс от Model, является элеиментом абстракции структурного паттерна «мост»;
      * **VolumeModelImpl** – базовый класс являющийся элеиментом реализации структурного паттерна «мост»;
      * **TrianglesModel** – производный класс от VolumeModelImpl, определя- ет 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 структур- ный паттерн «компоновщик». Шахматная доска состоит из трех моделей: модели определяющей деревянную основу доски и двух моделей опреде- ляющих множества черных и белых клеток;
      * **LoadManager** – класс реализующий загрузку моделей;
      * **ReaderSolution** – класс опрелеляющий используемый метод чтения файла;
      * **BaseReaderCreator** – базовый класс реализующий порождающий пат- терн «фабричный метод». Опреляет 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, представ- ленной классом TrianglesModel;
      * **DirectorSolution** – класс опрелеляющий используемый метод создания Model;
      * **BaseDirectorCreator** – базовый класс реализующий порождающий паттерн «фабричный метод». Опреляет реализацию класса Director, которая будет создана;
      * **DirectorCreator** – производный класс от BaseDirectorCreator;
      * **Director** – базовый класс реализующий порождающий паттерн «строи- тель», элемент «director». Опредеяет перацию создания фигуры (Model);
      * **ChessboardDirector** – производный класс от Director. Определяет опе- рацию создания класса шахматной доски (Chessboard);
      * **VolumeModelDirector** – базовый класс определяющий операцию созда- ния 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». Опредеяет множество операций необходимых для создания фигуры (Model);
      * **TrianglesModelBuilder** – производный класс от VolumeModelBuilder. Опредеяет множество операций необходимых для создания фигуры, пред- ставленной классом TrianglesModel;

# Вывод из конструкторской частьи

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

# Технологическая часть

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

# Средства реализации

В качестве языка программирования был выбран C++ [[10]](#_bookmark54) 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 фрейм- ворк QT [[11],](#_bookmark55) так как данный фреймфорк предоставляет инструменты для создания пользовательских интерфейсов и поддерживается язык программиро- вания C++.

Для сборки программного обеспечения использовалась утилита CMake [[12],](#_bookmark56)

так как с ee помощью возможно управлять процессом компиляции и сборки проекта, написанного на C++.

Для модульного тестирования компонент программного обеспечения был

выбран фреймворк GoogleTest [[13],](#_bookmark57) так как данный фреймфорк предоставляет инструменты для написания модульных тестов на языке C++.

# Реализации алгоритмов

В листингах [3.1-3.2](#_bookmark38) представлены реализация алгоритма обратной трассиров- ки лучей и алгоритма испускания луча.

Листинг 3.1 – Реализация алгоритма обратной трассировки лучей

v o i d Ray Tracing : : r e n d e r ( ) {

f o r ( i n t j = 0 ; j < drawer =>g et Img Height ( ) ; ++j ) { f o r ( i n t i = 0 ; i < drawer =>g et Img Width ( ) ; ++i ) {

1

2

3

|  |  |  |
| --- | --- | --- |
| 4 |  | Ray r a y = camera=>c r e a te R a y ( i , j ) ; |
| 5 |  | I n t e n s i t y i n t e n s = cast Ray ( r a y ) ; |
| 6 |  | drawer =>s e t P i x e l C o l o r ( i , j , Co l o r ( i n t e n s ) ) ; |
| 7 | } |  |
| 8 | } | |
| 9 | drawer =>s e t S c e n e ( ) ; | |
| 10 | } | |

Листинг 3.2 – Реализация алгоритма испускания луча

I n t e n s i t y Standard Ray Tracing : : cast Ray ( Ray &ray , c o n s t s i z e \_t depth

, b o o l p r i n t i n g ) c o n s t noexcept {

i f ( ! r a y . g e t D i r e c t i o n ( ) . i s N o r m a l i z e d ( ) ) r a y . g e t D i r e c t i o n ( ) . n o r m a l i z e ( ) ;

I n t e n s i t y c o l o r ( 0 , 0 , 0 ) ; i n t e r s e c t i o n \_ t i n t e r s e c t ;

i f ( scene => i n t e r s e c t i o n ( ray , i n t e r s e c t ) && depth < maxDepth ) {

Point 3 p o s L i g h t ; Vector 3 L ;

i n t e r s e c t i o n \_ t t m p I n t e r s e c t ; Vector 3 d i f f , spec ;

double d d i s t ;

f o r ( Scene : : i t e r a t o r L i g h t i t = scene =>b e g i n L i g h t ( ) ; i t != scene =>e n d L i g h t ( ) ; ++i t ) {

i f ( ( \* i t )=>g et Type ( ) == t y p e L i g h t : : POINT) {

L = ( p o s L i g h t = i n t e r s e c t . p o i n t ) . n o r m a l i z e d ( ) ;

i f ( f a b s ( i n t e r s e c t . normal . l e n g t h ( ) ) < EPS) { r e t u r n I n t e n s i t y ( 1 , 0 , 0 ) ;

}

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21 i f ( ! i n t e r s e c t . normal . i s N o r m a l i z e d ( ) )

22 i n t e r s e c t . normal . n o r m a l i z e ( ) ;

23 double nL = L . s c a l a r P r o d u c t ( i n t e r s e c t . normal ) ;

24

25 i f ( nL > 0 && ! scene => i n t e r s e c t i o n ( Ray ( i n t e r s e c t . p o i n t + 1 e=3 \* i n t e r s e c t . normal , L ) , t m p I n t e r s e c t ) ) {

26

27 d i f f = i n t e r s e c t . m a t e r i a l =>getKd ( ) \* nL ;

28

29 Vector 3 r e f l e c t L i g h t = L . r e f l e c t ( i n t e r s e c t . normal ) ;

30 double SR = r e f l e c t L i g h t . s c a l a r P r o d u c t (= r a y . g e t D i r e c t i o n ( ) ) ;

|  |  |  |
| --- | --- | --- |
| 31 |  | i f ( SR < 0 ) |
| 32 |  | SR = 0 ; |
| 33 |  | spec = i n t e r s e c t . m a t e r i a l =>g et Ks ( ) \* pow ( SR , i n t e r s e c t . m a t e r i a l =>getN ( ) ) ; |
| 34 |  |  |
| 35  36 |  | d d i s t = ( p o s L i g h t = i n t e r s e c t . p o i n t ) . l e n g t h ( ) / ( p o s L i g h t . l e n g t h ( ) + EPS) ;  i f ( d d i s t > 1 . 0 ) |
| 37 |  | d d i s t = 1 ; |
| 38 |  | e l s e i f ( d d i s t < 0 . 4 ) |
| 39 |  | d d i s t = 0 . 4 ; |
| 40 |  |  |
| 41 |  | c o l o r += ( d i f f + spec ) \* (\* i t )=> g e t I n t e n s i t y ( ) / d d i s t ; |
| 42 |  | } |
| 43 |  |  |
| 44 |  | } e l s e i f ( ( \* i t )=>g et Type ( ) == t y p e L i g h t : : AMBIENT) { |
| 45  46 |  | c o l o r += i n t e r s e c t . m a t e r i a l =>getKa ( ) \* (\* i t )=> g e t I n t e n s i t y ( ) ;  } |
| 47 | } |  |
| 48 |  |  |

49

50

51

52

53

54 }

Vector 3 r e f l e c t V e c = r a y . g e t D i r e c t i o n ( ) . r e f l e c t ( i n t e r s e c t . normal ) ;

Ray r e f l e c t R a y ( i n t e r s e c t . p o i n t + 1 e=3 \* i n t e r s e c t . normal , r e f l e c t V e c ) ;

I n t e n s i t y r e f l e c t I n t e n s i t y = cast Ray ( r e f l e c t R a y , depth + 1 ) ; c o l o r += i n t e r s e c t . m a t e r i a l =>g et Ks ( ) \* r e f l e c t I n t e n s i t y ;

|  |  |  |
| --- | --- | --- |
| 55 |  | e l s e |
| 56 |  | c o l o r = I n t e n s i t y ( 0 . 3 , 0 . 3 , 0 . 3 ) ; |
| 57 |  |  |
| 58 |  | r e t u r n c o l o r ; |
| 59 | } |  |

# Описание процесса сборки приложения

Для сборки программного обеспечения использовалась утилита CMake [[12].](#_bookmark56) Для сборки приложения необходимо в командной строке, находясь в директории проекта, выполнить следующие команды [3.3.](#_bookmark40)

Листинг 3.3 – Сборка программного обеспечения

$ cd b u i l d

$ cmake =S . .

$ cmake = b u i l d .

1

2

3

# Описание интерфейса приложения

# Тестирование

# Функциональное тестирование

# Модульное тестирование

**Вывод из технологической части**

Было реализовано програмное обеспечение и в данном разделе были пред- ставлены средства реализации, листинги основных алгоритмов, описаны процесс сборки, интерфейс приложения и методы тестирования.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. ФИДЕ Техническая комиссия. Стандарты шахматного оборудования и иг- ровых площадок, предназначенных для проведения турниров ФИДЕ. По- ложение о тай-брейках. 2015. URL: Режим доступа: [https://ruchess.ru/](https://ruchess.ru/upload/iblock/8d7/8d7dfc75a3c874ab01612272fe6c5964.pdf) [upload/iblock/8d7/8d7dfc75a3c874ab01612272fe6c5964.pdf](https://ruchess.ru/upload/iblock/8d7/8d7dfc75a3c874ab01612272fe6c5964.pdf) (дата об- ращения: 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, реалистиче- ские 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 Mo¨ ller 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 обраще- ния: 01.12.2024).
9. Peter Shirley Trevor David Black Steve Hollasch. Ray Tracing in One Weekend. 2024. URL: Режим доступа: [https://raytracing.github.io/](https://raytracing.github.io/books/RayTracingInOneWeekend.html) [books/RayTracingInOneWeekend.html](https://raytracing.github.io/books/RayTracingInOneWeekend.html) (дата обращения: 01.12.2024).
10. Standard C++ [Электронный ресурс]. URL: Режим доступа: [https://isocpp.](https://isocpp.org/) [org/](https://isocpp.org/) (дата обращения: 01.12.2024).
11. Qt | Cross-platform software development for embedded & desktop [Электрон- ный 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).

1. GoogleTest [Электронный ресурс]. URL: Режим доступа: [https://google.](https://google.github.io/googletest/advanced.html) [github.io/googletest/advanced.html](https://google.github.io/googletest/advanced.html) (дата обращения: 01.12.2024).